Workshop

DIY Static Code Analyzer

Building your own security tools with

Suchakra Sharma
Staff Scientist, ShiftLeft Inc.
Vickie Li
Developer Evangelist, ShiftLeft Inc.

nsec

May 21, 2021
Montreal, QC

Let’s Prep First

- Clone Workshop Repo

```
git clone https://github.com/joernio/workshops
cd workshops/2021-NSEC
apt install source-highlight graphviz unzip
```

- Download **Joern** and install

```
wget https://github.com/joernio/joern/releases/latest/download/joern-install.sh
chmod +x ./joern-install.sh
sudo ./joern-install.sh
```

- Download **VLC v3.0.12** source and extract in a convenient directory

```
wget http://get.videolan.org/vlc/3.0.12/vlc-3.0.12.tar.xz
tar -xvf vlc-3.0.12.tar.xz
```

Let’s Prep First

- Machine Requirements
  - At least 5-7GB free RAM (close as many browser tabs as possible, pkill slack etc)
  - At least 4 CPUs (preferably modern)
  - OpenJDK 1.8+
- Important Links
  - Joern Docs: https://docs.joern.io
  - Queries: https://queries.joern.io
  - Joern Community: https://discord.gg/AUzy45EHdf Join **#query-corner**

Suchakra Sharma

Staff Scientist, ShiftLeft Inc.

Github: tuxology
Twitter: @tuxology
Email: suchakra@shiftleft.io

PhD, Polytechnique Montréal
Loves systems, code analysis, performance analysis, hardware tracing, samosas and poutine!

Vickie Li

Developer Evangelist, ShiftLeft Inc.

Github: vickie-sl
Twitter: @vickieli7
Email: vickie@shiftleft.io

PepTalk

Why are you here?

You may have the following questions

- How do computer programs and programming languages work?
- I know some bad coding practices. How can I *mass detect* them in large codebases?
- How do static analysis tools work? Can I create my own custom static analysis tools and deploy them in CI/CD?
- *I am just here to have fun. Please don't mind me!*

You may have used or know about,

- Interactive debuggers (GDB, rr etc), SAST tools, Github, IDE to search your code

What you will learn today

- Gain ability to find vulnerabilities in large code-bases (such as VLC)
- Interactive code analysis and code exploration
- Convert your manual code auditing steps to automated analyses
- Get insights about how external libraries are being used by your own code
- Stop reliance on “vendor SAST” and roll your sleeves to find real bugs
- Some proficiency in Scala

you will be this person by EoD

Interactive Code Analysis

> Each program is its own universe, and hacking is about exploring, documenting and exploiting its rules
>
> ~ Fabian

- Debugging goes hand in hand with running code
  - AddressSanitizer, Valgrind, GDB, profilers, linters
- Many tools run, and then give results but Joern approach flips the table - we give the tool to ask questions about the code
- It’s like play-pause debugging, but for static analysis

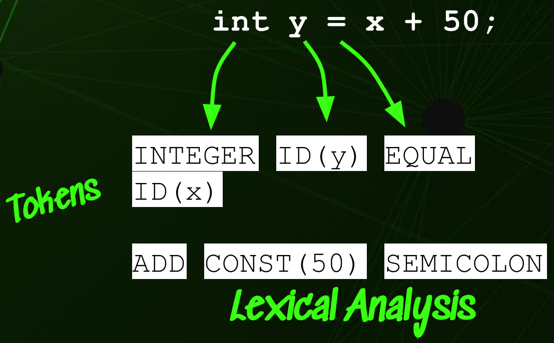
Programming Language

Fundamentals

What is even *code*?

```
int y = x + 50;
```

What is even *code*?



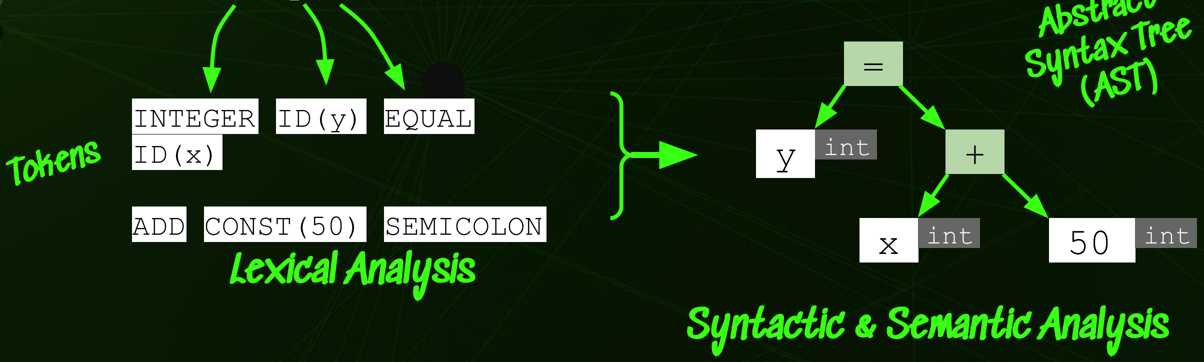
```
int y = x + 50;
```

INTEGER ID(y) EQUAL ID(x)

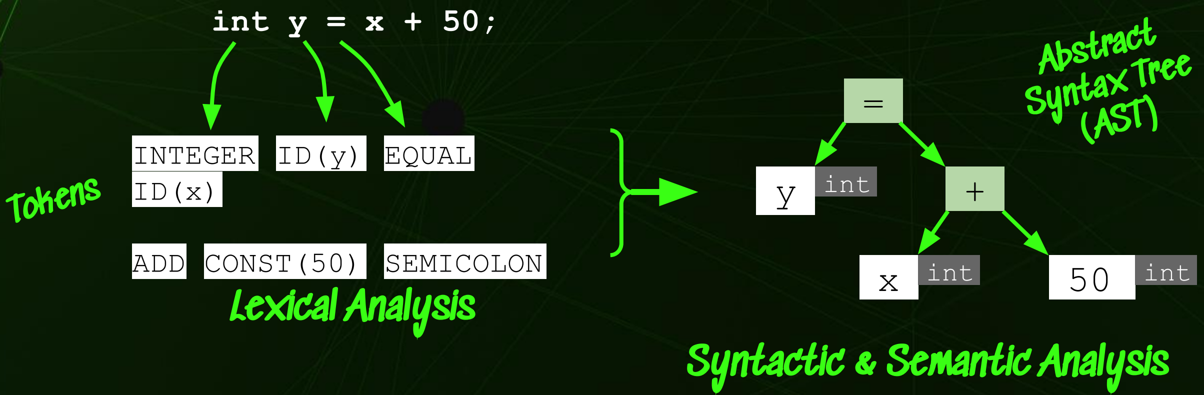
ADD CONST(50) SEMICOLON

What is even *code*?

```
int y = x + 50;
```



What is even *code*?



```
int y = x + 50;
```

Tokens

INTEGER ID(y) EQUAL ID(x)

ADD CONST(50) SEMICOLON

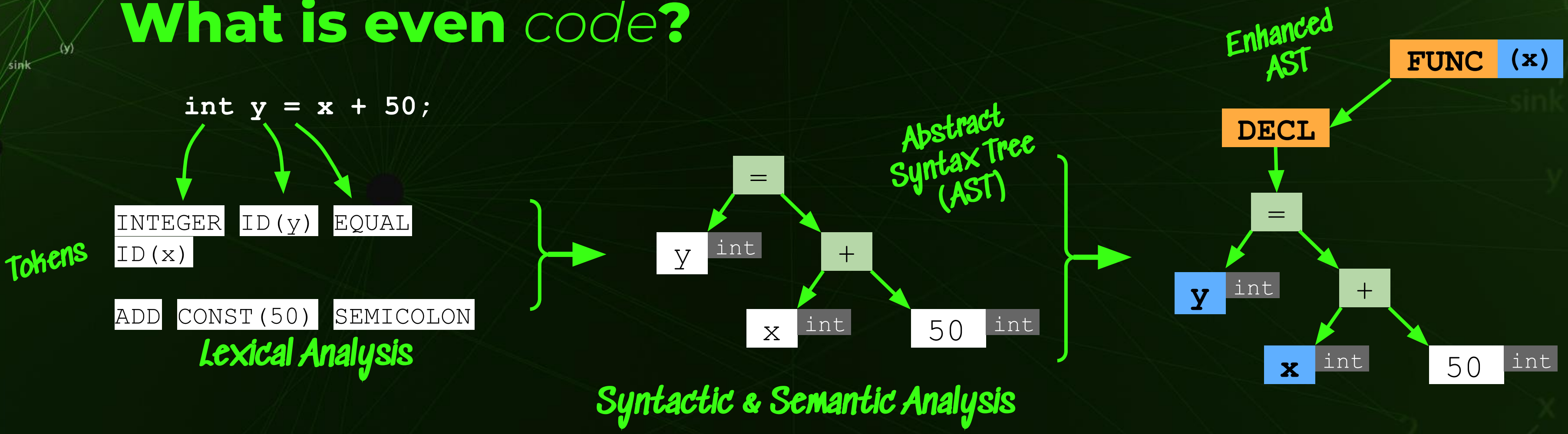
Lexical Analysis

Abstract Syntax Tree (AST)

Syntactic & Semantic Analysis

```
func(x) {
  int y = x + 50;
}
```

What is even *code*?



```
int y = x + 50;
```

Tokens

INTEGER ID(y) EQUAL ID(x)

ADD CONST(50) SEMICOLON

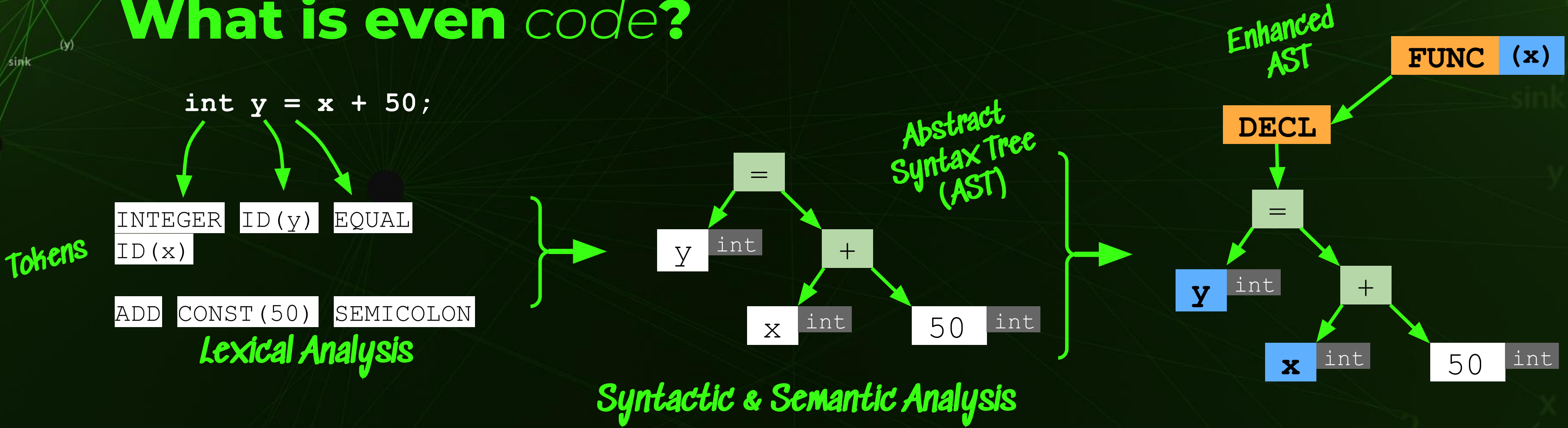
Lexical Analysis

Abstract Syntax Tree (AST)

Syntactic & Semantic Analysis

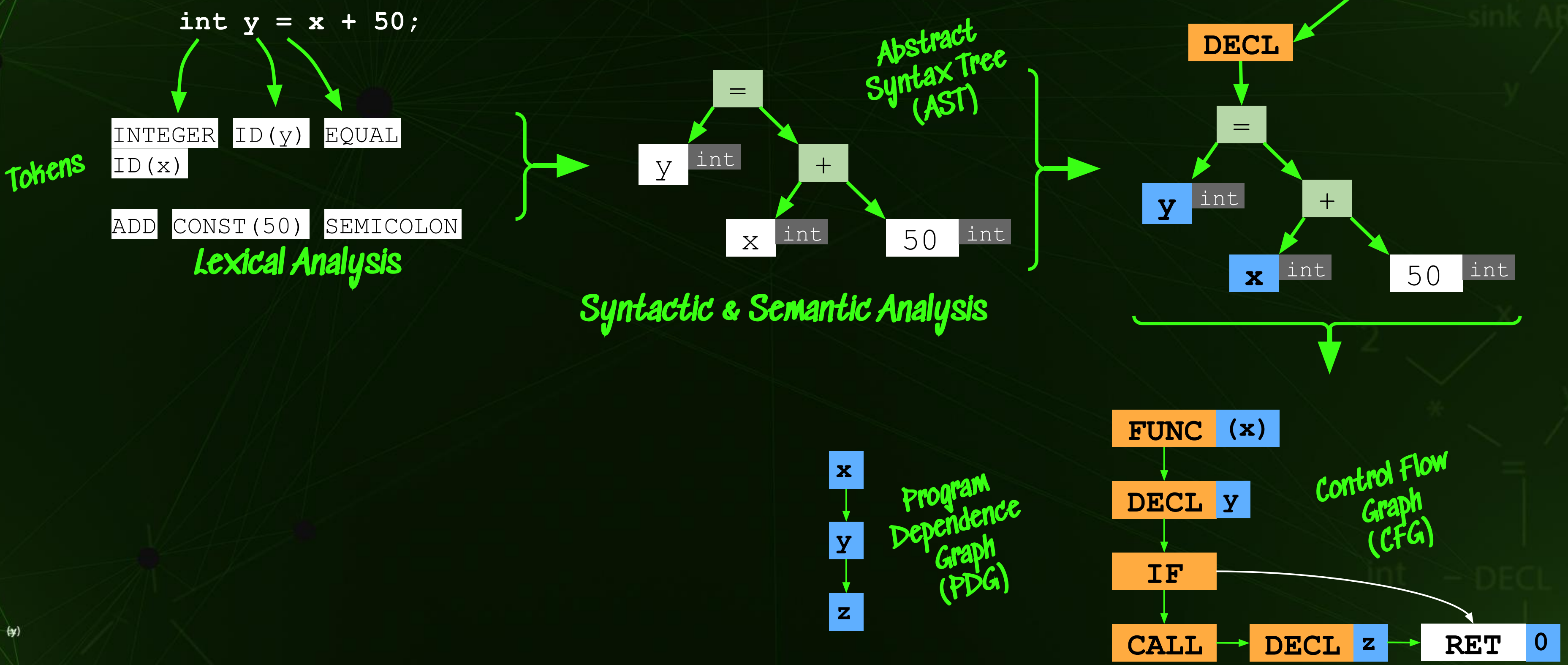
Enhanced AST

What is even *code*?



```
func(x) {
  int y = x + 50;
  if (y > 10) {
    wololo()
    z = y
  } else {
    return 0
  }
}
```

What is even *code*?



```
int y = x + 50;
```

Tokens

INTEGER ID(y) EQUAL ID(x) ADD CONST(50) SEMICOLON

Lexical Analysis

Abstract Syntax Tree (AST)

= y int + x int 50 int

Syntactic & Semantic Analysis

Enhanced AST

FUNC (x) DECL = y int + x int 50 int

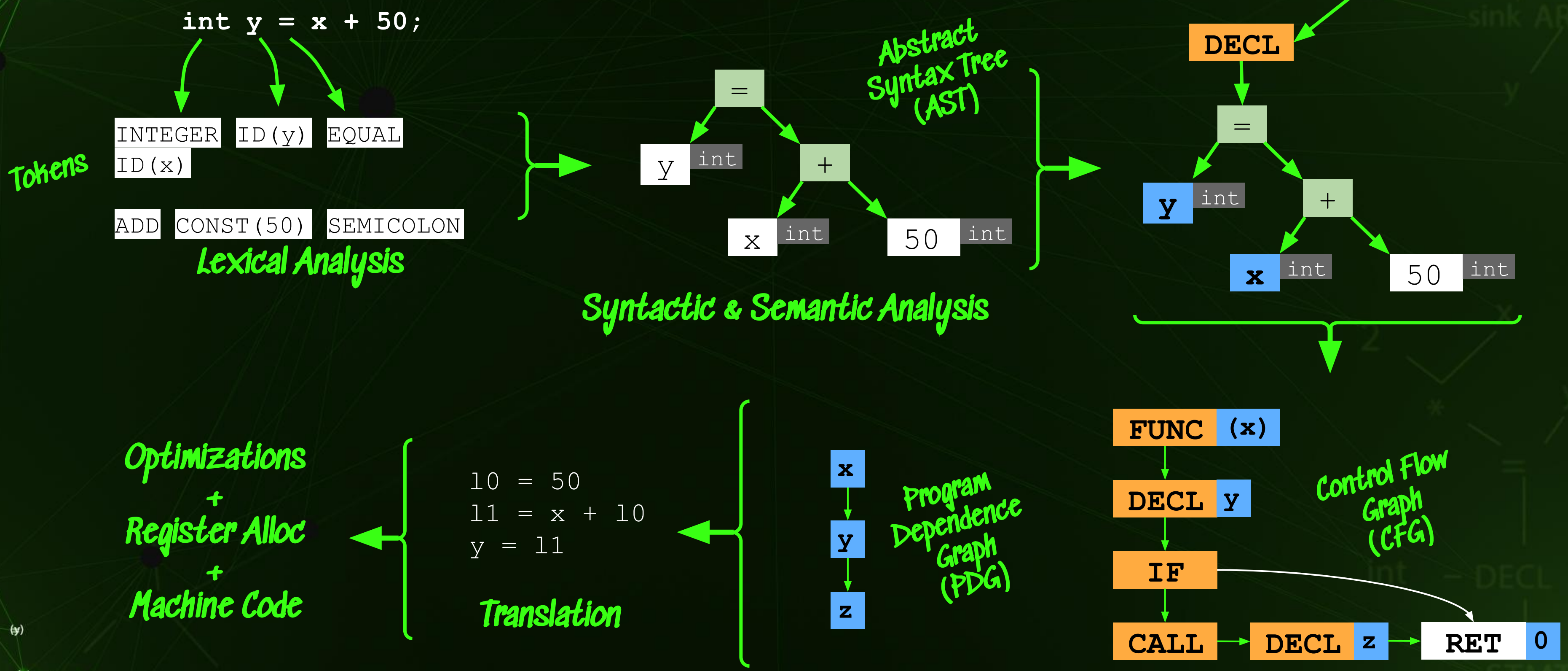
Control Flow Graph (CFG)

FUNC (x) DECL y IF CALL DECL z RET 0

Program Dependence Graph (PDG)

x y z

What is even *code*?



```
int y = x + 50;
```

Tokens

INTEGER ID(y) EQUAL ID(x)

ADD CONST(50) SEMICOLON

Lexical Analysis

Abstract Syntax Tree (AST)

Syntactic & Semantic Analysis

Enhanced AST

Control Flow Graph (CFG)

Program Dependence Graph (PDG)

```
l0 = 50
l1 = x + l0
y = l1
```

Translation

Optimizations
+
Register Alloc
+
Machine Code

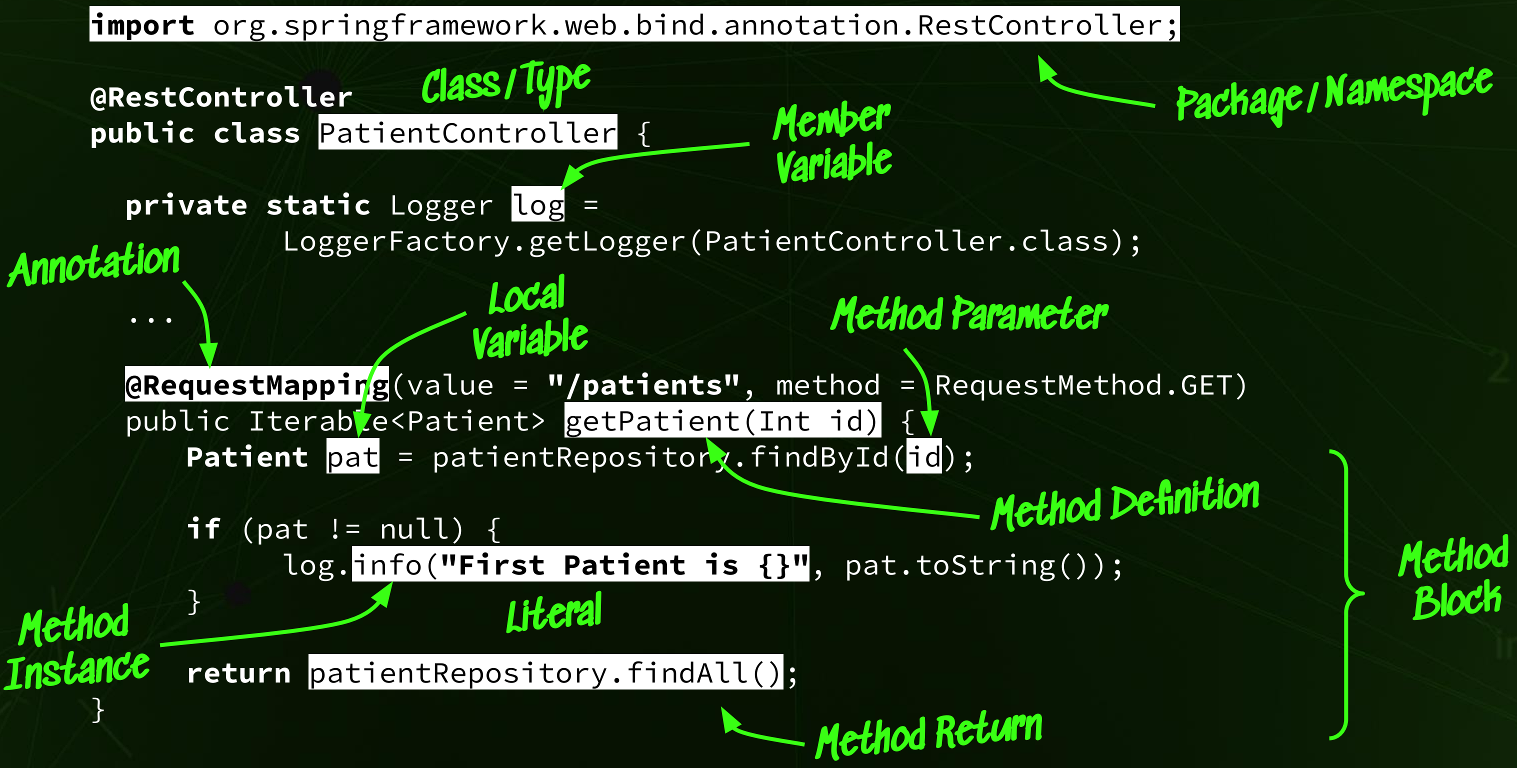
Building Blocks of Code

```
import org.springframework.web.bind.annotation.RestController;

@RestController
public class PatientController {

  private static Logger log =
           LoggerFactory.getLogger(PatientController.class);

  ...

  @RequestMapping(value = "/patients", method = RequestMethod.GET)
  public Iterable<Patient> getPatient(Int id) {
      Patient pat = patientRepository.findById(id);

      if (pat != null) {
           log.info("First Patient is {}", pat.toString());
      }

      return patientRepository.findAll();
}
```

Building Blocks of Code

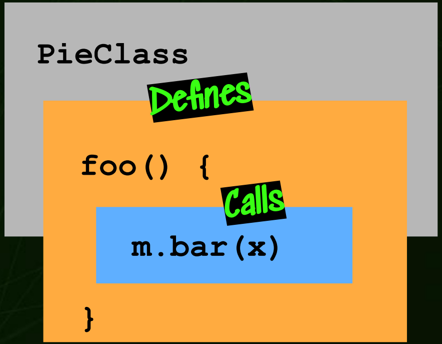
```
import org.springframework.web.bind.annotation.RestController;

@RestController
public class PatientController {

  private static Logger log =
           LoggerFactory.getLogger(PatientController.class);

  ...

  @RequestMapping(value = "/patients", method = RequestMethod.GET)
  public Iterable<Patient> getPatient(Int id) {
     Patient pat = patientRepository.findById(id);

     if (pat != null) {
           log.info("First Patient is {}", pat.toString());
     }

     return patientRepository.findAll();
}
```

Building Blocks of Code

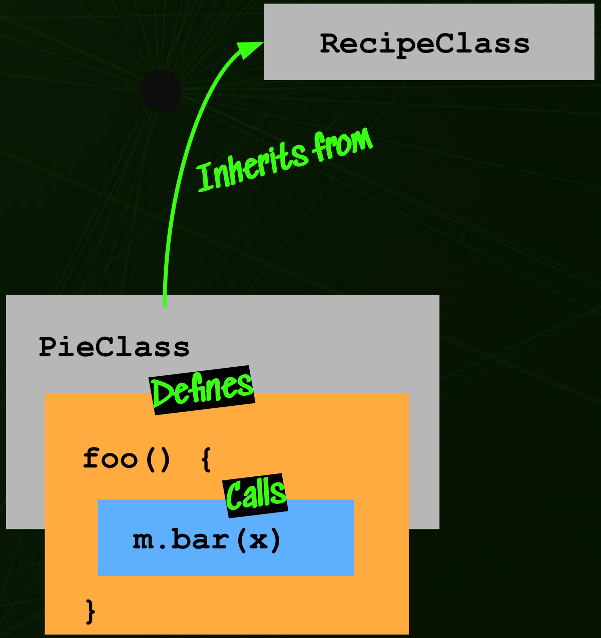


```
import org.springframework.web.bind.annotation.RestController;

@RestController
public class PatientController {

  private static Logger log =
           LoggerFactory.getLogger(PatientController.class);

  ...

  @RequestMapping(value = "/patients", method = RequestMethod.GET)
  public Iterable<Patient> getPatient(Int id) {
     Patient pat = patientRepository.findById(id);

     if (pat != null) {
           log.info("First Patient is {}", pat.toString());
     }

     return patientRepository.findAll();
}
```

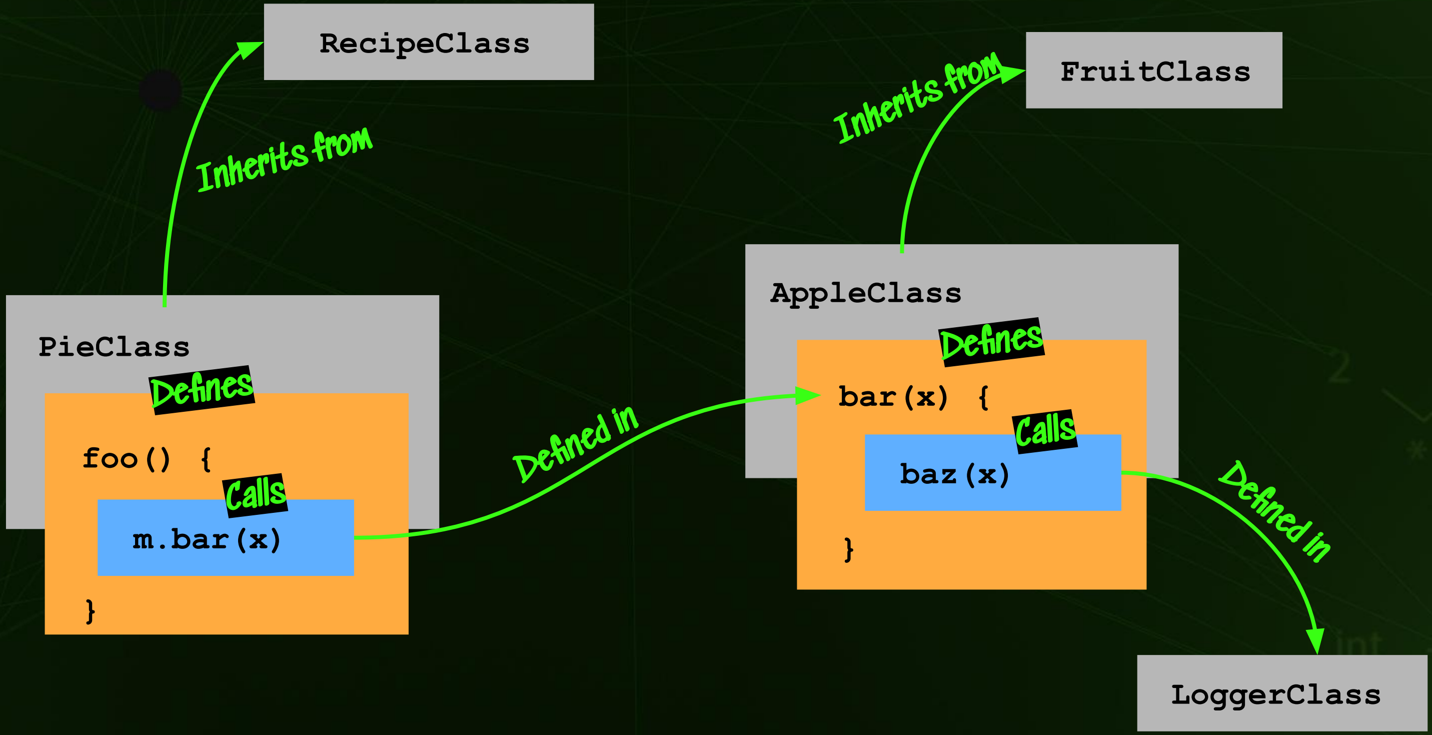
Building Blocks of Code



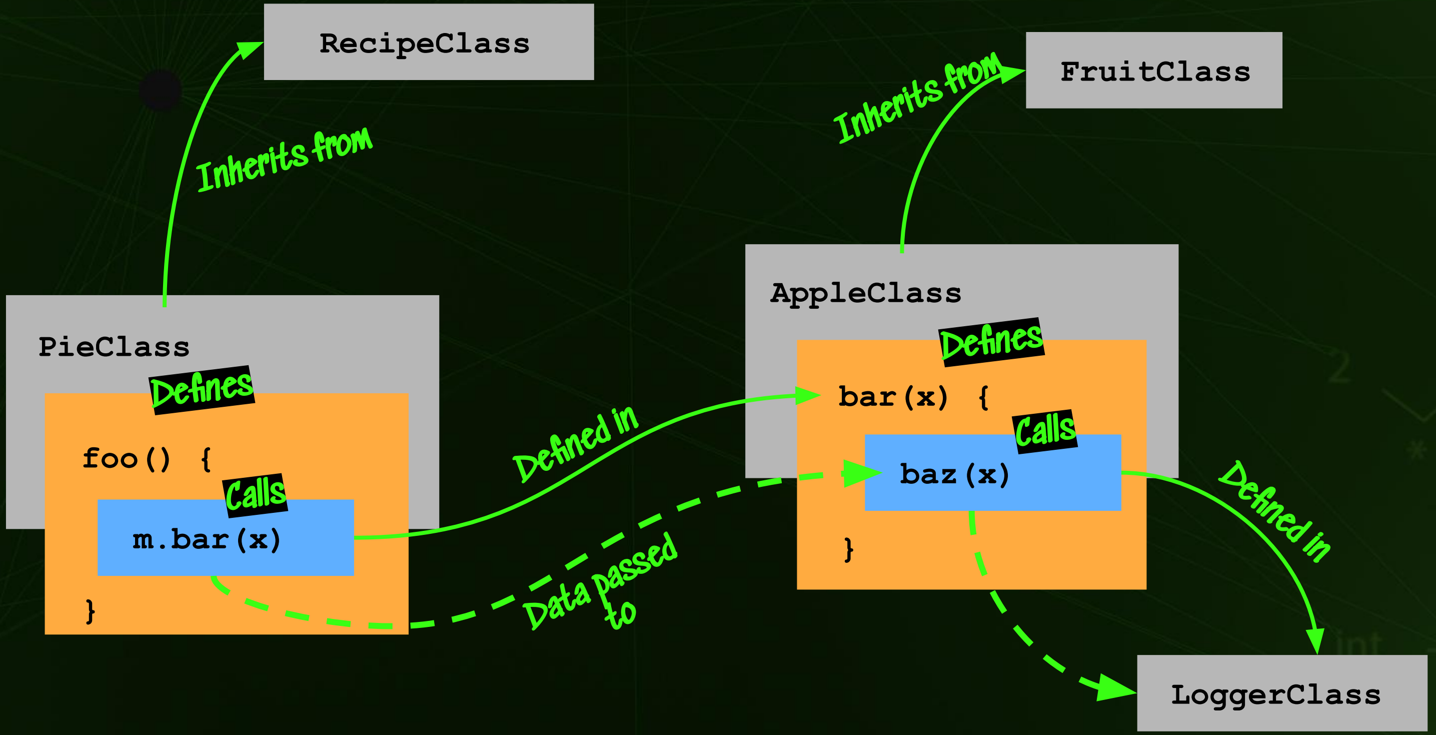
Building Blocks of Code



Building Blocks of Code

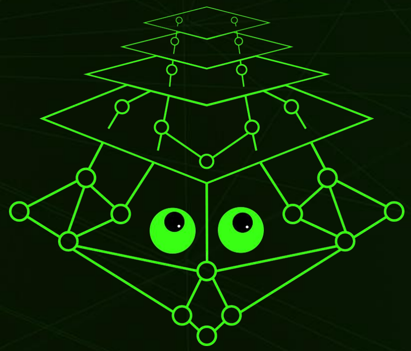


Building Blocks of Code



ALL THE CODE IS A GRAPH

If we think in graphs while coding, we should think in graphs while debugging

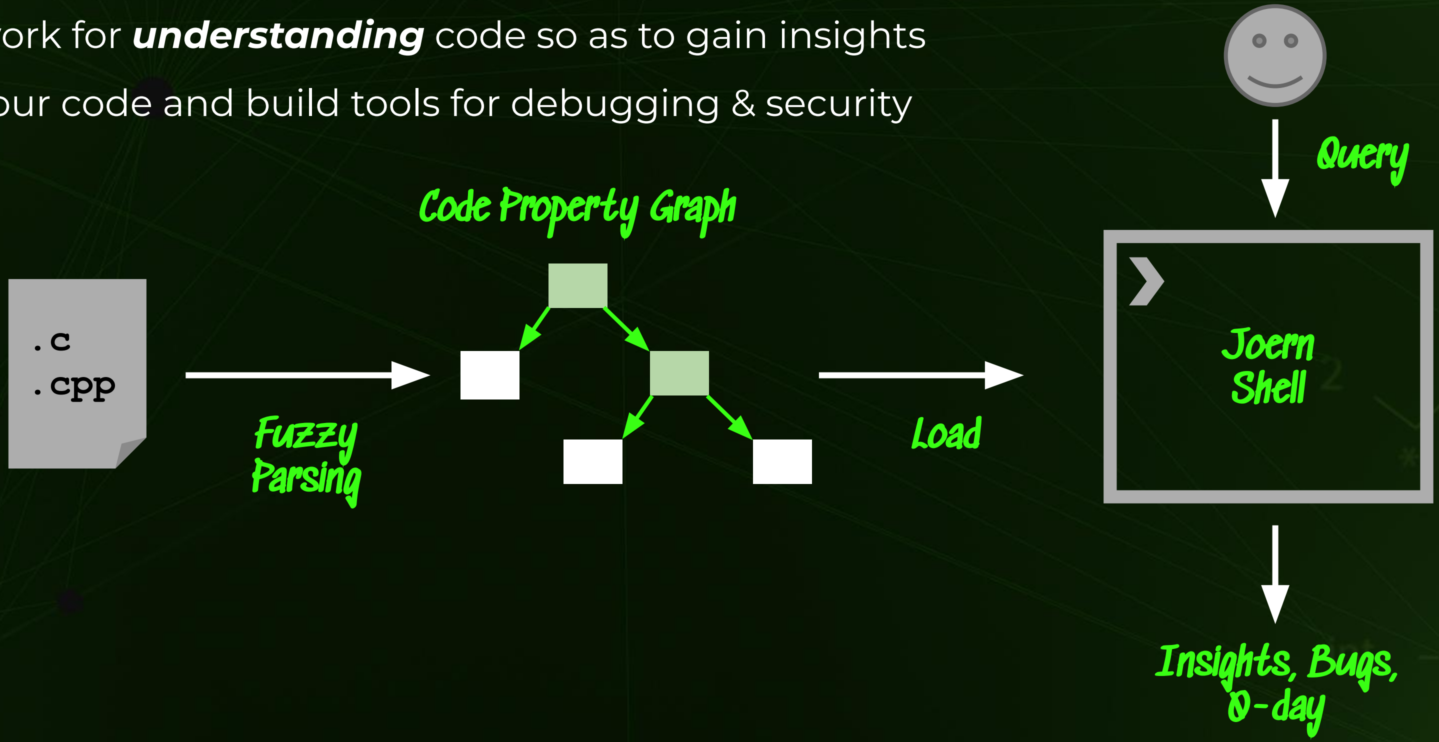


JOERN

[Yo! Urn]

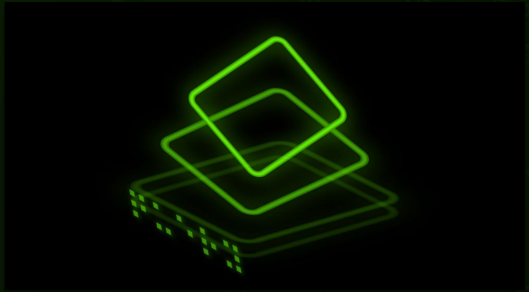
What is Joern?

Framework for ***understanding*** code so as to gain insights about your code and build tools for debugging & security



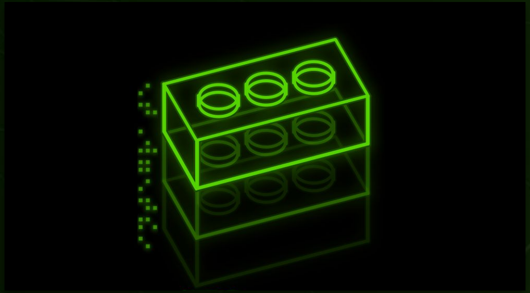
What is Joern?

Query



Automate

Integrate



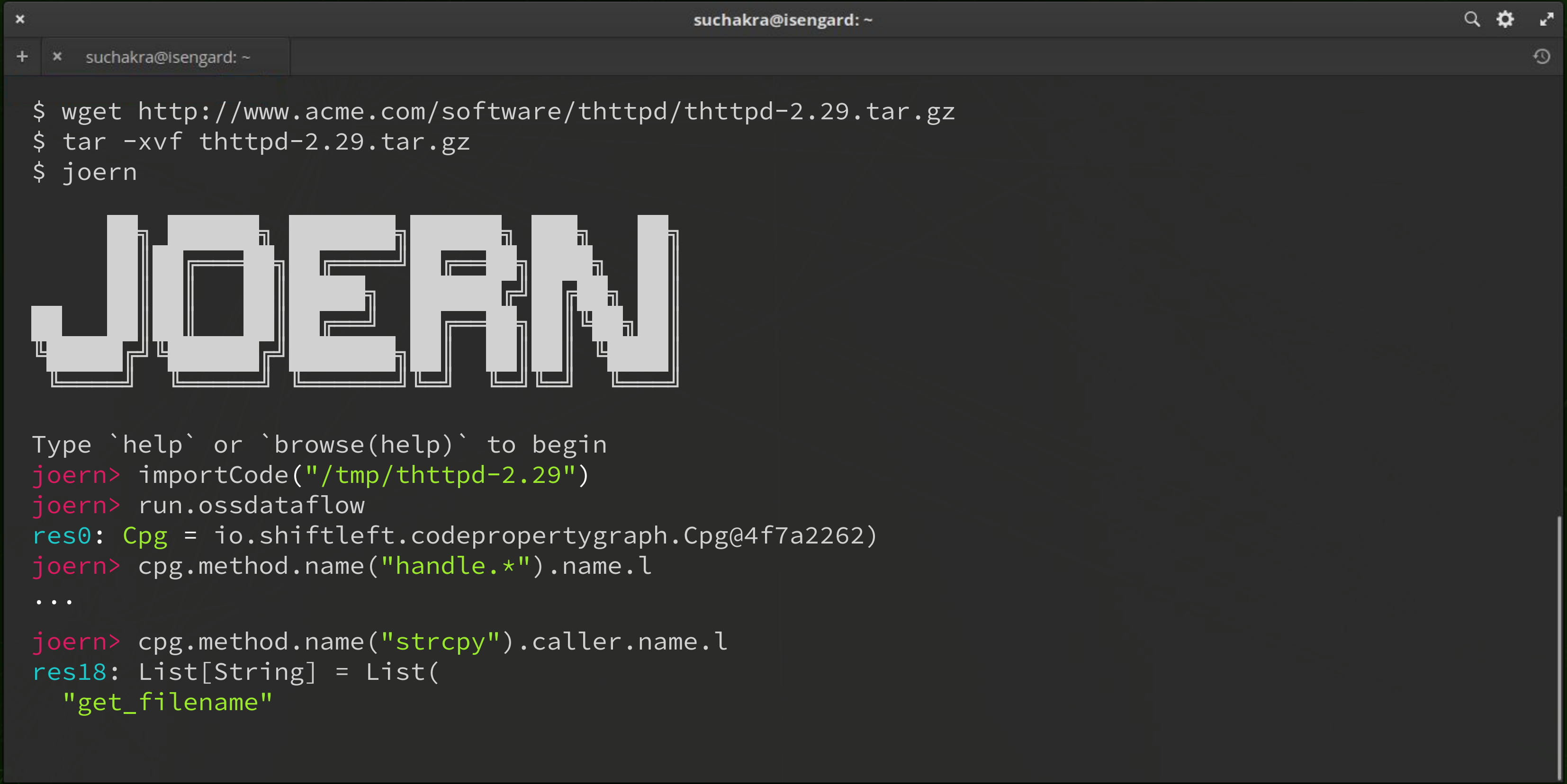
Ask questions on an interactive shell, iterate quickly ➔ Convert those questions to a recipe. Run across large codebases ➔ Take the recipe and integrate in your security pipeline or tools

Module 1

Code Navigation & Insights

Module 1

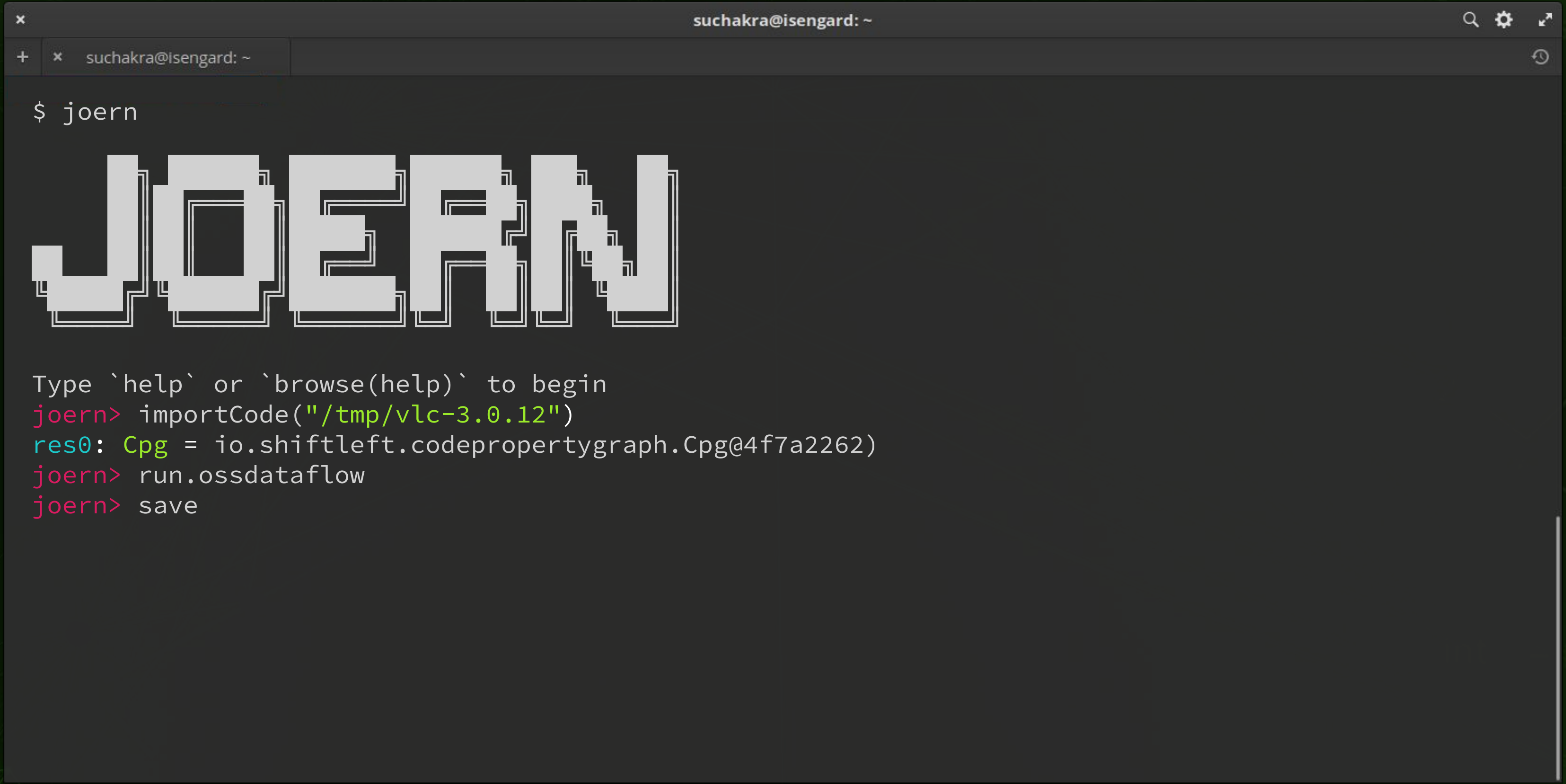
1. Quickstart



```
$ wget http://www.acme.com/software/thttpd/thttpd-2.29.tar.gz
$ tar -xvf thttpd-2.29.tar.gz
$ joern

Type `help` or `browse(help)` to begin
joern> importCode("/tmp/thttpd-2.29")
joern> run.ossdataflow
res0: Cpg = io.shiftleft.codepropertygraph.Cpg@4f7a2262)
joern> cpg.method.name("handle.*").name.l
...
joern> cpg.method.name("strcpy").caller.name.l
res18: List[String] = List(
  "get_filename"
```

Module 1

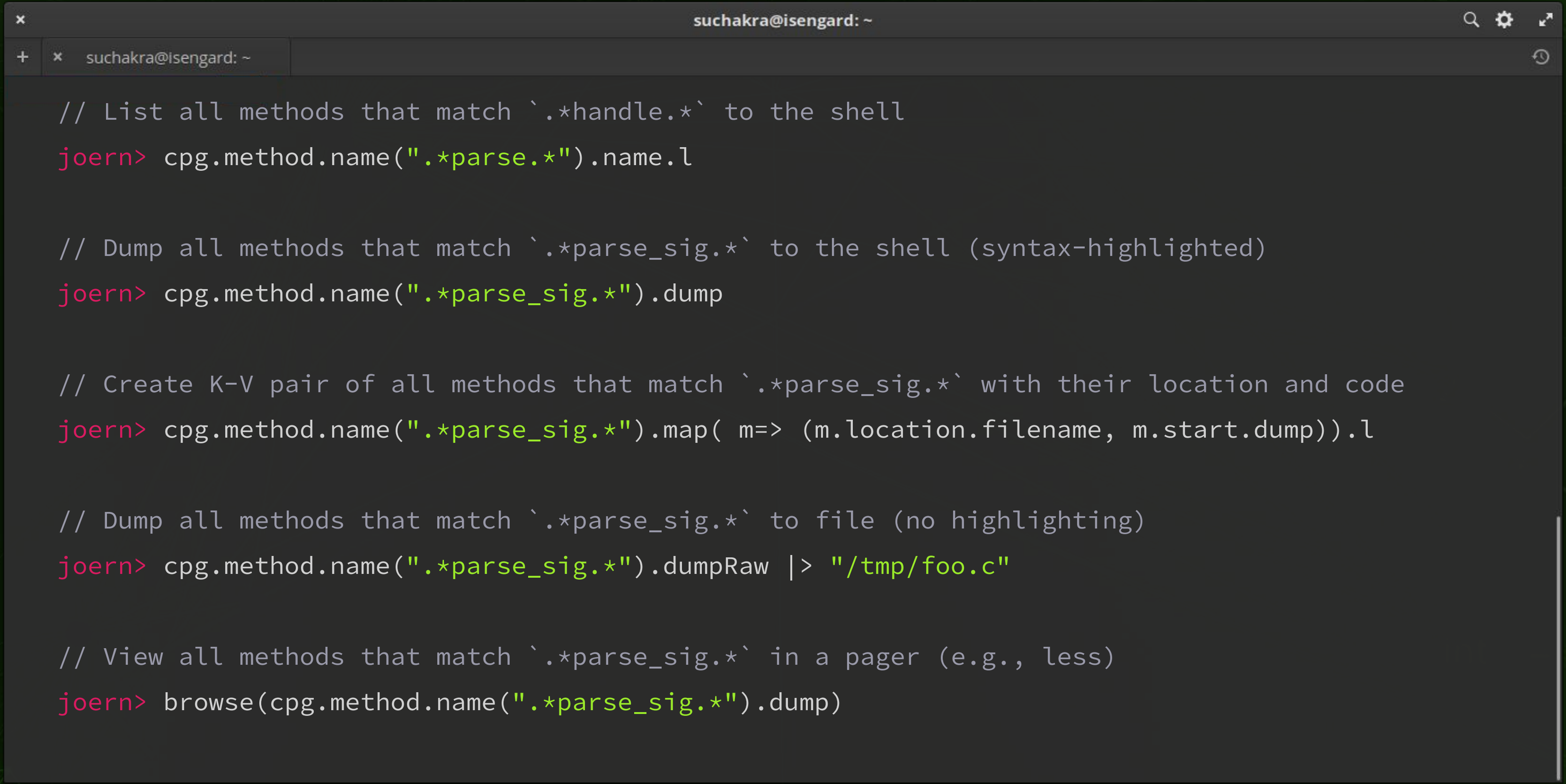
1. Parsing and Generating a CPG (VLC v3.0.12)



```
$ joern

Type `help` or `browse(help)` to begin
joern> importCode("/tmp/vlc-3.0.12")
res0: Cpg = io.shiftleft.codepropertygraph.Cpg@4f7a2262)
joern> run.ossdataflow
joern> save
```

Module 1

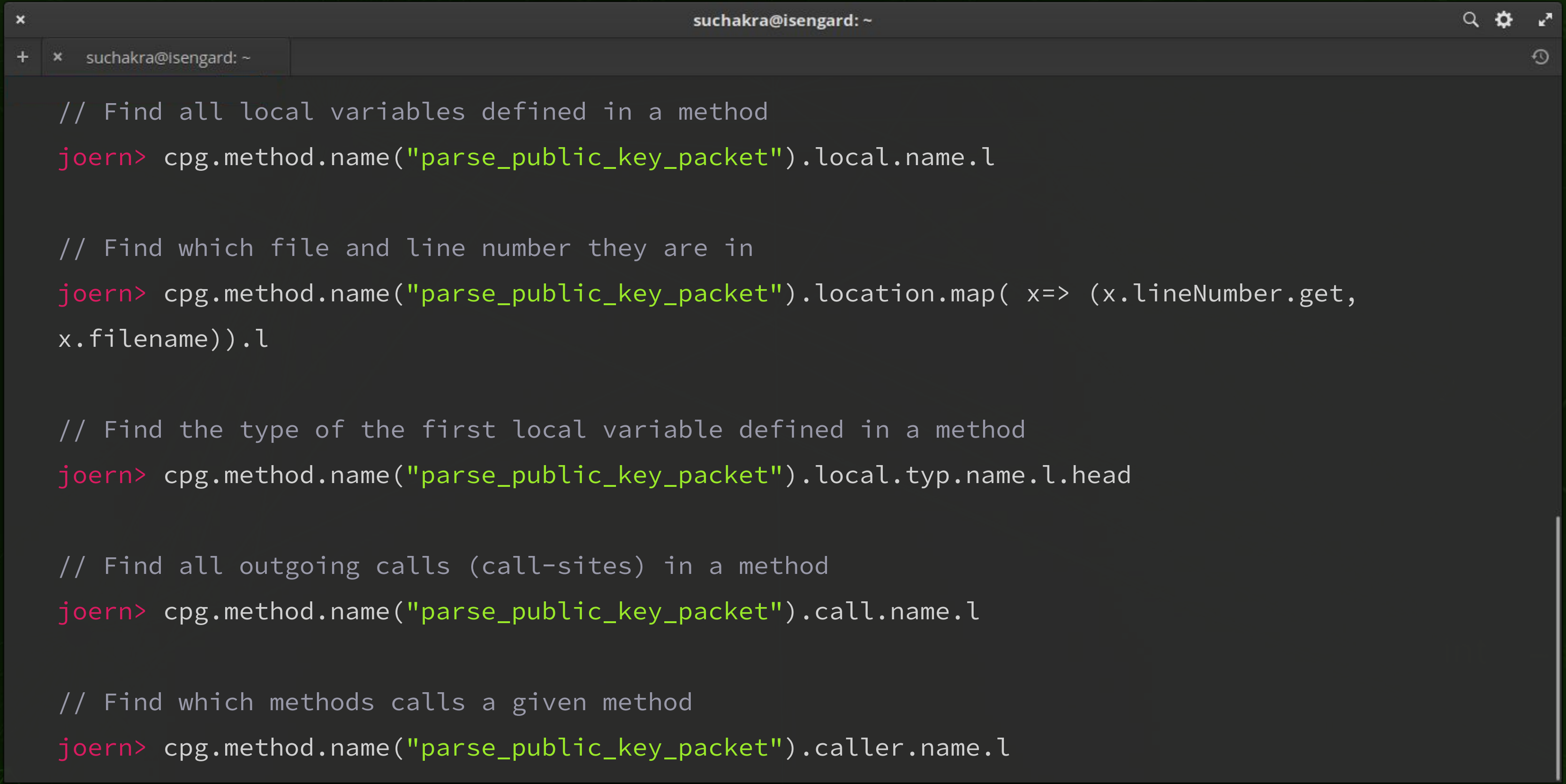
2. Basic Navigation - Methods



```
// List all methods that match `.*handle.*` to the shell
joern> cpg.method.name(".*parse.*").name.l

// Dump all methods that match `.*parse_sig.*` to the shell (syntax-highlighted)
joern> cpg.method.name(".*parse_sig.*").dump

// Create K-V pair of all methods that match `.*parse_sig.*` with their location and code
joern> cpg.method.name(".*parse_sig.*").map( m=> (m.location.filename, m.start.dump)).l

// Dump all methods that match `.*parse_sig.*` to file (no highlighting)
joern> cpg.method.name(".*parse_sig.*").dumpRaw |> "/tmp/foo.c"

// View all methods that match `.*parse_sig.*` in a pager (e.g., less)
joern> browse(cpg.method.name(".*parse_sig.*").dump)
```

Module 1

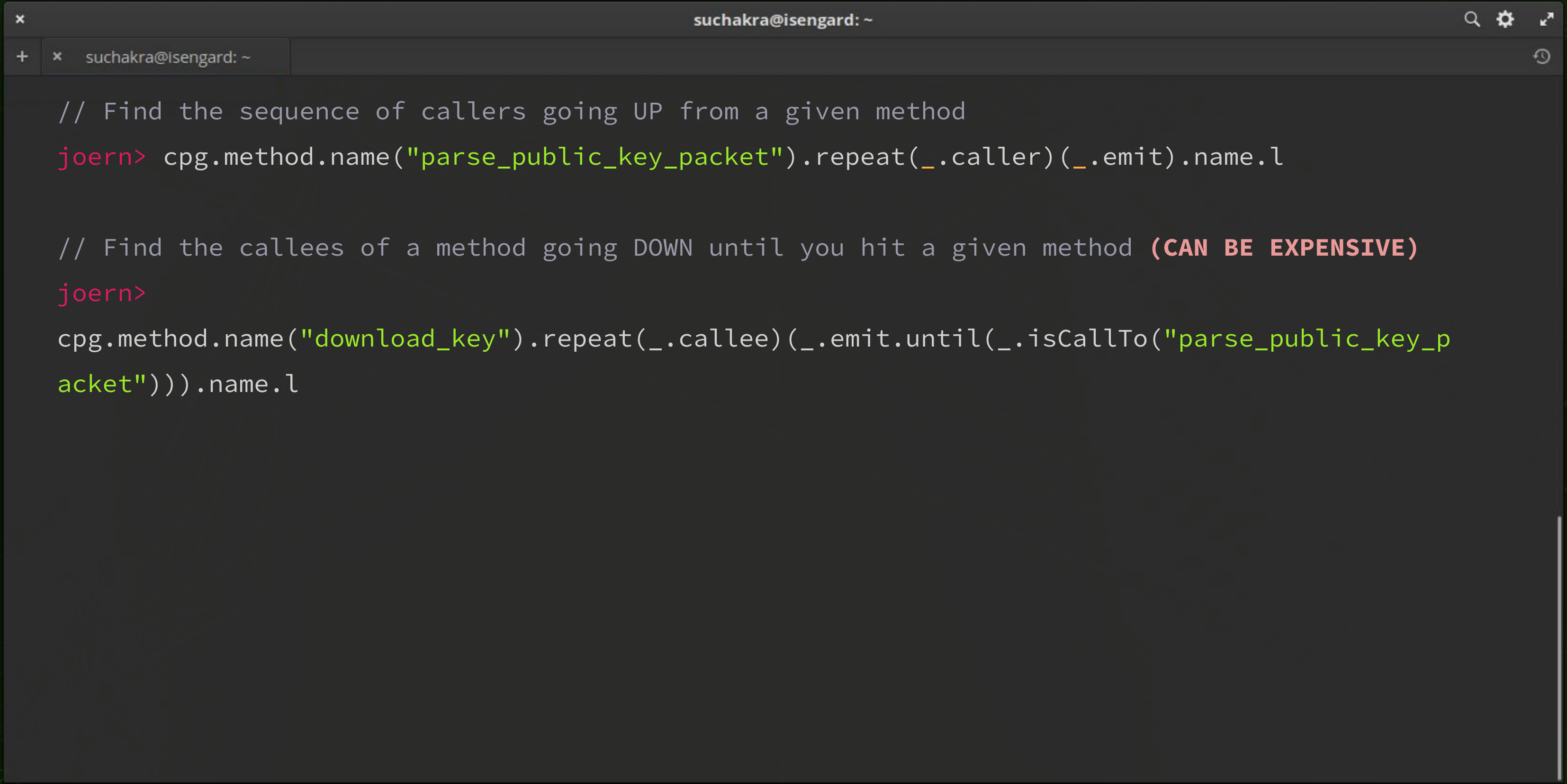
2. Basic Navigation - Methods



```
// Find all local variables defined in a method
joern> cpg.method.name("parse_public_key_packet").local.name.l

// Find which file and line number they are in
joern> cpg.method.name("parse_public_key_packet").location.map( x=> (x.lineNumber.get,
x.filename)).l

// Find the type of the first local variable defined in a method
joern> cpg.method.name("parse_public_key_packet").local.typ.name.l.head

// Find all outgoing calls (call-sites) in a method
joern> cpg.method.name("parse_public_key_packet").call.name.l

// Find which methods calls a given method
joern> cpg.method.name("parse_public_key_packet").caller.name.l
```

Module 1

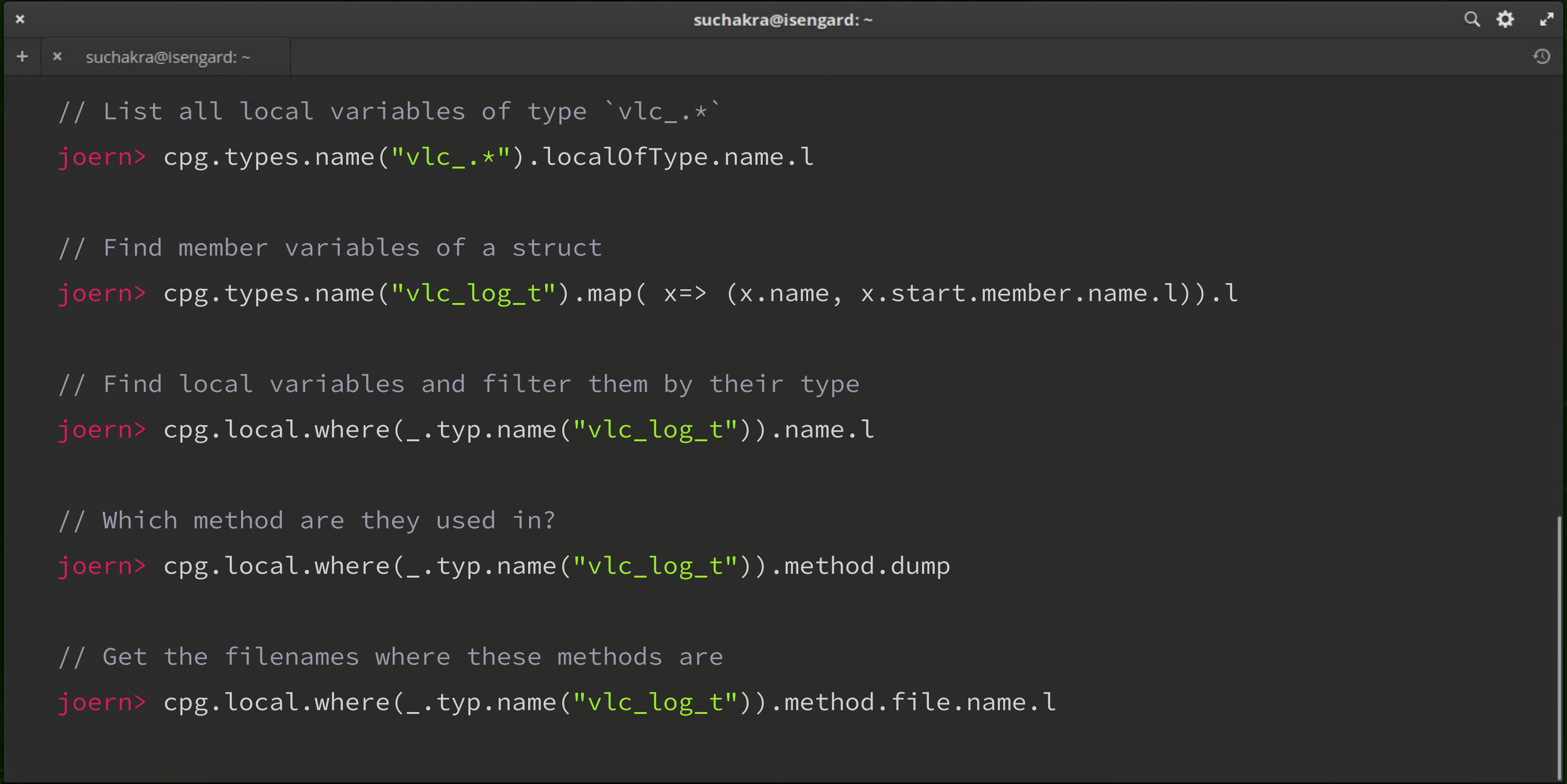
2. Basic Navigation - Repeating Graph Traversals



```
// Find the sequence of callers going UP from a given method
joern> cpg.method.name("parse_public_key_packet").repeat(_.caller)(_.emit).name.l

// Find the callees of a method going DOWN until you hit a given method (CAN BE EXPENSIVE)
joern>
cpg.method.name("download_key").repeat(_.callee)(_.emit.until(_.isCallTo("parse_public_key_p
acket"))).name.l
```

Module 1

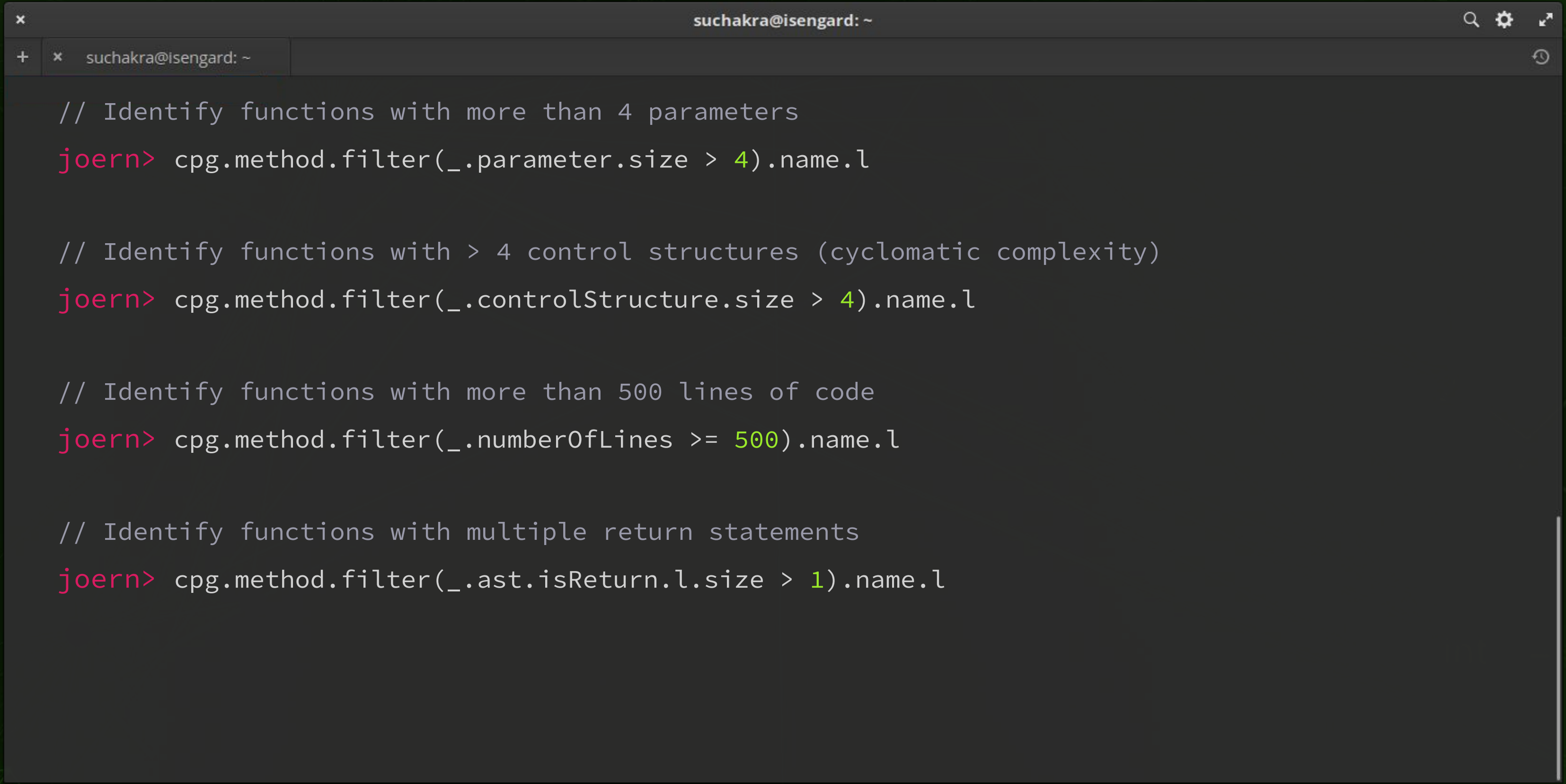
3. Basic Navigation - Types, Variables and Filtering



```
// List all local variables of type `vlc_.*`
joern> cpg.types.name("vlc_.*").localOfType.name.l

// Find member variables of a struct
joern> cpg.types.name("vlc_log_t").map( x=> (x.name, x.start.member.name.l)).l

// Find local variables and filter them by their type
joern> cpg.local.where(_.typ.name("vlc_log_t")).name.l

// Which method are they used in?
joern> cpg.local.where(_.typ.name("vlc_log_t")).method.dump

// Get the filenames where these methods are
joern> cpg.local.where(_.typ.name("vlc_log_t")).method.file.name.l
```

Module 1

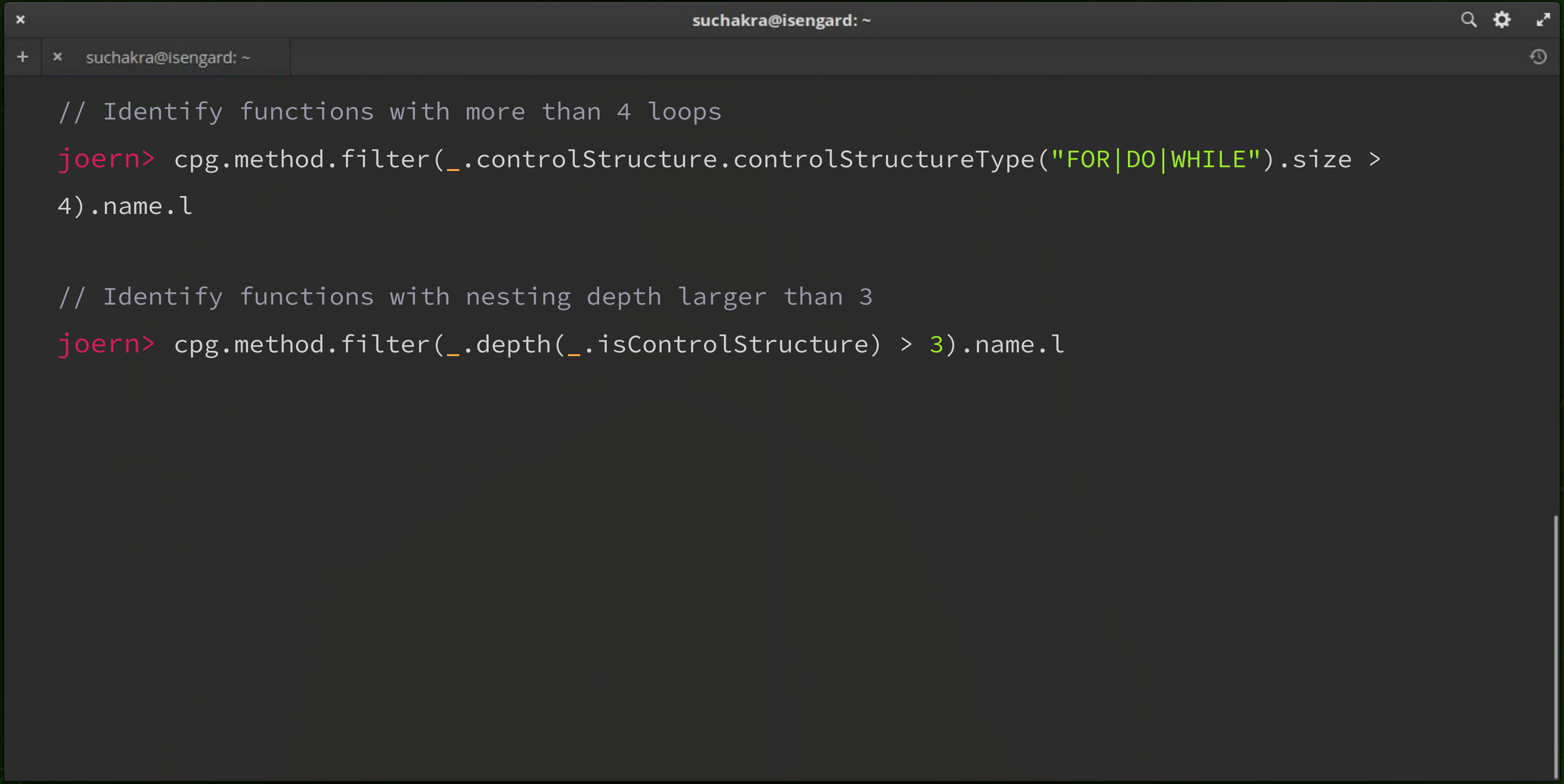
4. Basic Insights - Code Complexity



```
// Identify functions with more than 4 parameters
joern> cpg.method.filter(_.parameter.size > 4).name.l

// Identify functions with > 4 control structures (cyclomatic complexity)
joern> cpg.method.filter(_.controlStructure.size > 4).name.l

// Identify functions with more than 500 lines of code
joern> cpg.method.filter(_.numberOfLines >= 500).name.l

// Identify functions with multiple return statements
joern> cpg.method.filter(_.ast.isReturn.l.size > 1).name.l
```

Module 1

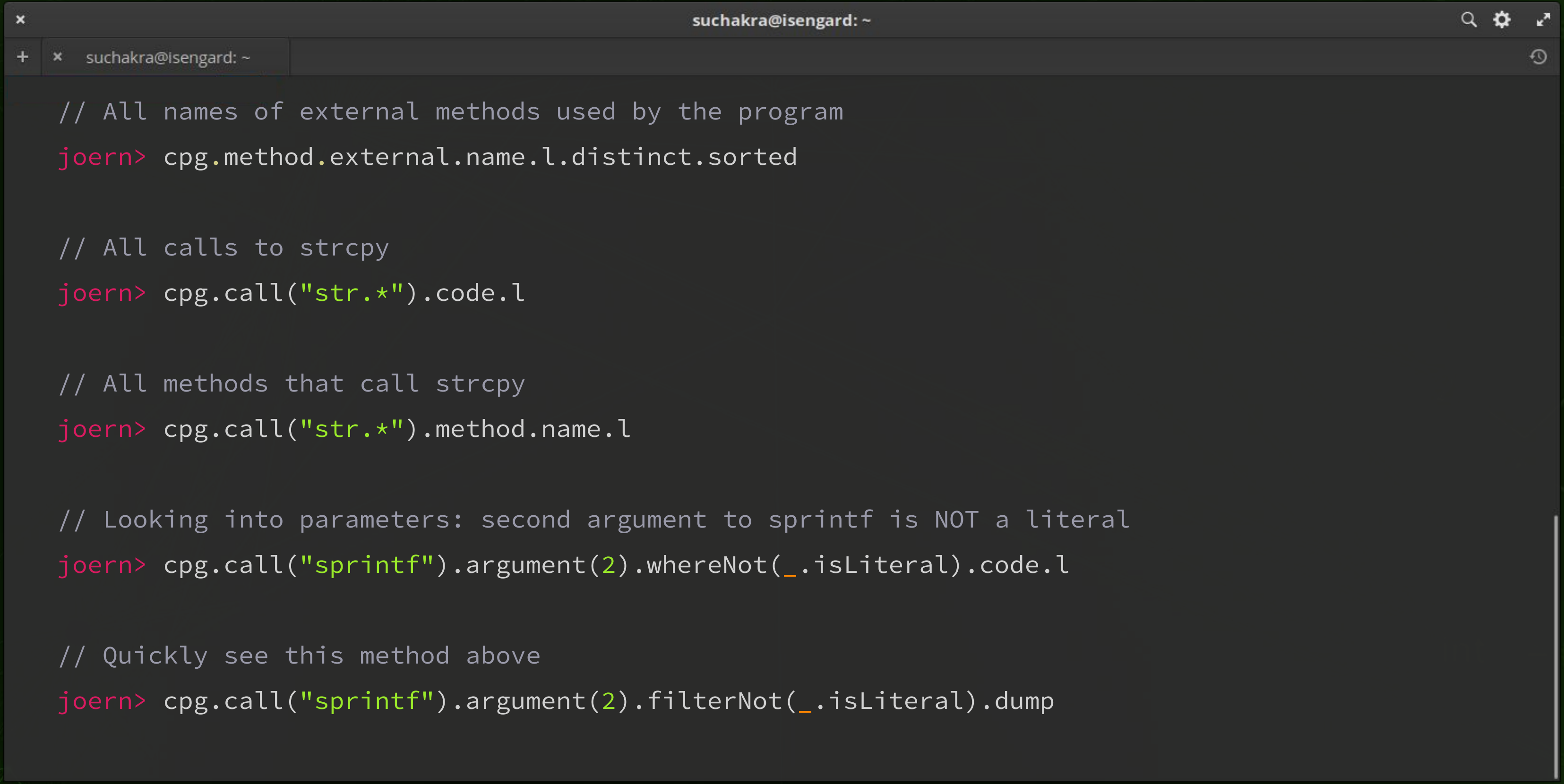
4. Basic Insights - Code Complexity



```
// Identify functions with more than 4 loops
joern> cpg.method.filter(_.controlStructure.controlStructureType("FOR|DO|WHILE").size >
4).name.l

// Identify functions with nesting depth larger than 3
joern> cpg.method.filter(_.depth(_.isControlStructure) > 3).name.l
```

Module 1

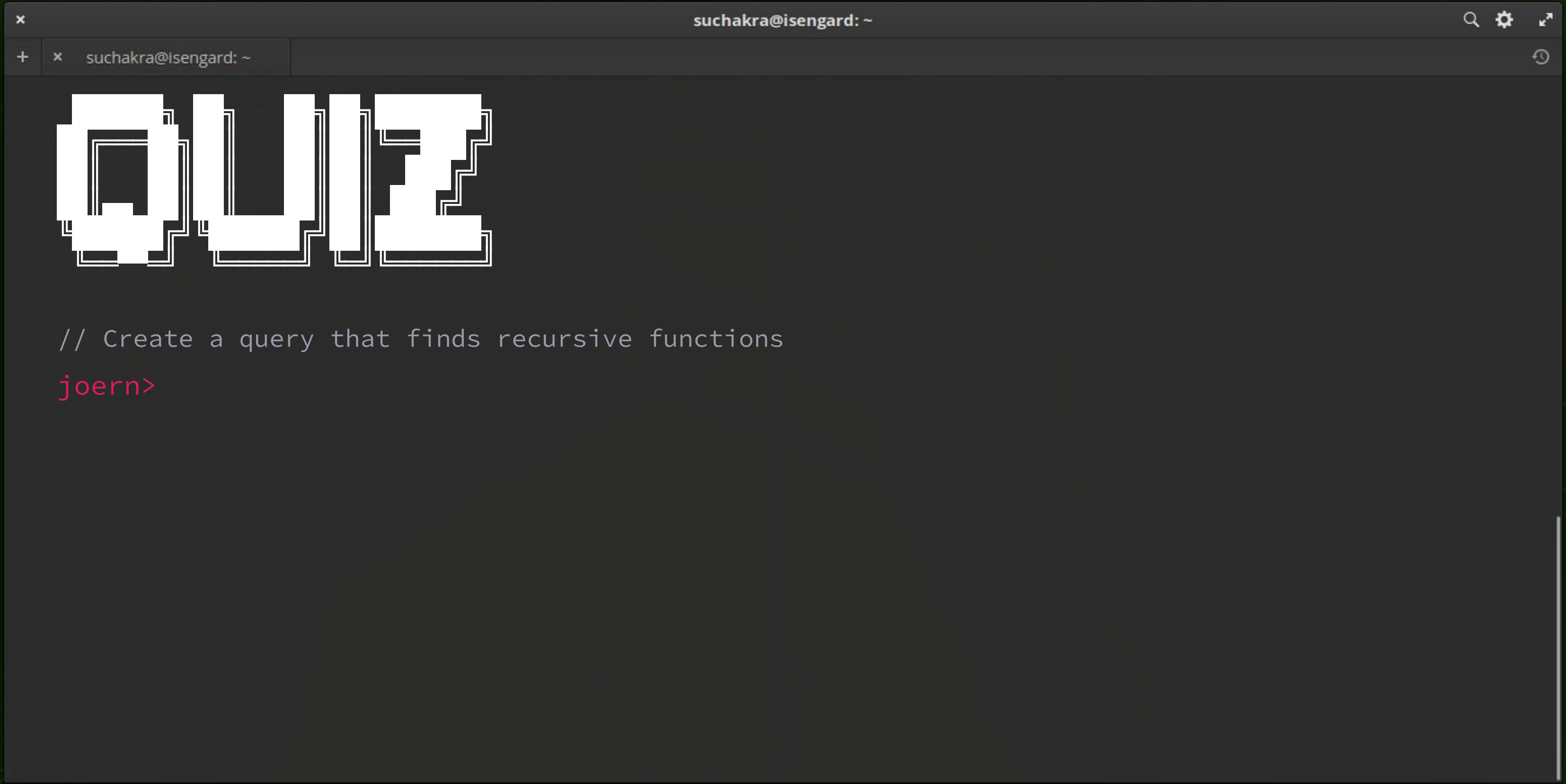
5. Basic Insights - Calls into Libraries



```
// All names of external methods used by the program
joern> cpg.method.external.name.l.distinct.sorted

// All calls to strcpy
joern> cpg.call("str.*").code.l

// All methods that call strcpy
joern> cpg.call("str.*").method.name.l

// Looking into parameters: second argument to sprintf is NOT a literal
joern> cpg.call("sprintf").argument(2).whereNot(_.isLiteral).code.l

// Quickly see this method above
joern> cpg.call("sprintf").argument(2).filterNot(_.isLiteral).dump
```

Module 1

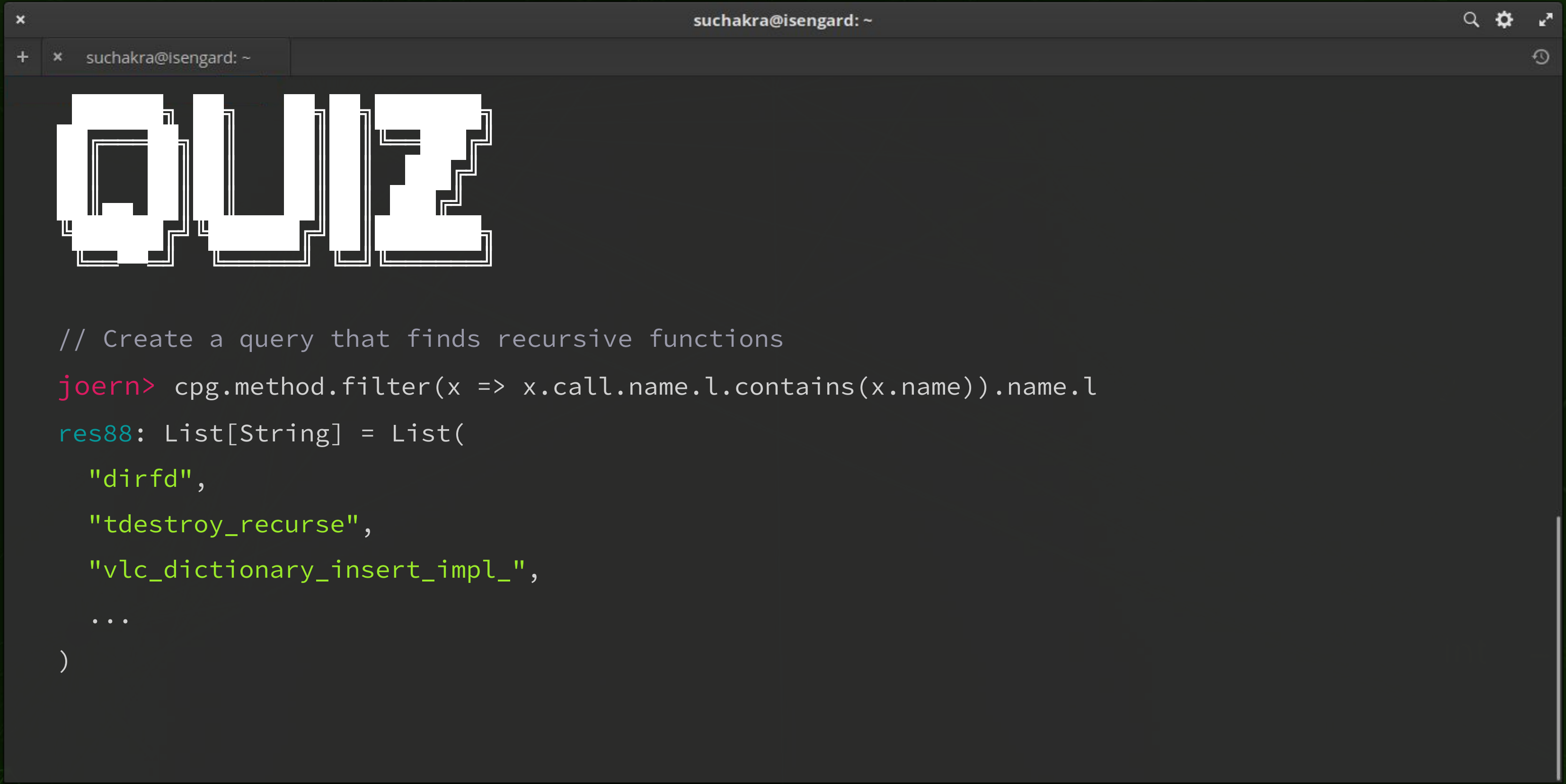
QUIZ



```
// Create a query that finds recursive functions
joern>
```

Module 1

QUIZ



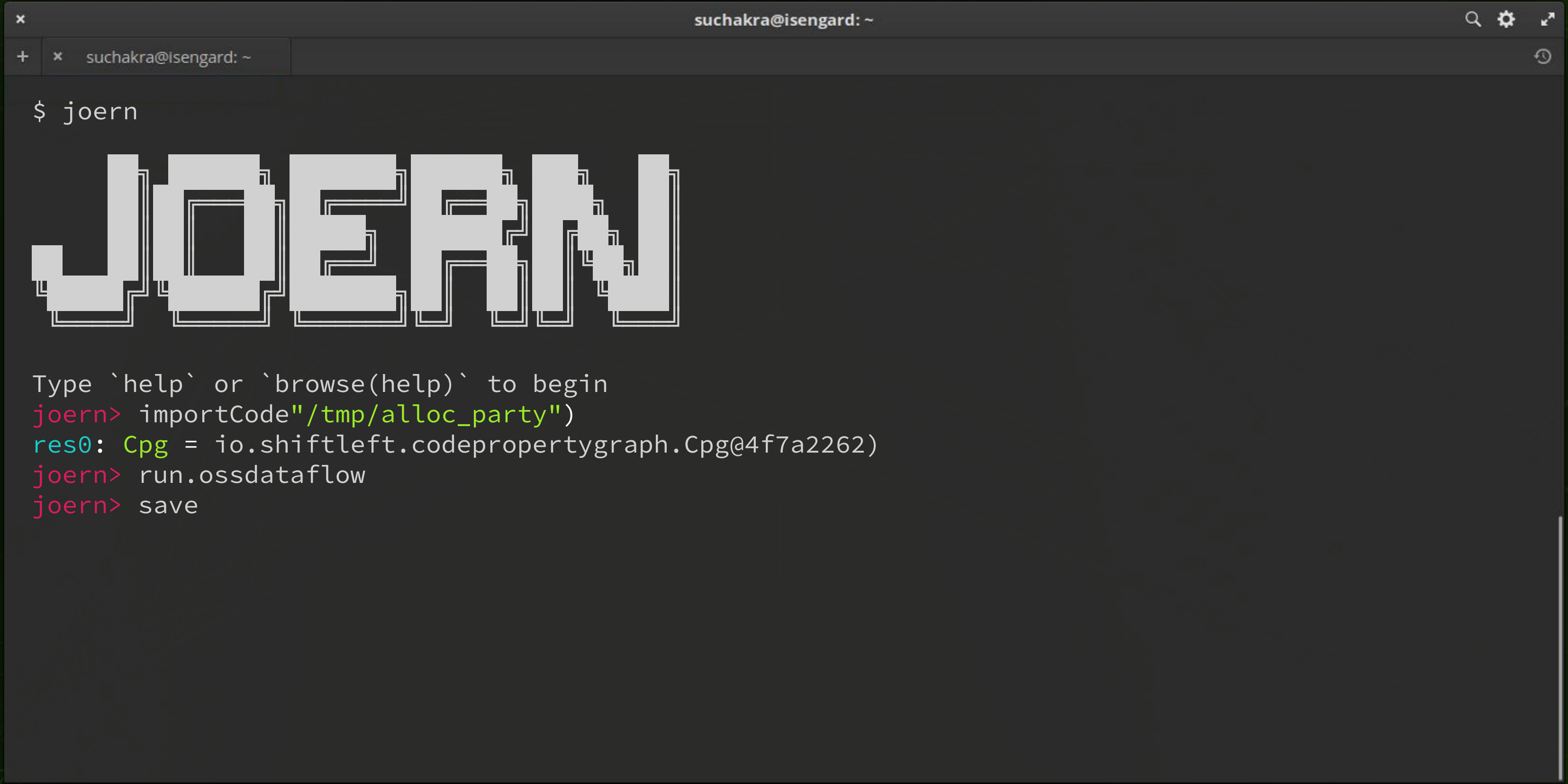
```
// Create a query that finds recursive functions
joern> cpg.method.filter(x => x.call.name.l.contains(x.name)).name.l
res88: List[String] = List(
  "dirfd",
  "tdestroy_recurse",
  "vlc_dictionary_insert_impl_",
  ...
)
```

Module 2

Hunting Memory Bugs

Module 2

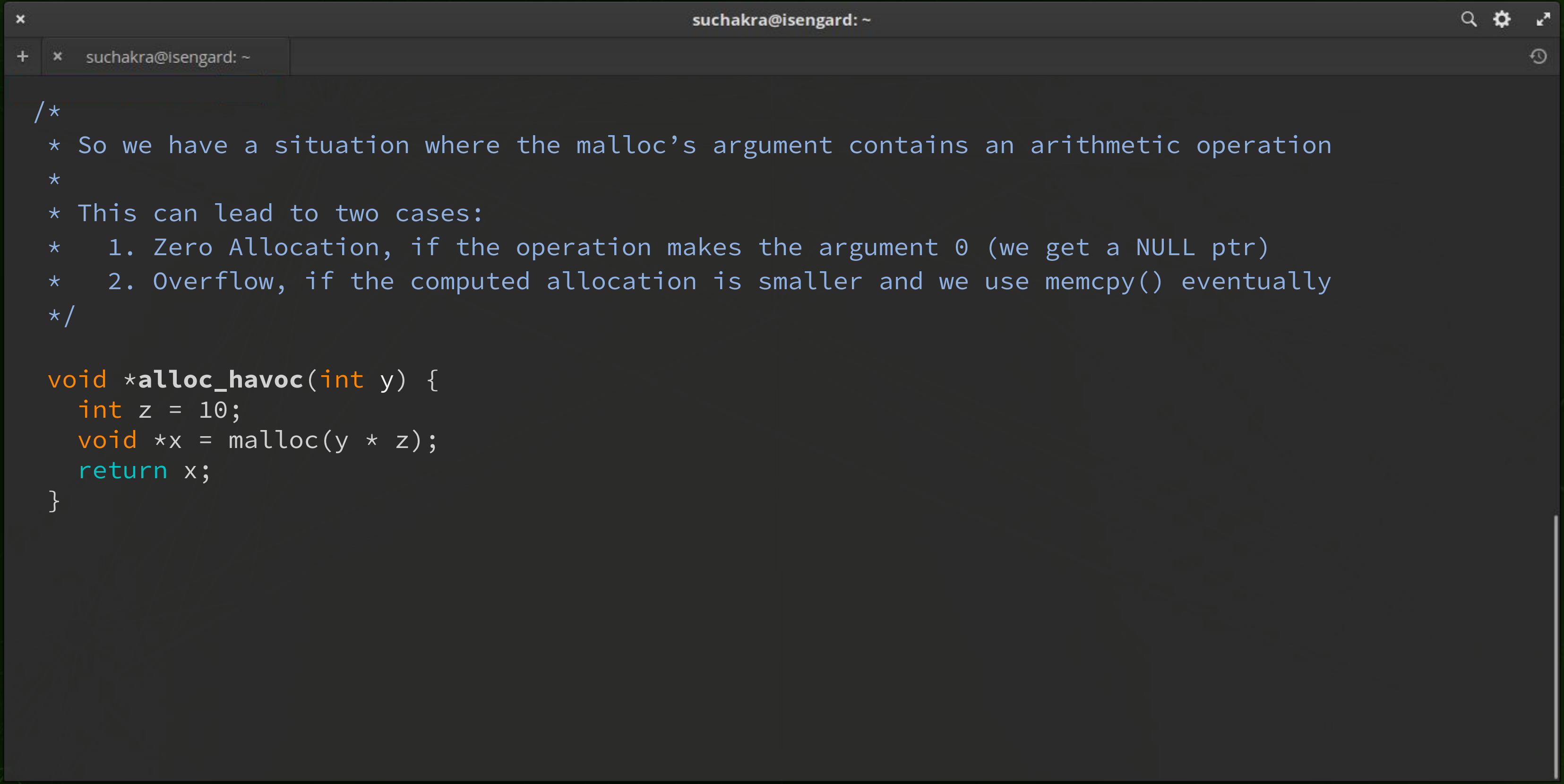
1. Generating CPG for alloc_party.c



```
$ joern

Type `help` or `browse(help)` to begin
joern> importCode"/tmp/alloc_party")
res0: Cpg = io.shiftleft.codepropertygraph.Cpg@4f7a2262)
joern> run.ossdataflow
joern> save
```

Module 2

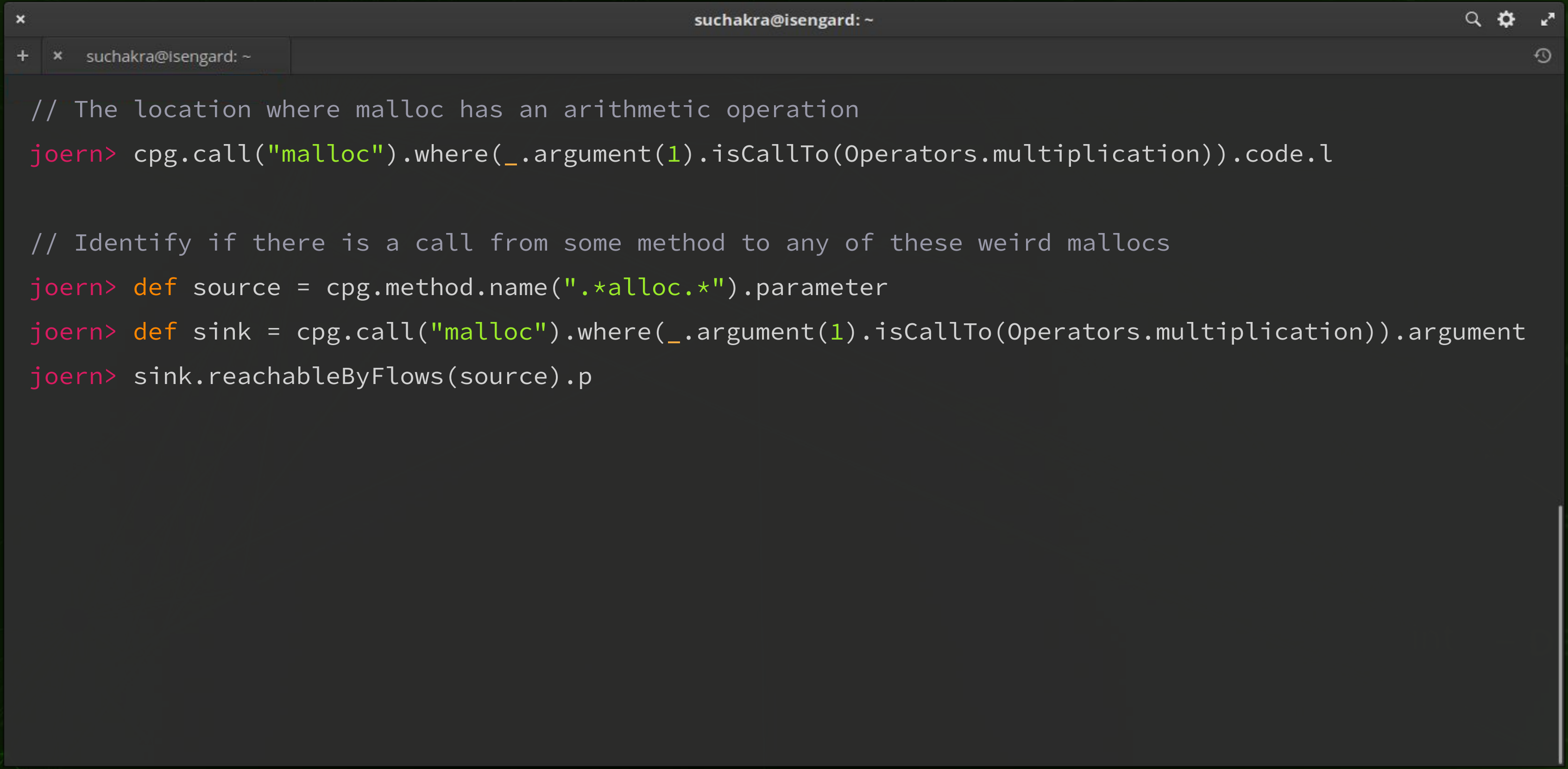
2. Memory Allocation Bugs - Zero Alloc/Overflow



```
/*
 * So we have a situation where the malloc’s argument contains an arithmetic operation
 *
 * This can lead to two cases:
 *   1. Zero Allocation, if the operation makes the argument 0 (we get a NULL ptr)
 *   2. Overflow, if the computed allocation is smaller and we use memcpy() eventually
 */

 void *alloc_havoc(int y) {
   int z = 10;
   void *x = malloc(y * z);
   return x;
 }
```

Module 2

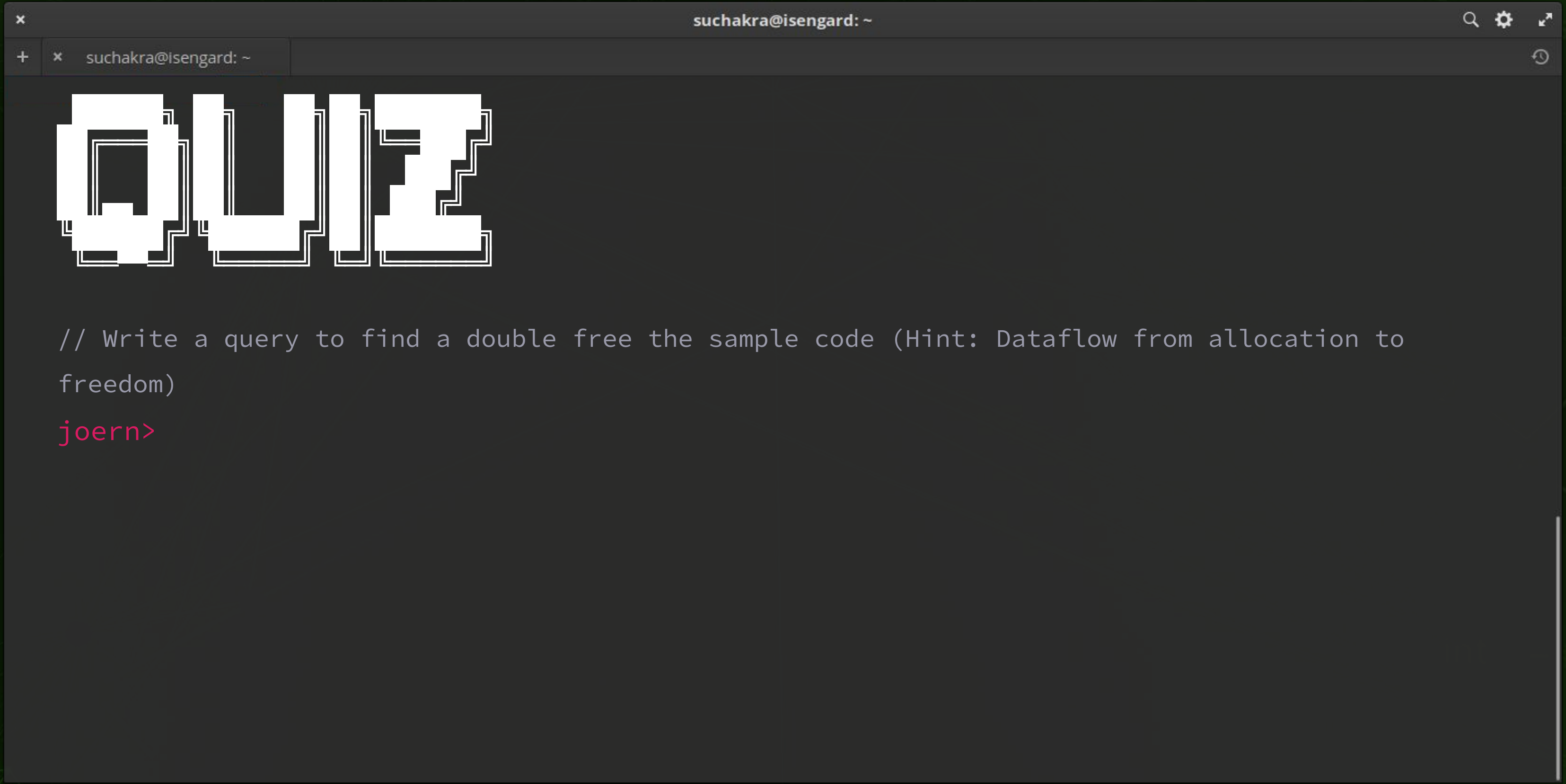
2. Memory Allocation Bugs - Zero Alloc/Overflow



```
// The location where malloc has an arithmetic operation
joern> cpg.call("malloc").where(_.argument(1).isCallTo(Operators.multiplication)).code.l

// Identify if there is a call from some method to any of these weird mallocs
joern> def source = cpg.method.name(".*alloc.*").parameter
joern> def sink = cpg.call("malloc").where(_.argument(1).isCallTo(Operators.multiplication)).argument
joern> sink.reachableByFlows(source).p
```

Module 2

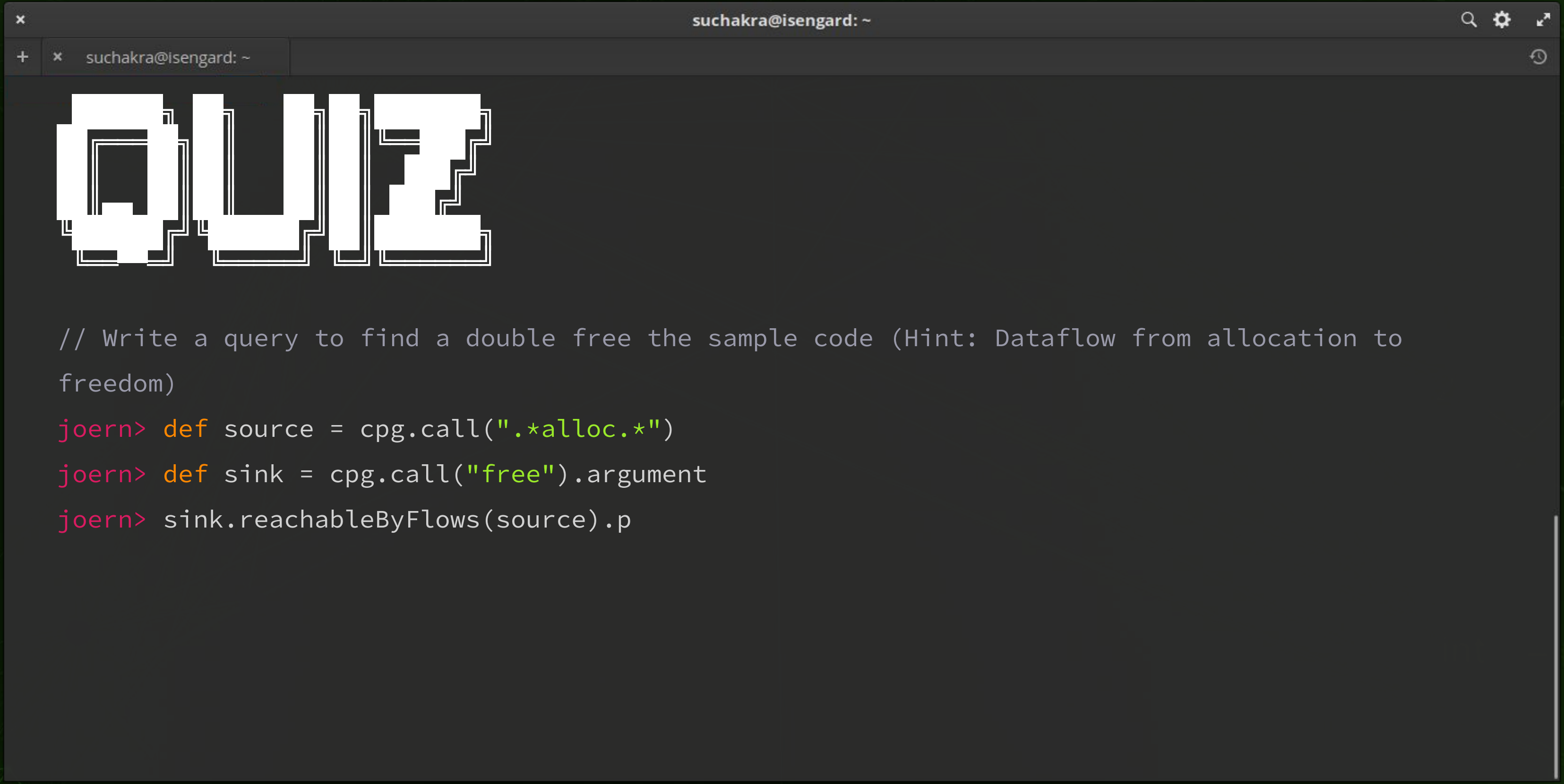
QUIZ



```
// Write a query to find a double free the sample code (Hint: Dataflow from allocation to
freedom)
joern>
```

Module 2

QUIZ



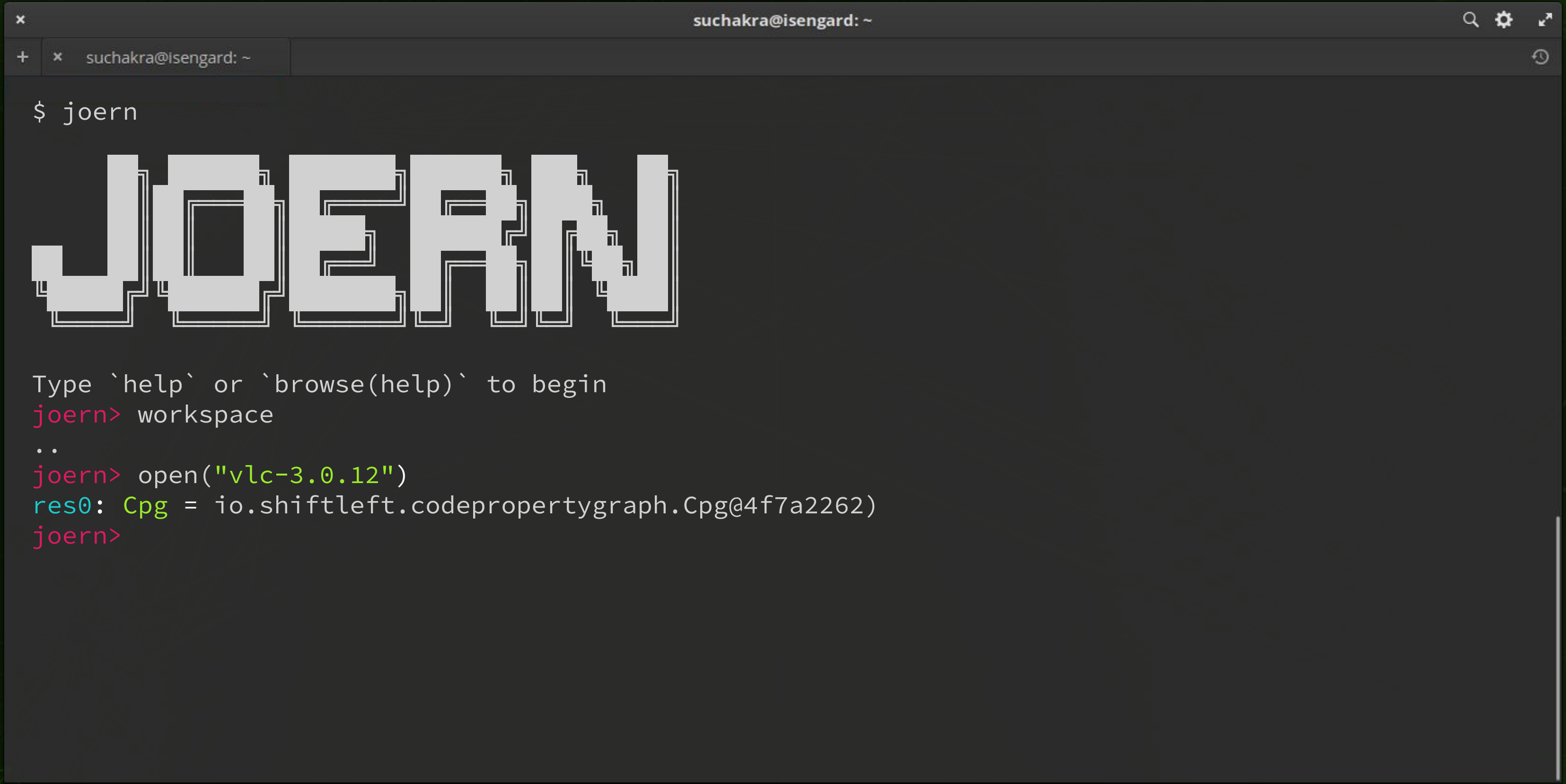
```
// Write a query to find a double free the sample code (Hint: Dataflow from allocation to
freedom)
joern> def source = cpg.call(".*alloc.*")
joern> def sink = cpg.call("free").argument
joern> sink.reachableByFlows(source).p
```

Module 3

Finding Vulnerabilities in VLC

Module 3

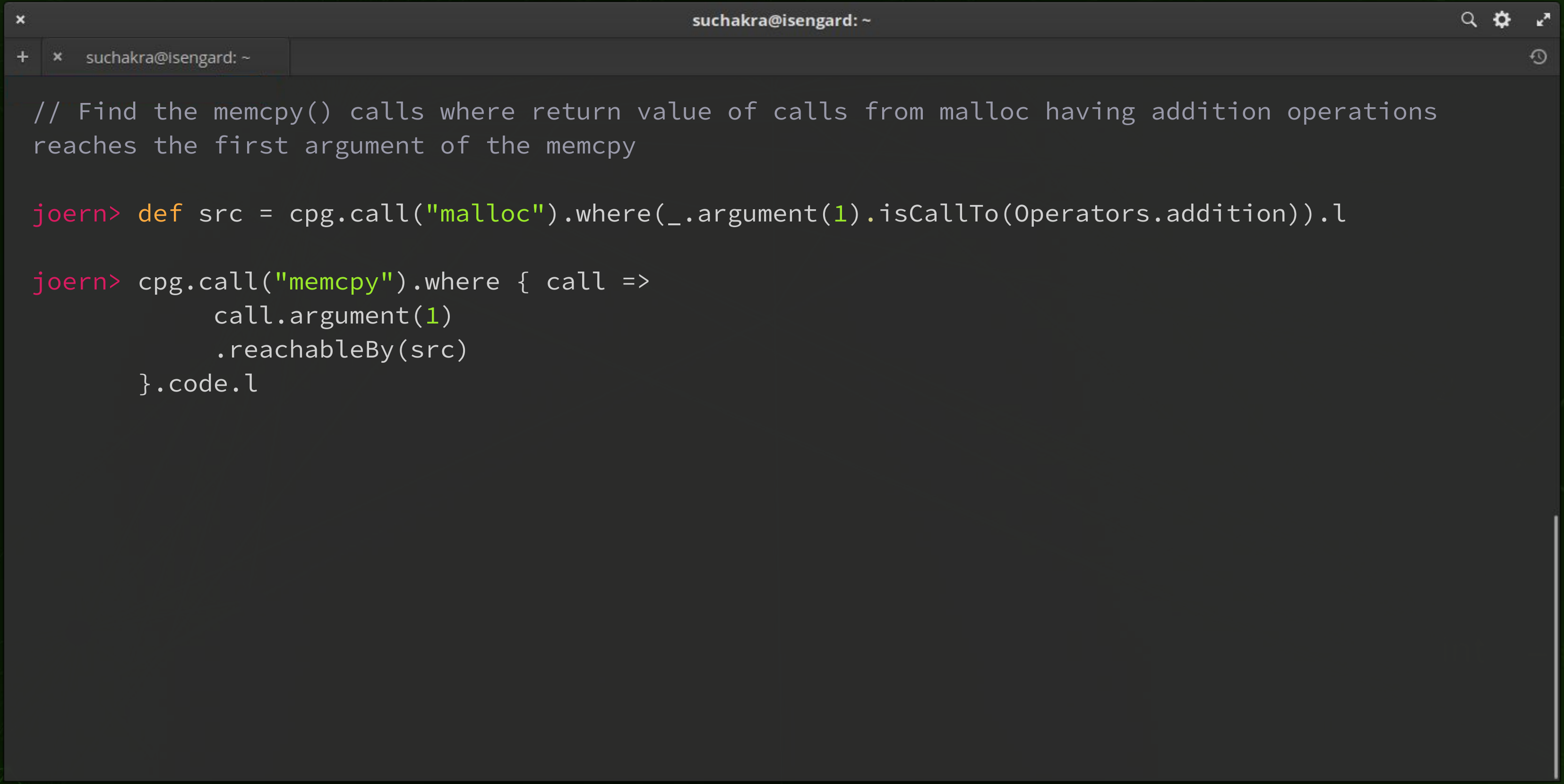
3. Back to the VLC CPG



```
$ joern

Type `help` or `browse(help)` to begin
joern> workspace
..
joern> open("vlc-3.0.12")
res0: Cpg = io.shiftleft.codepropertygraph.Cpg@4f7a2262)
joern>
```

Module 3

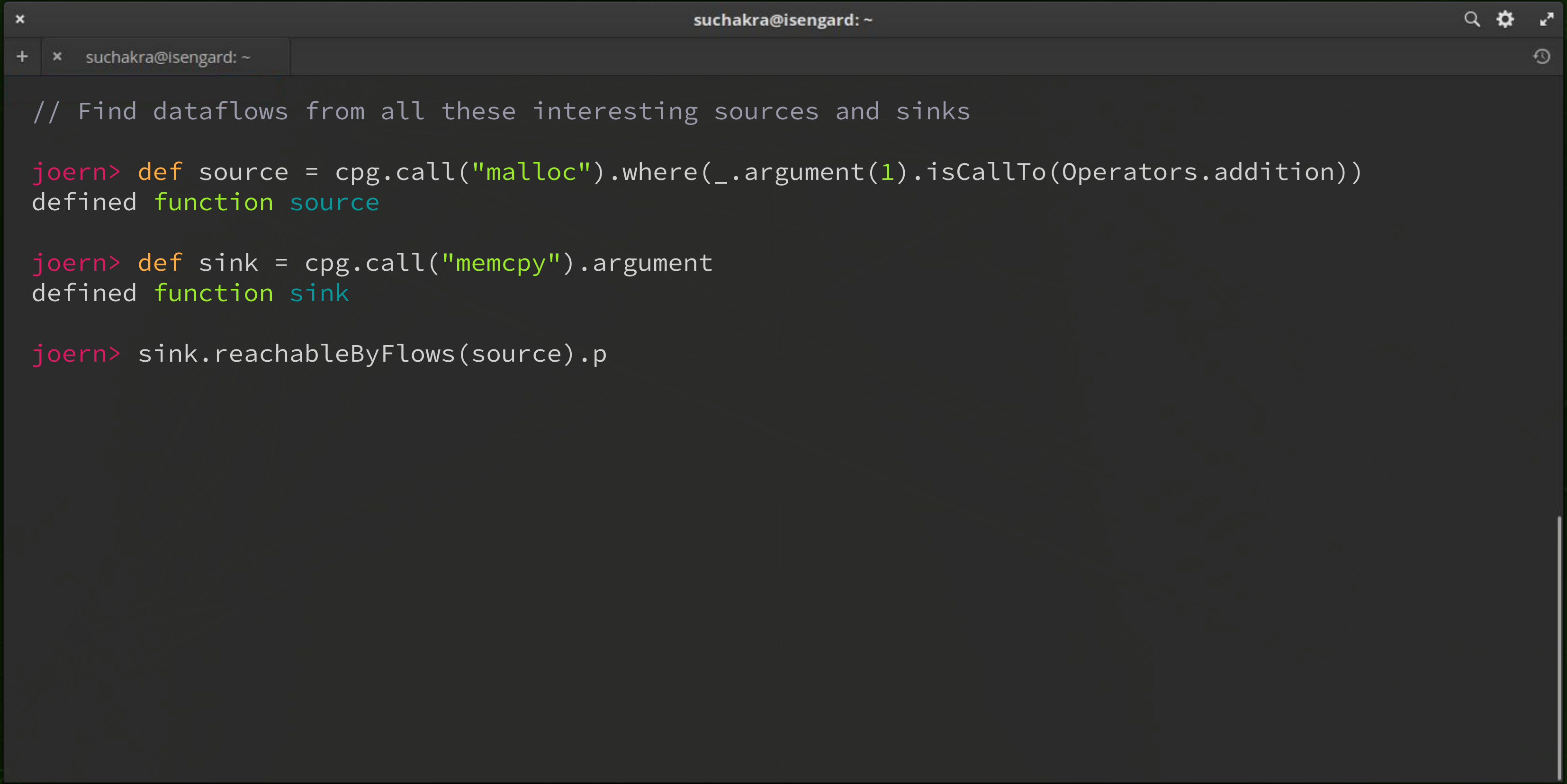
3. Buffer Overflow Hunting in VLC - First Try



```
// Find the memcpy() calls where return value of calls from malloc having addition operations
reaches the first argument of the memcpy

joern> def src = cpg.call("malloc").where(_.argument(1).isCallTo(Operators.addition)).l

joern> cpg.call("memcpy").where { call =>
            call.argument(1)
            .reachableBy(src)
       }.code.l
```

Module 3

3. Buffer Overflow Hunting in VLC - Dataflow



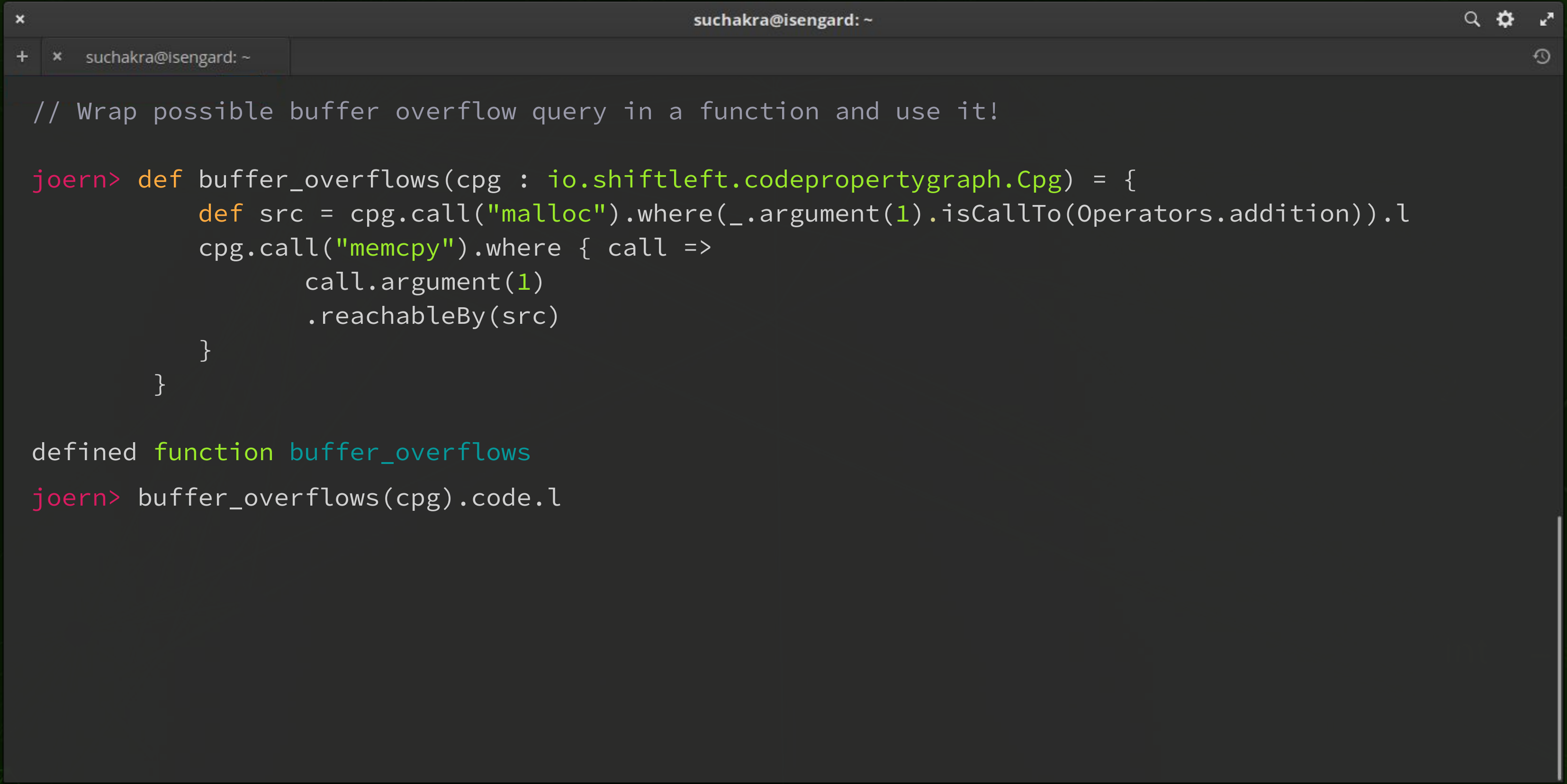
```
// Find dataflows from all these interesting sources and sinks

joern> def source = cpg.call("malloc").where(_.argument(1).isCallTo(Operators.addition))
defined function source

joern> def sink = cpg.call("memcpy").argument
defined function sink

joern> sink.reachableByFlows(source).p
```

Module 4

Joern Scripting

Module 4

1. Scripting - DRY Function

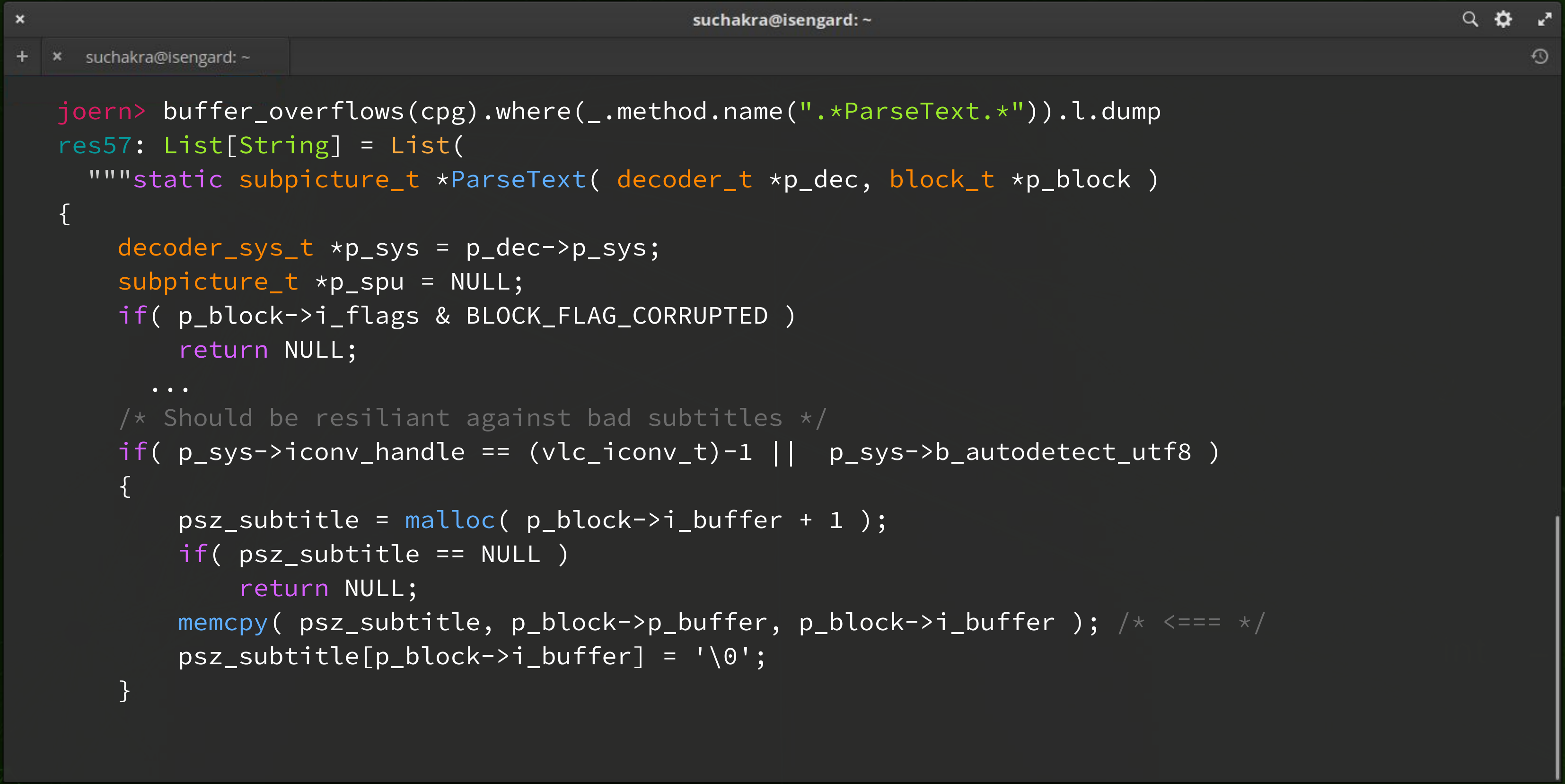


```
// Wrap possible buffer overflow query in a function and use it!

joern> def buffer_overflows(cpg : io.shiftleft.codepropertygraph.Cpg) = {
           def src = cpg.call("malloc").where(_.argument(1).isCallTo(Operators.addition)).l
           cpg.call("memcpy").where { call =>
                  call.argument(1)
                  .reachableBy(src)
           }
        }

defined function buffer_overflows

joern> buffer_overflows(cpg).code.l
```

Let’s test on VLC..

Module 4

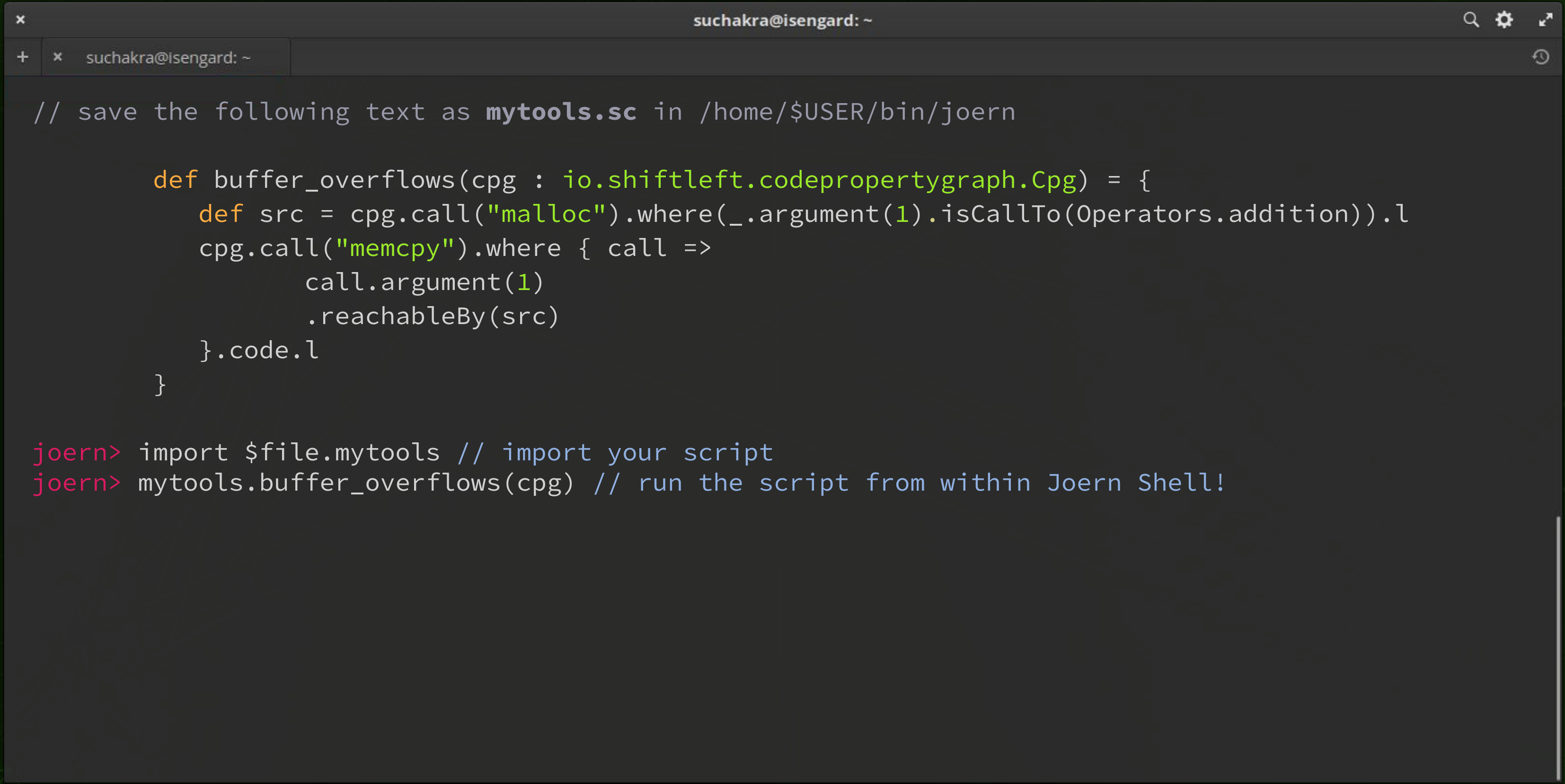
p_block->i_buffer == MAX_UINT64 **causes an overflow!**



```
joern> buffer_overflows(cpg).where(_.method.name(".*ParseText.*")).l.dump
res57: List[String] = List(
  """static subpicture_t *ParseText( decoder_t *p_dec, block_t *p_block )
{
    decoder_sys_t *p_sys = p_dec->p_sys;
    subpicture_t *p_spu = NULL;
    if( p_block->i_flags & BLOCK_FLAG_CORRUPTED )
        return NULL;
      ...
    /* Should be resiliant against bad subtitles */
    if( p_sys->iconv_handle == (vlc_iconv_t)-1 ||  p_sys->b_autodetect_utf8 )
    {
        psz_subtitle = malloc( p_block->i_buffer + 1 );
        if( psz_subtitle == NULL )
            return NULL;
        memcpy( psz_subtitle, p_block->p_buffer, p_block->i_buffer ); /* <=== */
        psz_subtitle[p_block->i_buffer] = '\0';
    }
```

Module 4

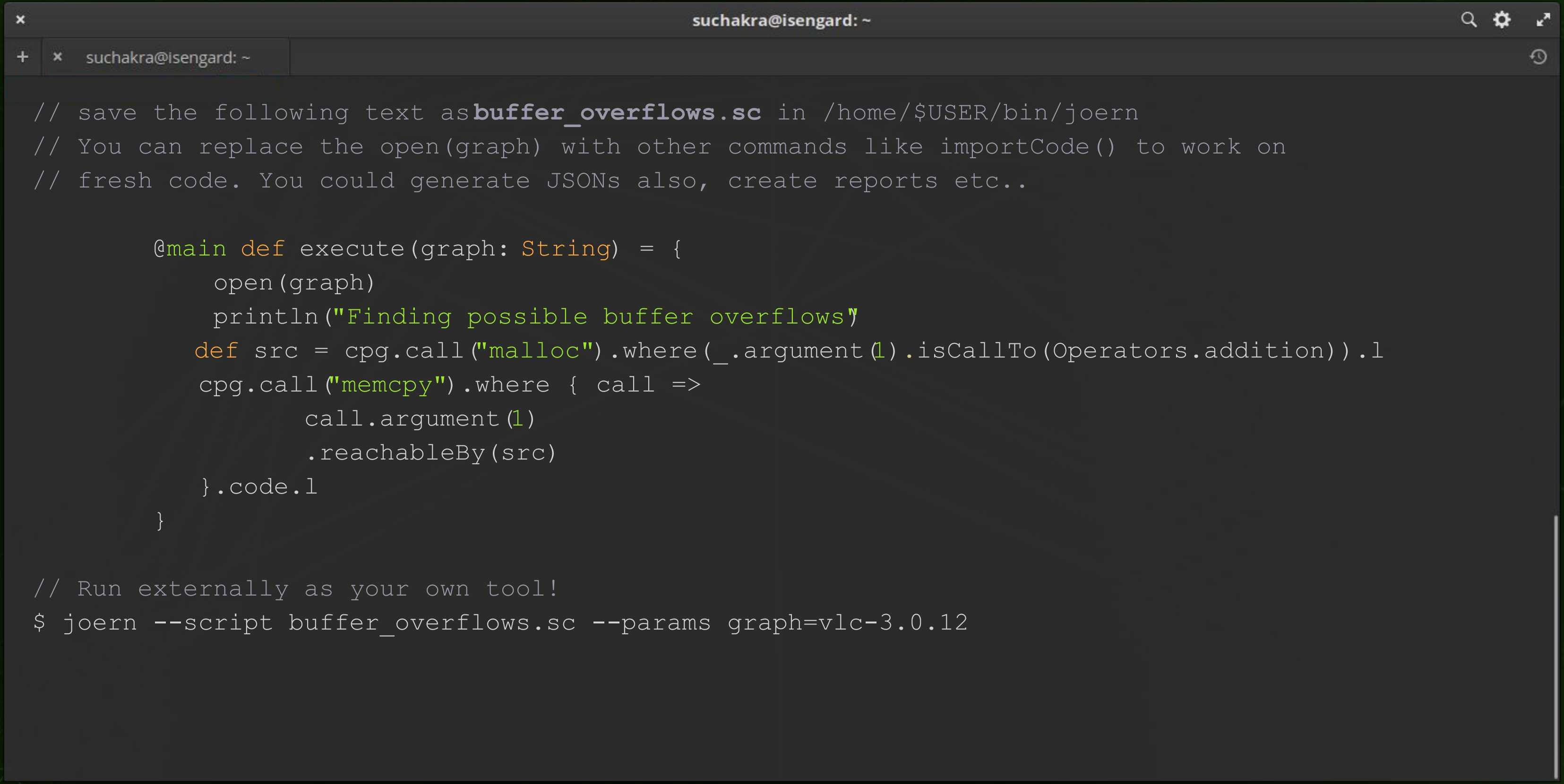
1. Scripting - Creating Internal Tools



```
// save the following text as mytools.sc in /home/$USER/bin/joern

        def buffer_overflows(cpg : io.shiftleft.codepropertygraph.Cpg) = {
           def src = cpg.call("malloc").where(_.argument(1).isCallTo(Operators.addition)).l
           cpg.call("memcpy").where { call =>
                  call.argument(1)
                  .reachableBy(src)
           }.code.l
        }

joern> import $file.mytools // import your script
joern> mytools.buffer_overflows(cpg) // run the script from within Joern Shell!
```

Module 4

1. Scripting - Creating External Standalone Tools



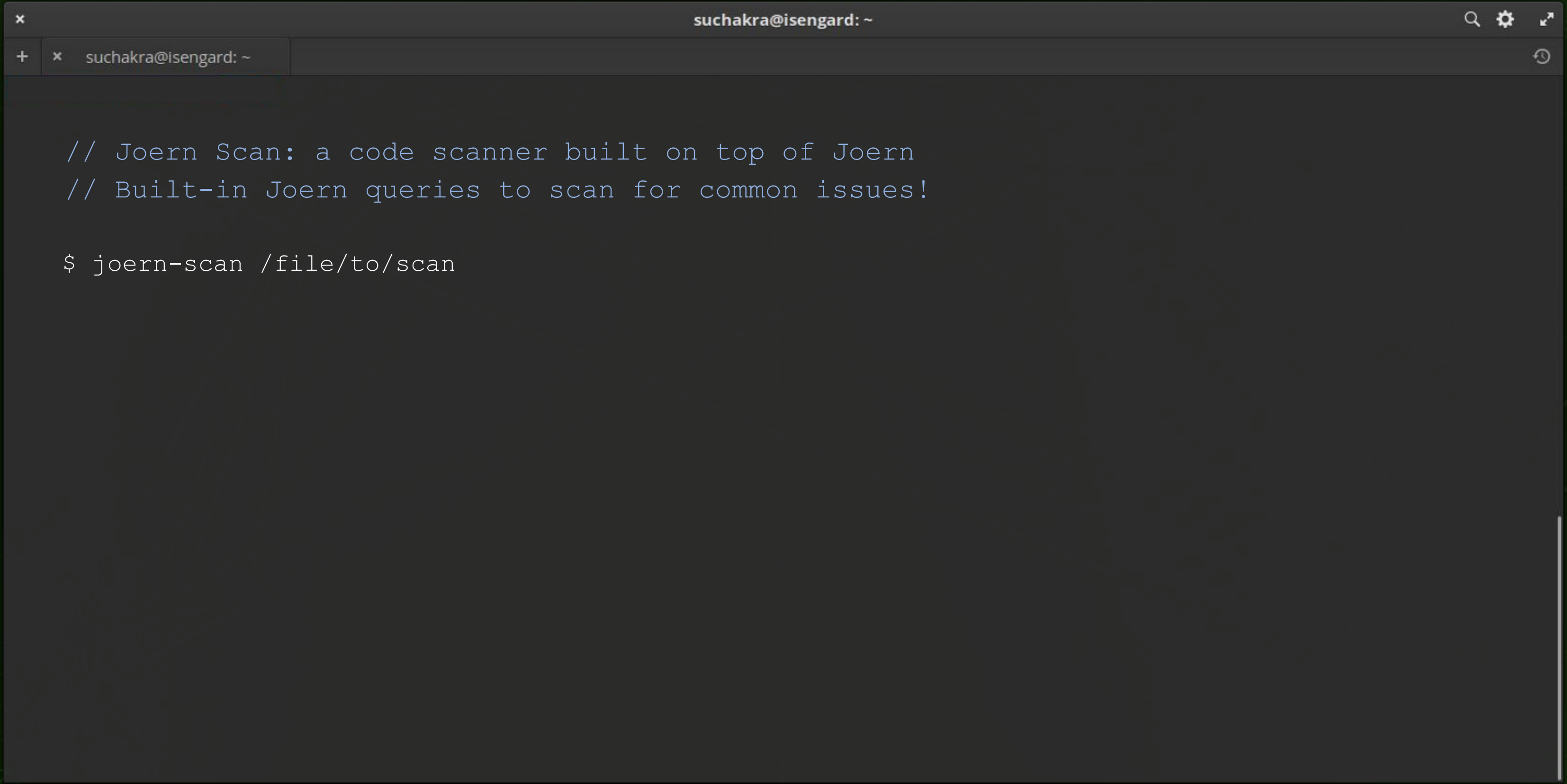
```
// save the following text asbuffer_overflows.sc in /home/$USER/bin/joern
// You can replace the open(graph) with other commands like importCode() to work on
// fresh code. You could generate JSONs also, create reports etc..

        @main def execute(graph: String) = {
            open(graph)
            println("Finding possible buffer overflows")
           def src = cpg.call("malloc").where(_.argument(1).isCallTo(Operators.addition)).l
           cpg.call("memcpy").where { call =>
                  call.argument(1)
                  .reachableBy(src)
           }.code.l
        }

// Run externally as your own tool!
$ joern --script buffer_overflows.sc --params graph=vlc-3.0.12
```

Module 5

Building Custom Scanners

Module 5

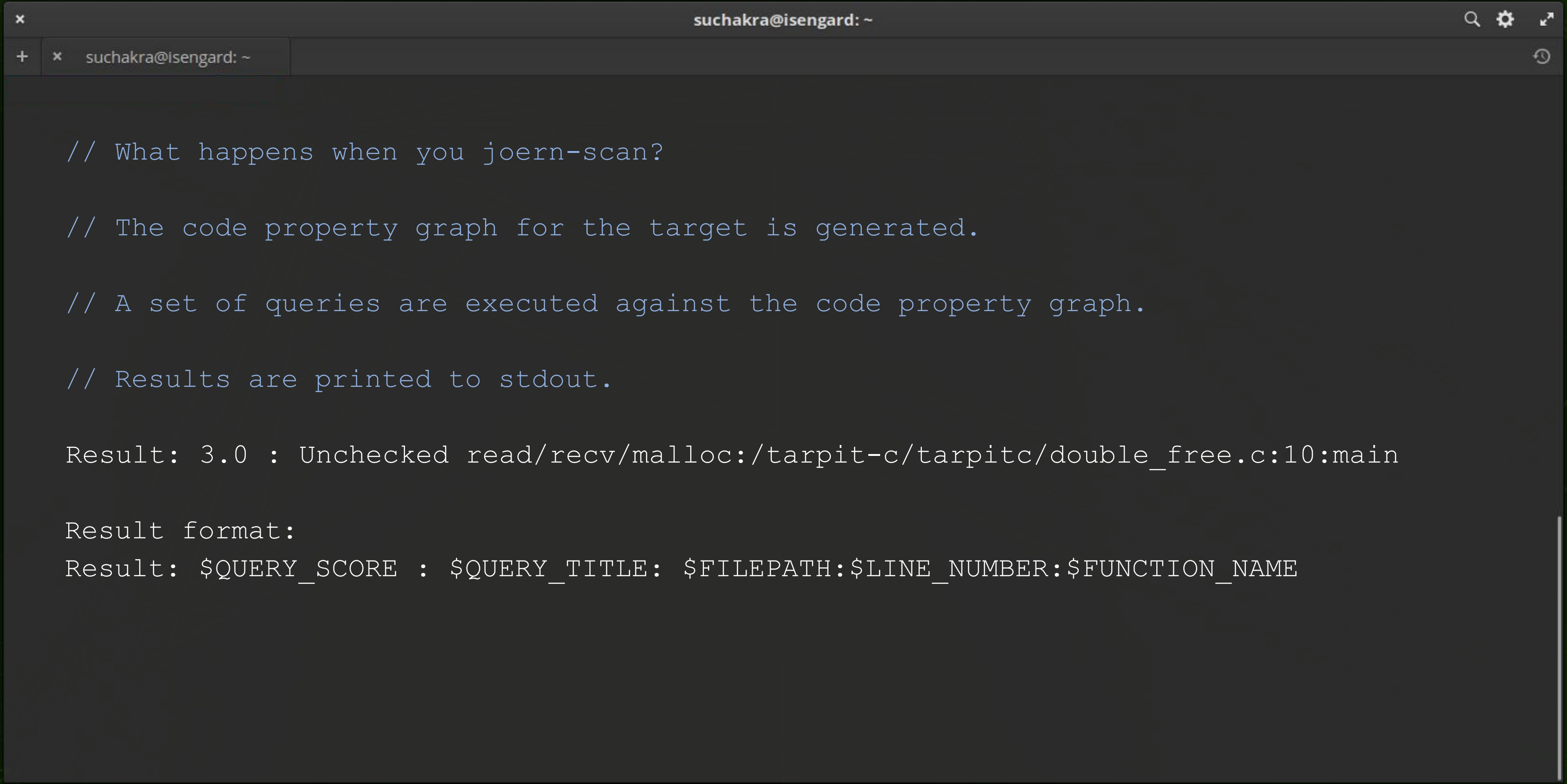
1. Custom Scanning - Joern Scan



```
// Joern Scan: a code scanner built on top of Joern
// Built-in Joern queries to scan for common issues!

$ joern-scan /file/to/scan
```

Module 5

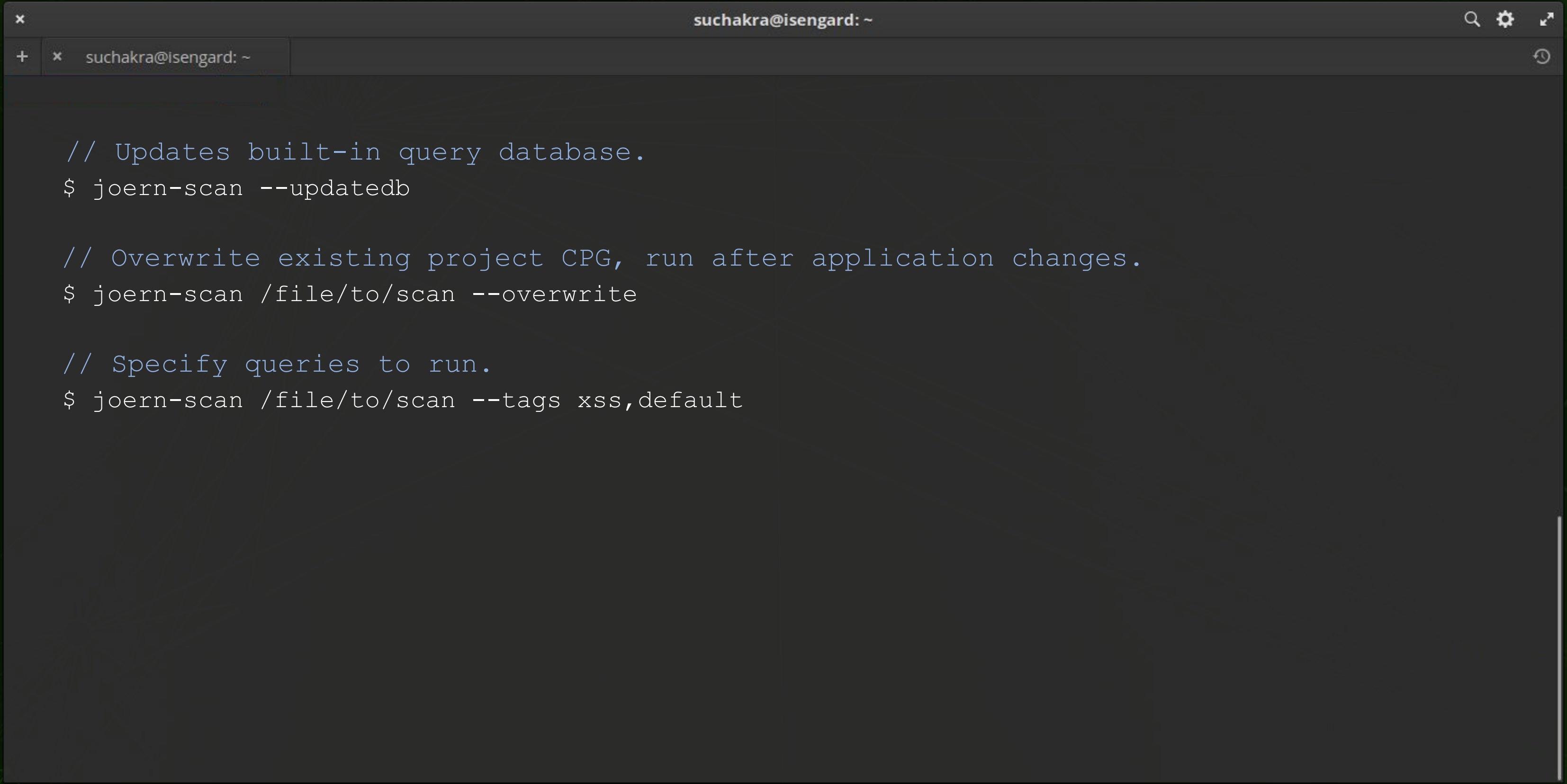
2. Custom Scanning - Under The Hood



```
// What happens when you joern-scan?

// The code property graph for the target is generated.

// A set of queries are executed against the code property graph.

// Results are printed to stdout.

Result: 3.0 : Unchecked read/recv/malloc:/tarpit-c/tarpitc/double_free.c:10:main

Result format:
Result: $QUERY_SCORE : $QUERY_TITLE: $FILEPATH:$LINE_NUMBER:$FUNCTION_NAME
```

Module 5

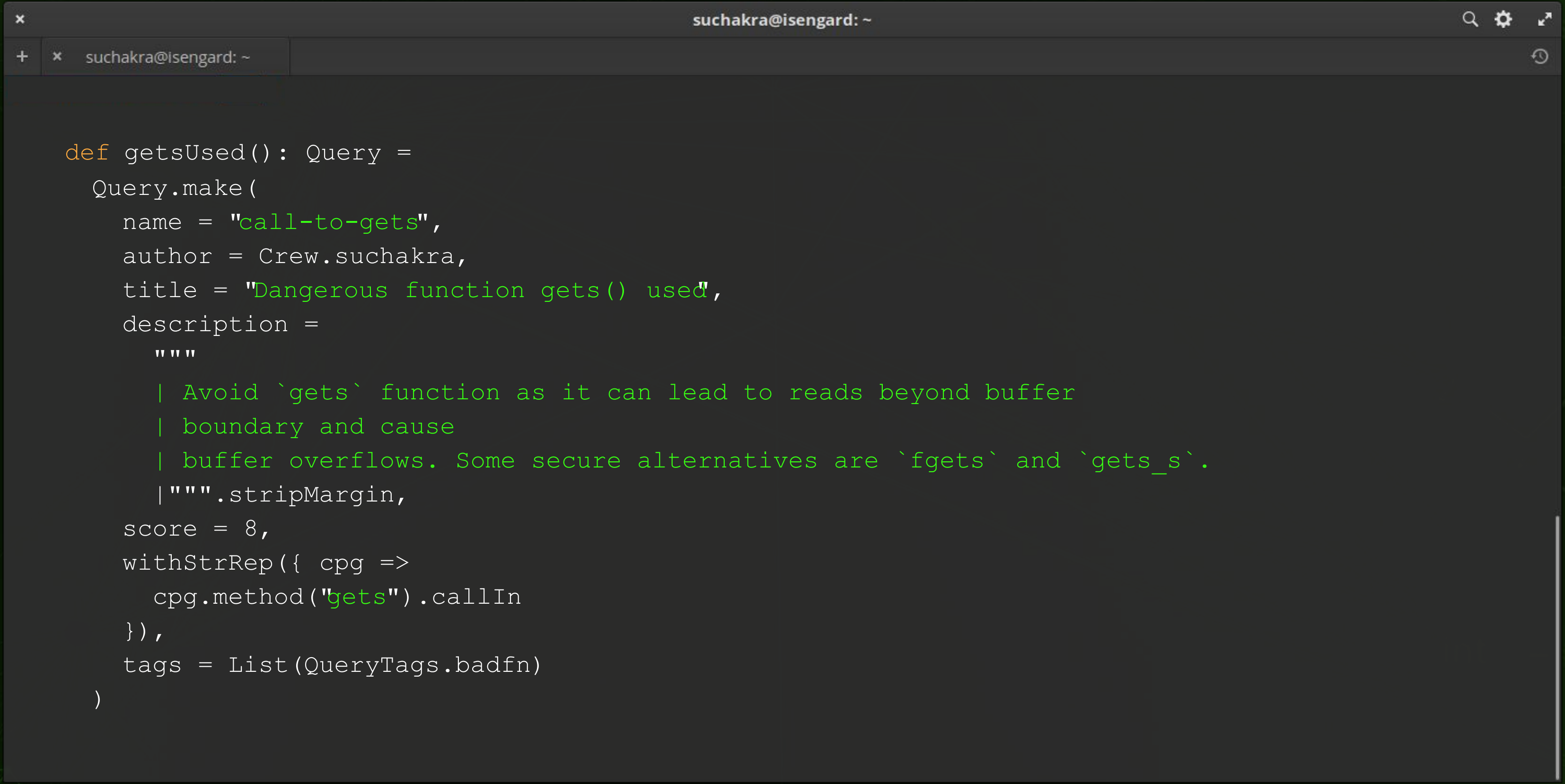
3. Custom Scanning - Joern Scan Options



```
// Updates built-in query database.
$ joern-scan --updatedb

// Overwrite existing project CPG, run after application changes.
$ joern-scan /file/to/scan --overwrite

// Specify queries to run.
$ joern-scan /file/to/scan --tags xss,default
```

Module 5

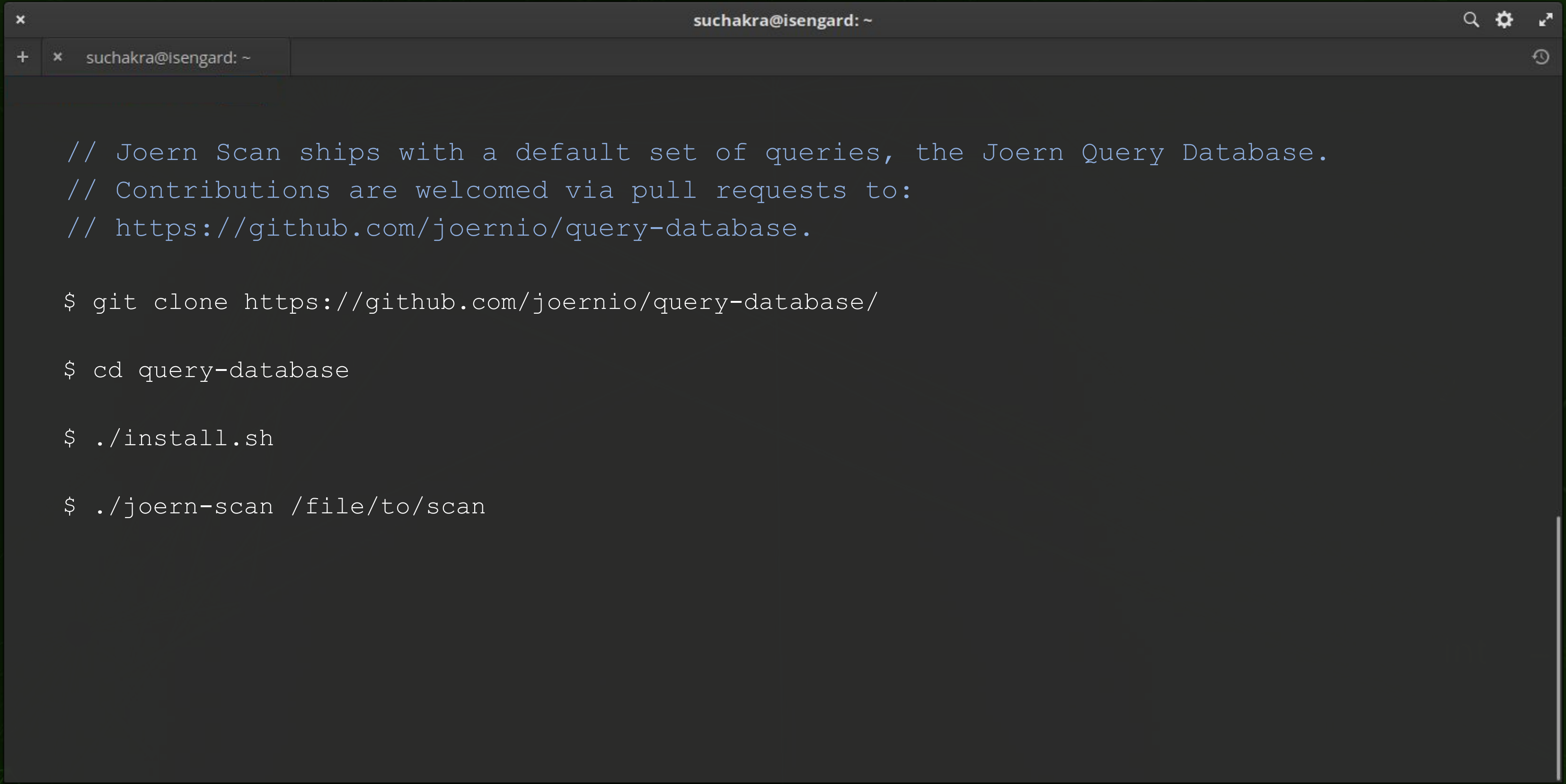
4. Custom Scanning - Joern Scan Queries



```
def getsUsed(): Query =
  Query.make(
    name = "call-to-gets",
    author = Crew.suchakra,
    title = "Dangerous function gets() used",
    description =
      """
      | Avoid `gets` function as it can lead to reads beyond buffer
      | boundary and cause
      | buffer overflows. Some secure alternatives are `fgets` and `gets_s`.
      |""".stripMargin,
    score = 8,
    withStrRep({ cpg =>
      cpg.method("gets").callIn
    }),
    tags = List(QueryTags.badfn)
  )
```

Module 5

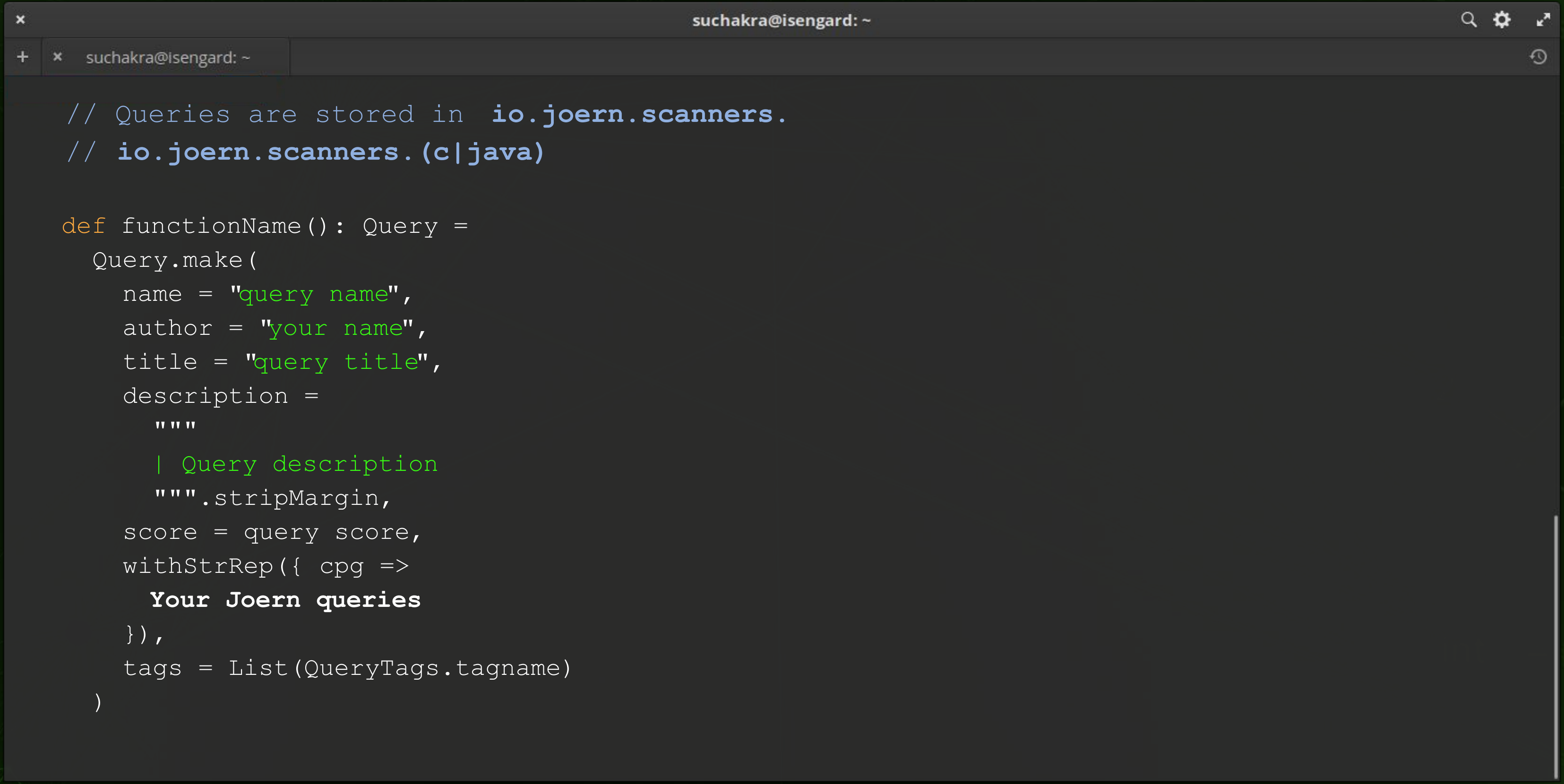
5. Custom Scanning - Extending Joern Scan



```
// Joern Scan ships with a default set of queries, the Joern Query Database.
// Contributions are welcomed via pull requests to:
// https://github.com/joernio/query-database.

$ git clone https://github.com/joernio/query-database/

$ cd query-database

$ ./install.sh

$ ./joern-scan /file/to/scan
```

Module 5

6. Custom Scanning - Adding Your Own Queries



```
// Queries are stored in  io.joern.scanners.
// io.joern.scanners.(c|java)

def functionName(): Query =
  Query.make(
    name = "query name",
    author = "your name",
    title = "query title",
    description =
      """
      | Query description
      """.stripMargin,
    score = query score,
    withStrRep({ cpg =>
      Your Joern queries
    }),
    tags = List(QueryTags.tagname)
  )
```

Open Forum

Q&A - Please use Discord

Fin

http://joern.io