Hunting Vulnerabilities with Graph Databases

Fabian ‘fabs’ Yamaguchi
Nico Golde (Qualcomm)
INBOT’14

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

CVE-2013-6381: *qeth buffer overflow in snmp ioctl*

```
int qeth_snmp_command(struct qeth_card *card, char __user *udata)
{
        struct qeth_cmd_buffer *iob;
        struct qeth_ipa_cmd *cmd;
        struct qeth_snmp_ureq *ureq;
        int req_len;
        struct qeth_arp_query_info qinfo = {0, };
        int rc = 0;

        // [...]

        /* skip 4 bytes (data_len struct member) to get req_len */
        if (copy_from_user(&req_len, udata + sizeof(int), sizeof(int)))
                return -EFAULT;
        ureq = memdup_user(udata, req_len + sizeof(struct qeth_snmp_ureq_hdr));
        if (IS_ERR(ureq)) {
                QETH_CARD_TEXT(card, 2, "snmpnome");
                return PTR_ERR(ureq);
        }
        qinfo.udata_len = ureq->hdr.data_len;
        qinfo.udata = kzalloc(qinfo.udata_len, GFP_KERNEL);
        if (!qinfo.udata) {
                kfree(ureq);
                return -ENOMEM;
        }
        // [...]
        memcpy(&cmd->data.setadapterparms.data.snmp, &ureq->cmd, req_len);
        // [...]
        return rc;
```

CVE-2013-6381: *qeth buffer overflow in snmp ioctl*

```
int qeth_snmp_command(struct qeth_card *card, char __user *udata)
{
        struct qeth_cmd_buffer *iob;
        struct qeth_ipa_cmd *cmd;
        struct qeth_snmp_ureq *ureq;
        int req_len;
        struct qeth_arp_query_info qinfo = {0, };
        int rc = 0;

        // [...]

        /* skip 4 bytes (data_len struct member) to get req_len */
        if (copy_from_user(&req_len, udata + sizeof(int), sizeof(int)))
                return -EFAULT;
        ureq = memdup_user(udata, req_len + sizeof(struct qeth_snmp_ureq_hdr));
        if (IS_ERR(ureq)) {
                QETH_CARD_TEXT(card, 2, "snmpnome");
                return PTR_ERR(ureq);
        }
        qinfo.udata_len = ureq->hdr.data_len;
        qinfo.udata = kzalloc(qinfo.udata_len, GFP_KERNEL);
        if (!qinfo.udata) {
                kfree(ureq);
                return -ENOMEM;
        }
        // [...]
        memcpy(&cmd->data.setadapterparms.data.snmp, &ureq->cmd, req_len);
        // [...]
        return rc;
```

First arg of copy_from_user propagates to third arg of memcpy without being checked.

CVE-2013-6381: *qeth buffer overflow in snmp ioctl*

```
int qeth_snmp_command(struct qeth_card *card, char __user *udata)
{
        struct qeth_cmd_buffer *iob;
        struct qeth_ipa_cmd *cmd;
        struct qeth_snmp_ureq *ureq;
        int req_len;
        struct qeth_arp_query_info qinfo = {0, };
        int rc = 0;

        // [...]

        /* skip 4 bytes (data_len struct member) to get req_len */
        if (copy_from_user(&req_len, udata + sizeof(int), sizeof(int)))
                return -EFAULT;
                                                                    );
```

Goal: make searching for bugs as easy as talking about them.

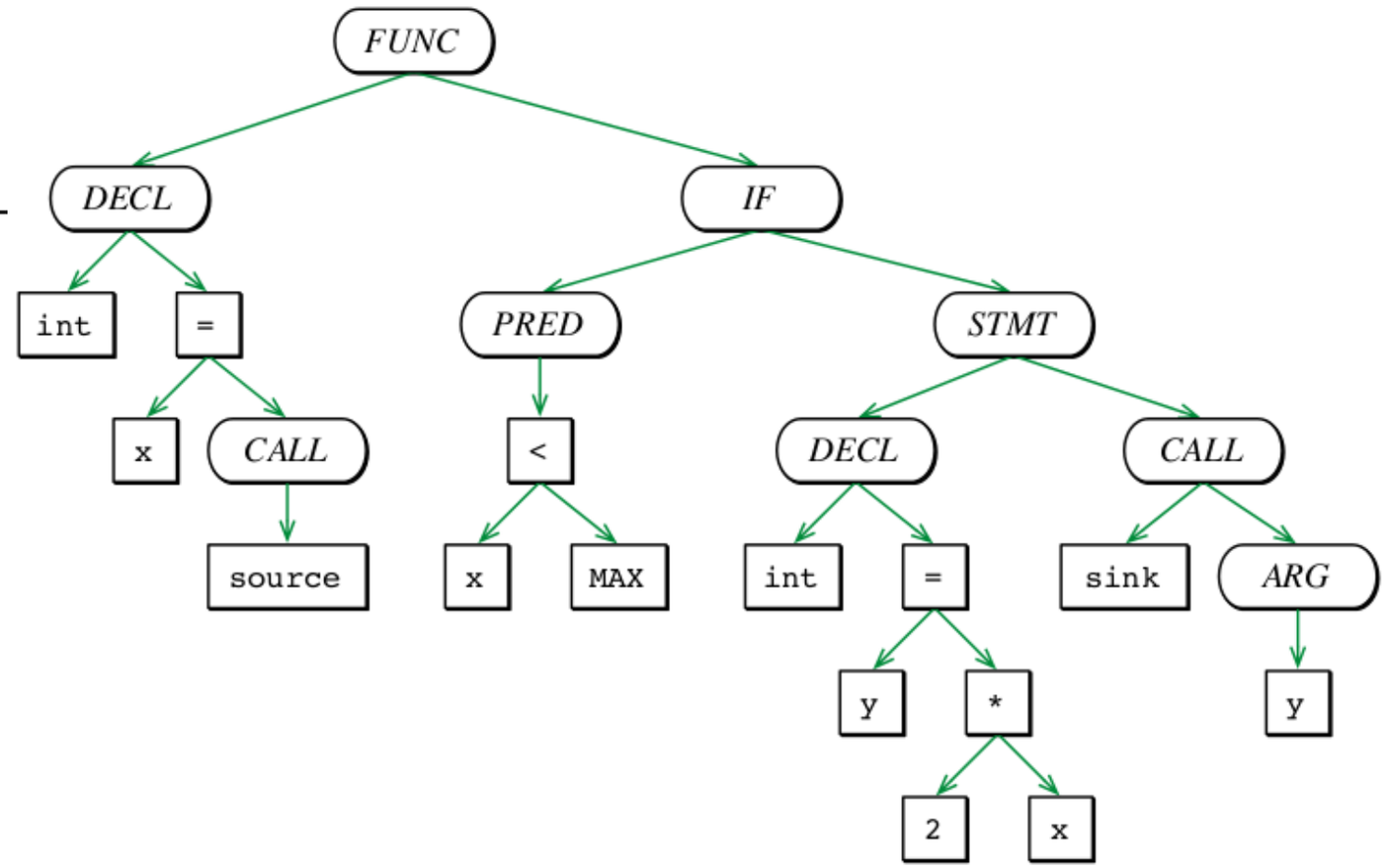
```
        }
        qinfo.udata_len = ureq->hdr.data_len;
        qinfo.udata = kzalloc(qinfo.udata_len, GFP_KERNEL);
        if (!qinfo.udata) {
                kfree(ureq);
                return -ENOMEM;
        }
        // [...]
        memcpy(&cmd->data.setadapterparms.data.snmp, &ureq->cmd, req_len);
        // [...]
        return rc;
```

Building a Bug Search Engine

- It’s only really useful it it’s generic enough to express lots of different kinds of bugs
- We want to be able to query the database for
  - What the code looks like (syntax)
  - Statement execution order (control flow)
  - How data flows through the program (data flow)
- Long story short: We built it, let’s talk about it.

How compilers analyze syntax: ASTs

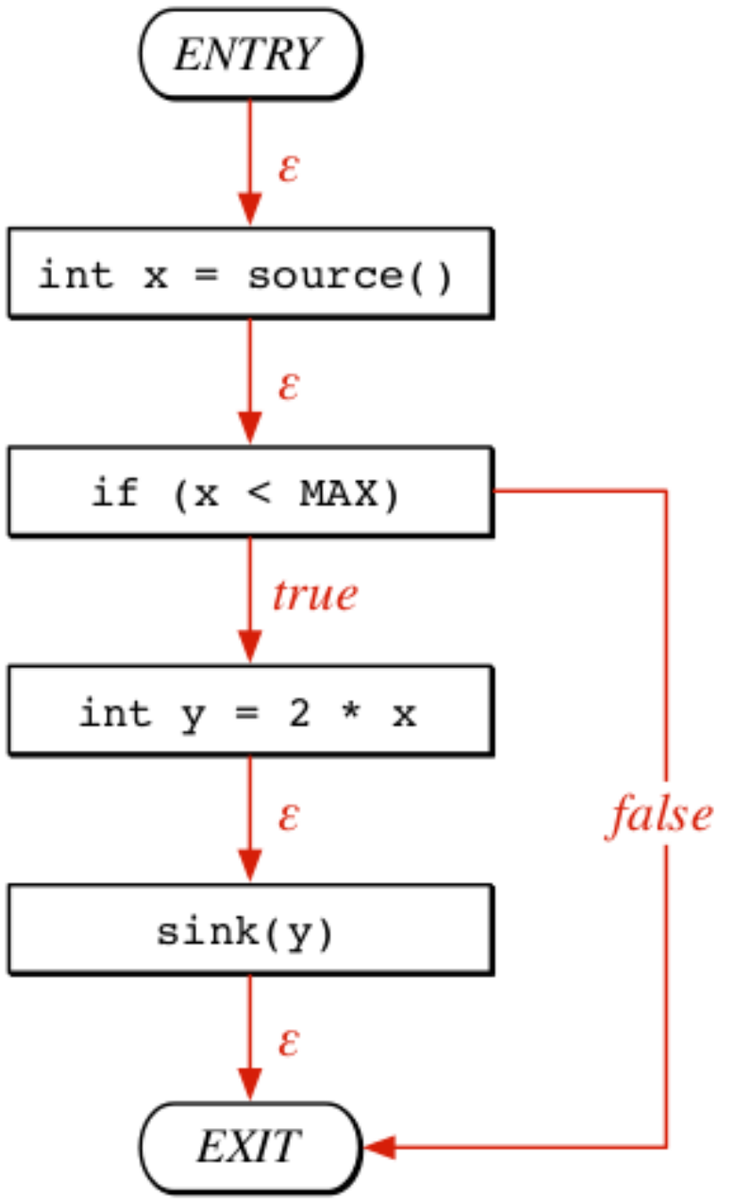
```
void foo()
{
    int x = source();
    if (x < MAX)
    {
        int y = 2 * x;
        sink(y);
    }
}
```



(a) Abstract syntax tree (AST)

How compilers analyze control flow

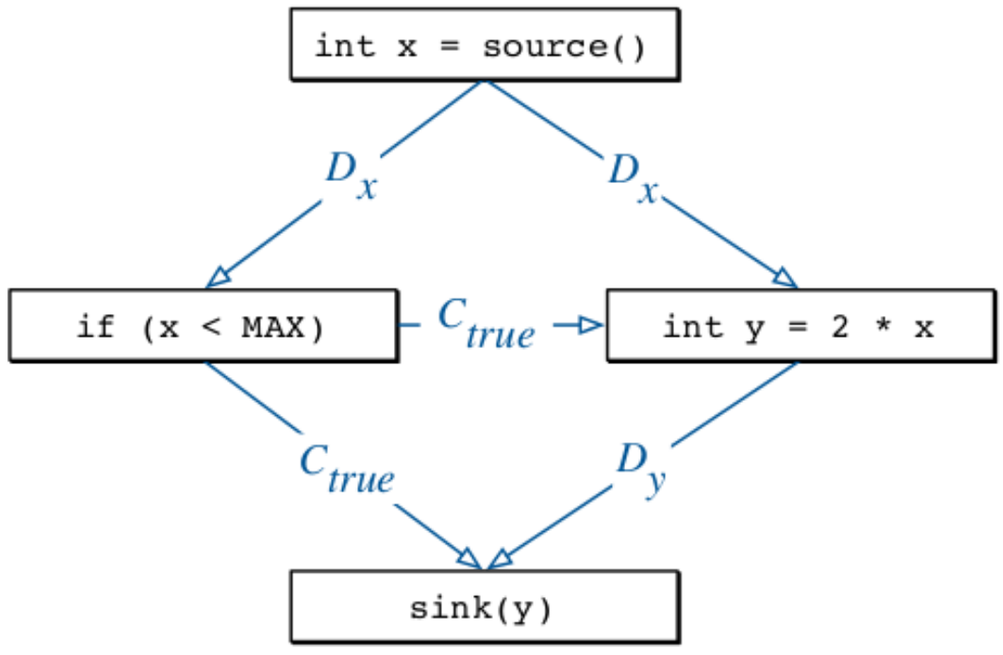
```
void foo()
{
    int x = source();
    if (x < MAX)
    {
        int y = 2 * x;
        sink(y);
    }
}
```



(b) Control flow graph (CFG)

How compilers analyze data flow: dependence graphs

```
void foo()
{
    int x = source();
    if (x < MAX)
    {
        int y = 2 * x;
        sink(y);
    }
}
```



(c) Program dependence graph (PDG)

Merge: Code Property Graphs

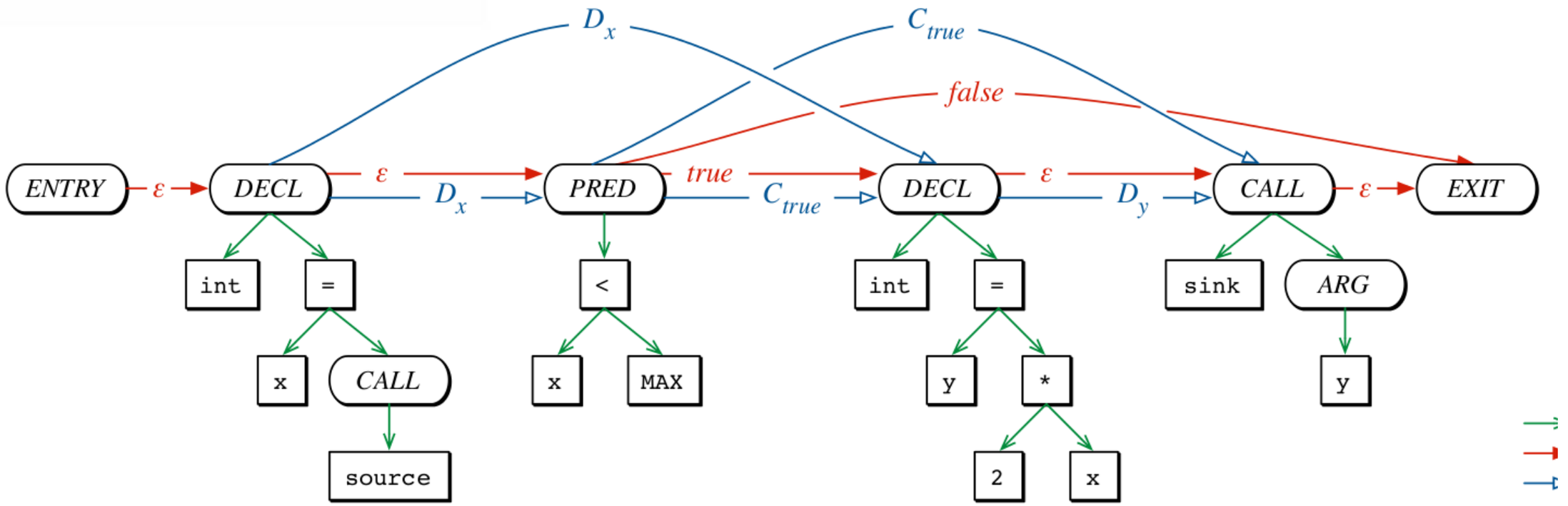
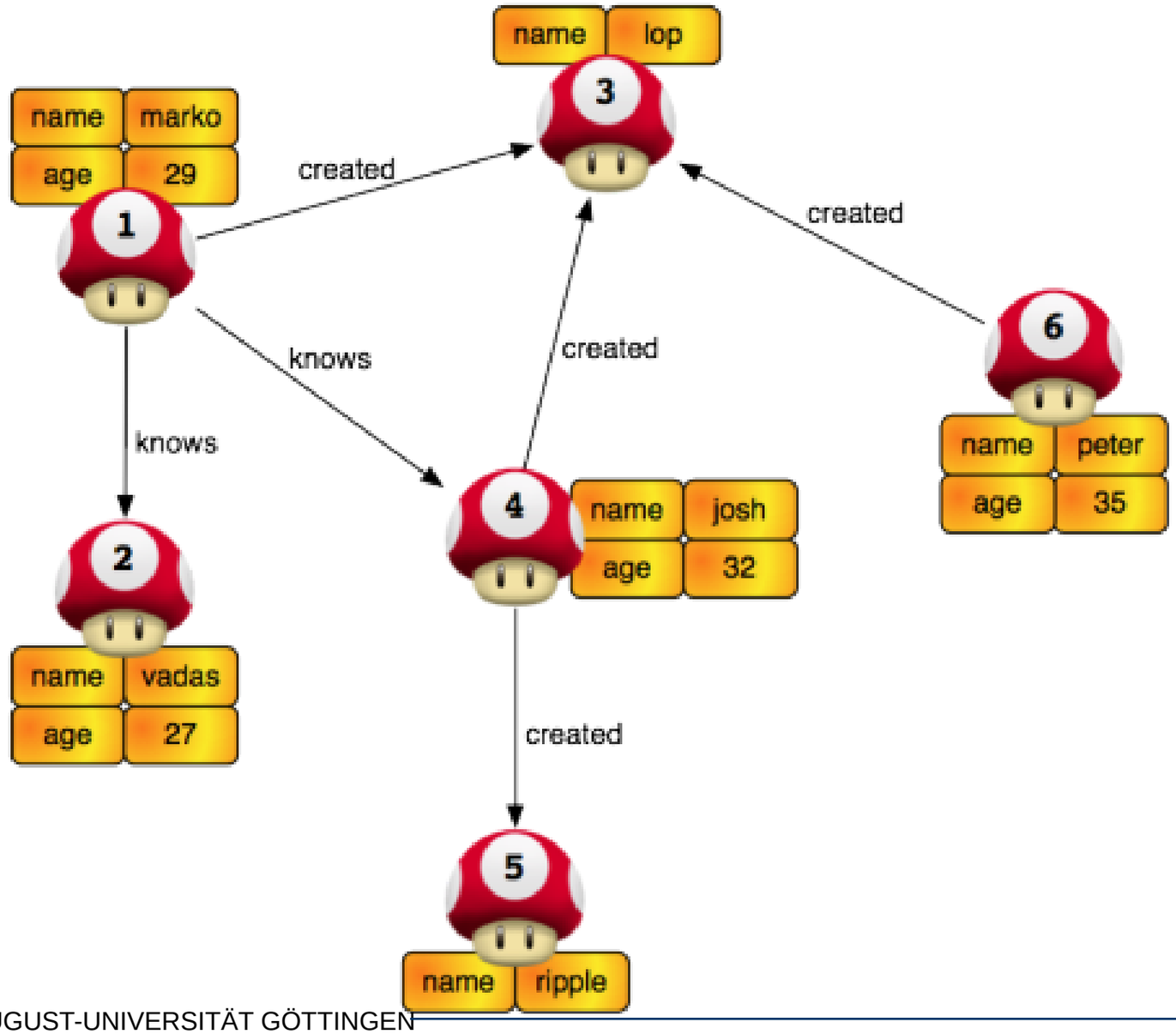


Fig. 4: Code property graph for the code sample given in Figure 1.

Graph databases! Big Data! Cloud-era!

- **Designed to store social networks**



Neo4j

From: http://markorodriguez.com/2011/08/03/on-the-nature-of-pipes/

GraphDBs: Not limited to storage of useless crap

» You can store code property graphs in graph databases

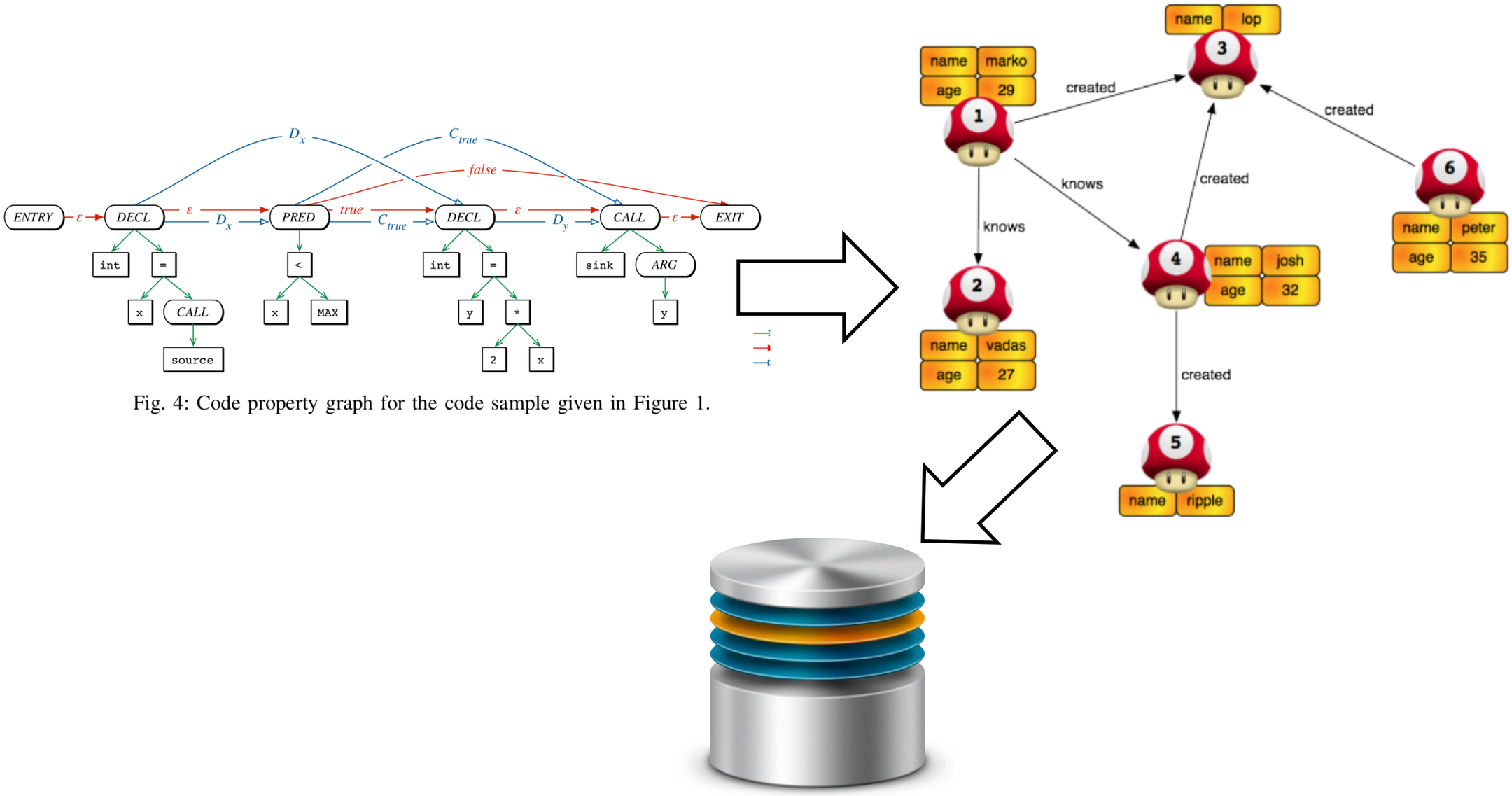


Fig. 4: Code property graph for the code sample given in Figure 1.

Overview of the System (Joern)

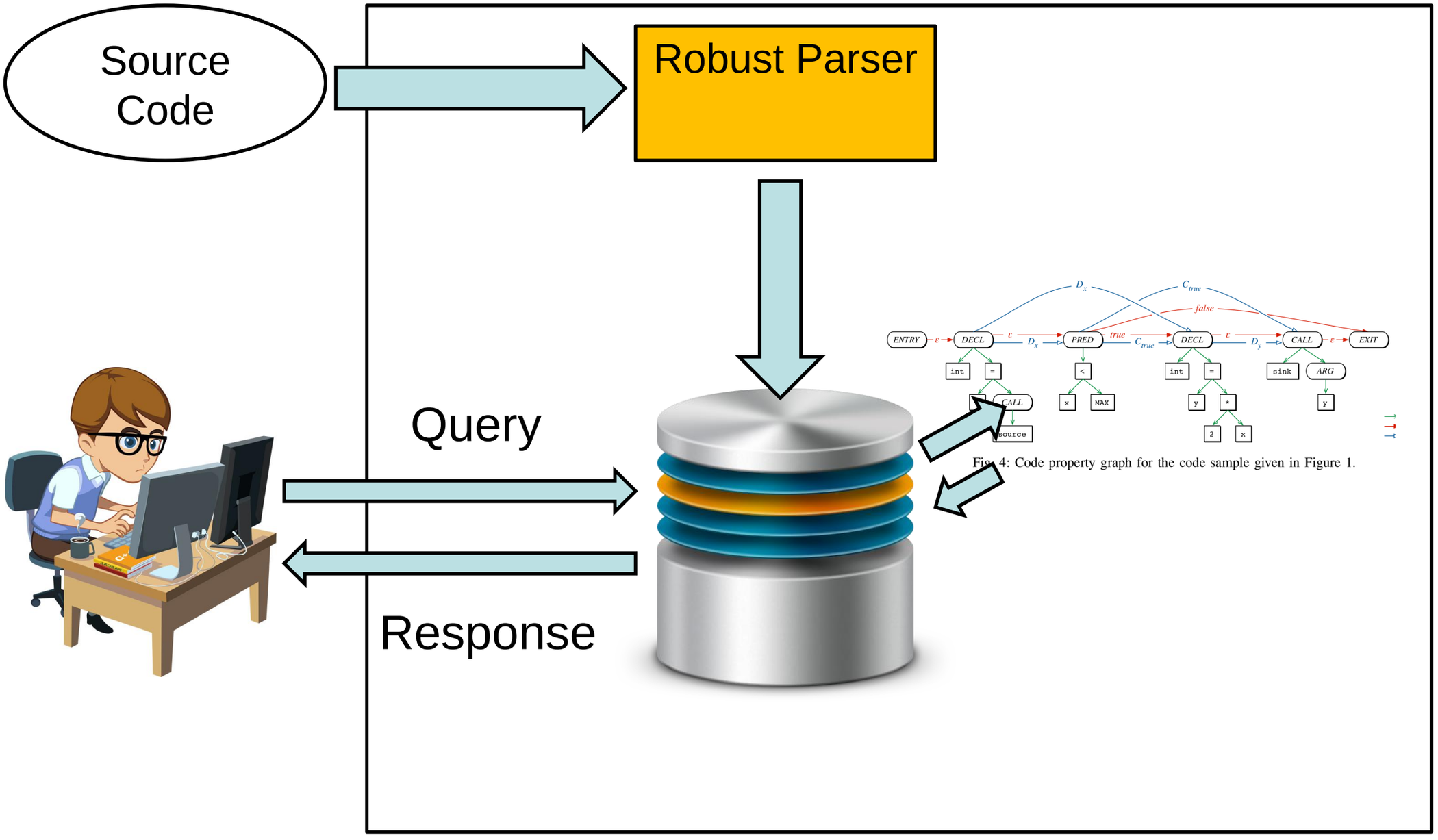


Fig. 4: Code property graph for the code sample given in Figure 1.

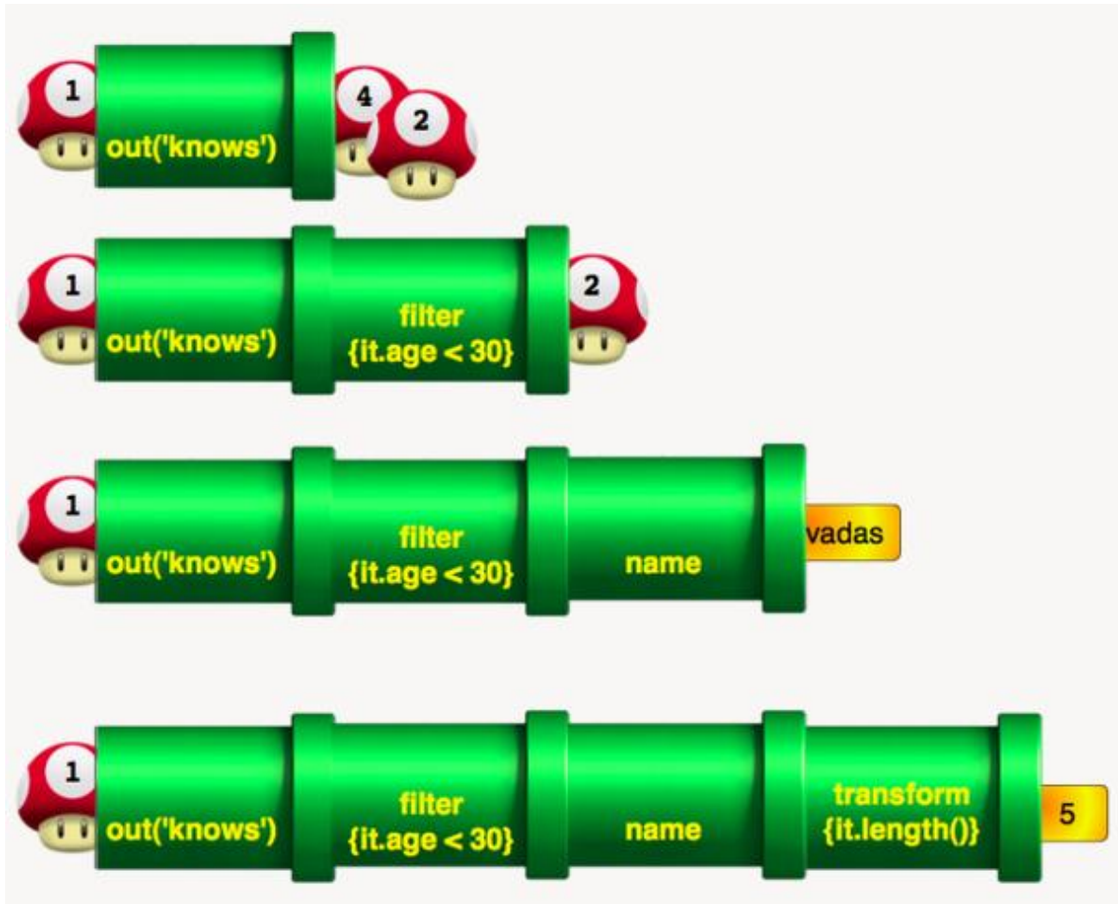
Querying the Database: Sending in the Gremlin

- Imperative language to traverse graphs by Marko Rodriguez
  - Turing complete, embedded groovy
  - Feels functional
  - Runs on top of Blueprints

- Traversals
  - Find starting nodes using an index
  - Describe how to walk the graph from the starting node by chaining elementary traversals (“pipes”)
  - Return all nodes reached

Gremlin – Chaining Pipes

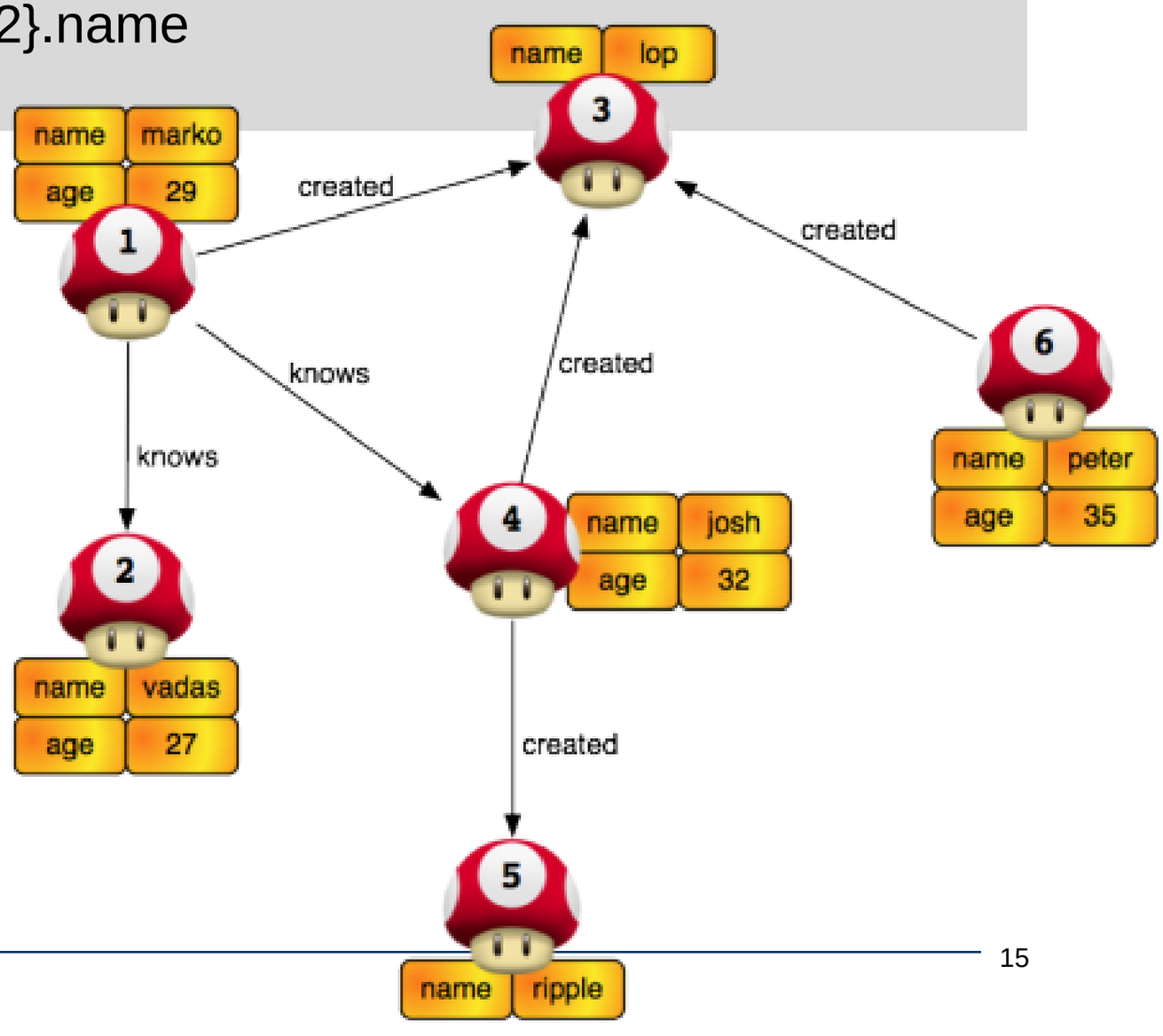
- A pipe maps sets of nodes to new sets of nodes
- Pipes are chained to describe traversals in the graph
- Chained pipes can be encapsulated in “custom pipes”



From: http://markorodriguez.com/2011/08/03/on-the-nature-of-pipes/

Gremlin in a Nuttshell

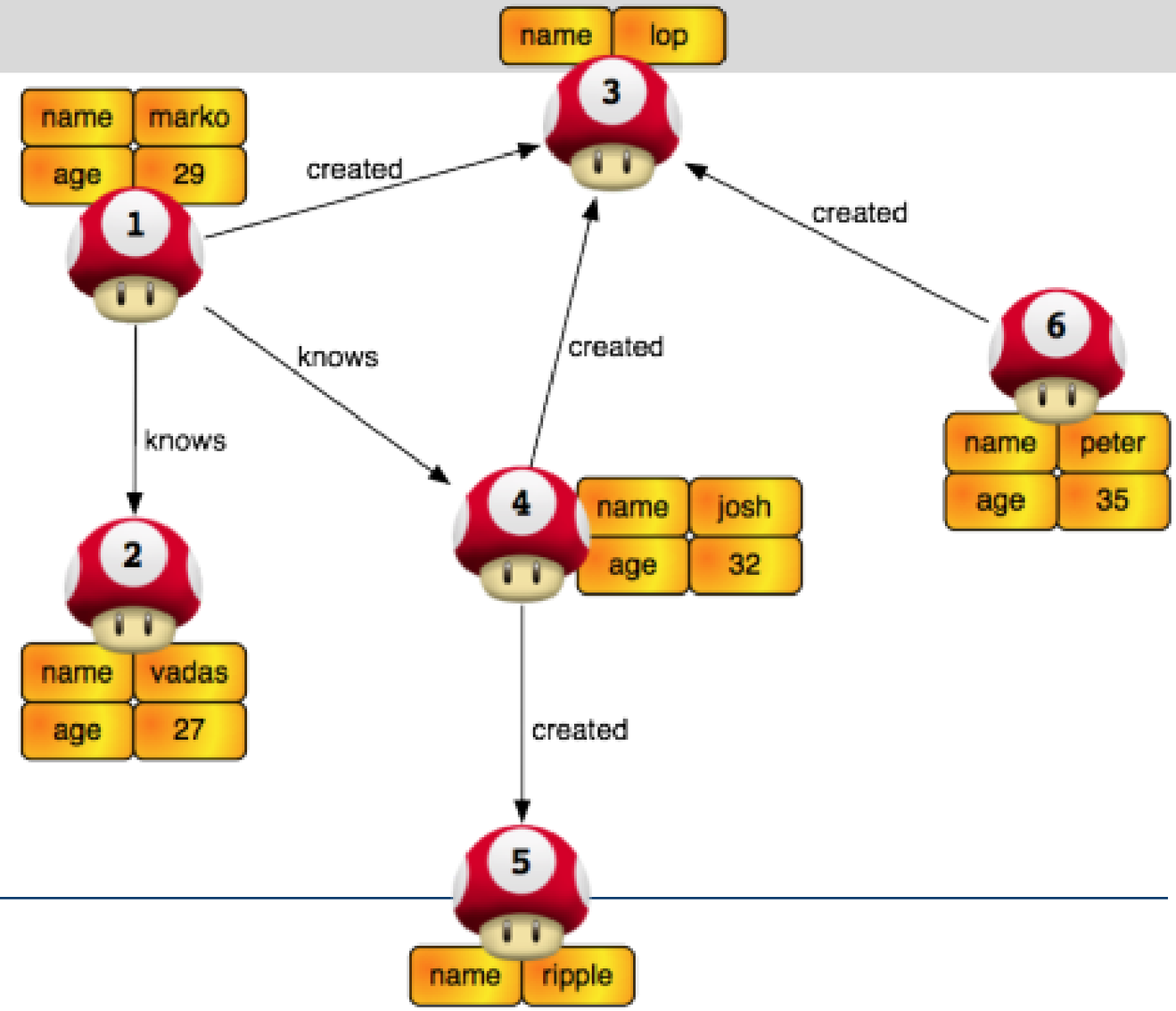
```
>> g.v(1).out('knows').name
vadas, josh
>> g.v(1).out('knows').filter{ it.age > 30}.name
josh
>> g.v(1).out().loop(1){ it.loops < 2}.name
ripple
```



Custom steps – Basis for code analysis with Gremlin

```
Gremlin.defineStep('twoHopsAway', [Vertex,Pipe], {
        _().out().out()
})

>> g.v(1).twoHopsAway().name
ripple
```



Steps for code analysis: python-joern

- Start node selection:
  - getArgs, getParameters, getAssignments…
- Elementary AST-steps
  - children(), parents(), ithChild(), astNodes()
- Node-specific utility steps for ASTs
  - callToIthArgument(), assignmentToLval(), …
- Data flow and control flow steps
  - sources(), definitions(), unsanitized(), paths() …
- A “language” for bug description

Simple example: multiplications inside args to malloc

```
$ echo "getArguments('malloc', '0')
        .astNodes().filter{it.type == 'MultiplicativeExpression'}.code“
| lookup.py -g
```

- Syntax-only:
- Get all first arguments to malloc
- Get all nodes of trees rooted at these (astNodes)
- Select only multiplicative expressions
- Get rid of ‘sizeof’
- Pipe to lookup.py of joern-tools

Building block for tainting – The “unsanitized” pipe

- It’s possible to build very complex pipes
  - Inject arbitrary groovy code into the DB!
- Meet the pipe ‘unsanitized(sanitizers)’
  - All CFG paths from sources to a sink where
    - A symbol is propagated from source to sink (data flow)
    - (No node on the path re-define the symbol)
    - No node on the path matches a sanitizer-description

Determining symbols used and possible sources

```
Gremlin.defineStep('unsanitized', [Vertex, Pipe], { sanitizer, src = { [1]._() }->
        _().sideEffect{ dst = it; }
        .uses().sideEffect{ symbol = it.code }
        .transform{ dst.producers([symbol]) }.scatter()
        .transform{
                cfgPaths(symbol, sanitizer, it, dst.statements().toList()[0] )
        }.scatter()
  .firstElem()
})
```

- What this query does
  - Determine symbols used by a statement
  - Determine producers of that symbol
  - Return first element of cfgPaths for (producer,symbol)

Determine CFG paths with a functional program

```
cfgPaths = { symbol, sanitizer, src, dst ->
        _cfgPaths(symbol, sanitizer, src, dst, [:], []) }

Object.metaClass._cfgPaths = {symbol, sanitizer, curNode, dst,
                                visited, path ->
  if(curNode == dst) return [path + curNode] as Set

  if( ( path != [] ) && isTerminationNode(symbol, sanitizer, curNode, visited))
    return [] as Set

  def X = [] as Set; def x;
  def children = curNode._().out(CFG_EDGE).toList()
  for(child in children){
    def curNodeId = curNode.toString()
    x = _cfgPaths(symbol, sanitizer, child, dst,
                  visited + [ (curNodeId) : (visited.get(curNodeId, 0) + 1)],
                  path + curNode)
    X += x
  } X
}
```

“Functional feel”

```
cfgPaths = { symbol, sanitizer, src, dst ->
        _cfgPaths(symbol, sanitizer, src, dst, [:], []) }

Object.metaClass._cfgPaths = {symbol, sanitizer, curNode, dst,
                                visited, path ->
  if(curNode == dst) return [path + curNode] as Set

  if( ( path != [] ) && isTerminationNode(symbol, sanitizer, curNode, visited))
    return [] as Set

  def children = curNode._().out(CFG_EDGE).toList()
  for(child in children){
        def curNodeId = curNode.toString()

        X += _cfgPaths(symbol, sanitizer, child, dst,
                visited + [ (curNodeId) : (visited.get(curNodeId, 0) + 1)],
                path + curNode)
  } X
}
```

Taking Joern for a spin on the Linux Kernel

- Crafted queries for four different types of vulnerabilities
  - Buffer overflows
  - Zero-byte allocation
  - Memory mapping bugs
  - Memory disclosure
- Let’s take a look at the buffer overflow examples

Query for buffer overflows in write handlers (Joern 0.2)

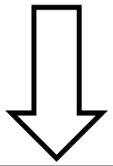
```
query1 = “””
getFunctionASTsByName('*_write*')
.getArguments('(copy_from_user OR memcpy)', '2')
.sideEffect{ paramName = 'c(ou)?nt'; }
.filter{ it.code.matches(paramName) }
.unsanitized(
        { it._().or(
          _().isCheck('.*' + paramName + '.*'),
          _().codeContains('.*alloc.*' + paramName + '.*'),
          _().codeContains('.*min.*')
        )}
)
.param( '.*c(ou)?nt.*' )
.locations() “””
```

Query for qeth-style buffer overflows (Joern 0.2)

```
query2 = “””
getArguments('(copy_from_user OR memcpy)', '2')
.filter{ !it.argToCall().toList()[0].code.matches('.*(sizeof|min).*')  }
.sideEffect{ argument = it.code; } /* store argument */
.sideEffect{ dstId = it.statements().toList()[0].id; } /* store id of sink */
.unsanitized({
        it._().or(
          _().isCheck('.*' + Pattern.quote(argument) + '.*') ,
          _().codeContains('.*alloc.*' + Pattern.quote(argument) + '.*'),
          _().codeContains('.*min.*')
        )
}, { it._().filter{ it.code.contains('copy_from_user')} }
)
.filter{ it.id != dstId } /* filter flows from node to self */
.locations()”””
```

Profit: Seven 0-days in eleven hits

Running queries 1 and 2



Filename	Function
arch/um/kernel/exitcode.c	exitcode_proc_write
security/smack/smackfs.c	smk_write_rules_list
drivers/staging/ozwpan/ozcdev.c	oz_cdev_write
drivers/infiniband/hw/ipath/ipath_diag.c	ipath_diagpkt_write
drivers/infiniband/hw/qib/qib_diag.c	qib_diagpkt_write
drivers/scsi/megaraid/megaraid_mm.c	mimd_to_kioc
drivers/scsi/megaraid.c	megadev_ioctl
drivers/char/xilinx_.../xilinx_hwicap.c	hwicap_write
drivers/s390/net/qeth_core_main.c	qeth_snmp_command
drivers/staging/wlags49_h2/wl_priv.c	wvlan_uil_put_info
arch/ia64/sn/kernel/sn2/sn_hwperf.c	sn_hwperf_ioctl

Search queries for four different types of bugs

Type	Location	Developer Feedback	Identifier
Buffer Overflow	`arch/um/kernel/exitcode.c`	Fixed	CVE-2013-4512
Buffer Overflow	`drivers/staging/ozwpan/ozcdev.c`	Fixed	CVE-2013-4513
Buffer Overflow	`drivers/s390/net/qeth_core_main.c`	Fixed	CVE-2013-6381
Buffer Overflow	`drivers/staging/wlags49_h2/wl_priv.c`	Fixed	CVE-2013-4514
Buffer Overflow	`drivers/scsi/megaraid/megaraid_mm.c`	Fixed	-
Buffer Overflow	`drivers/infiniband/hw/ipath/ipath_diag.c`	Fixed	-
Buffer Overflow	`drivers/infiniband/hw/qib/qib_diag.c`	Fixed	-
Memory Disclosure	`drivers/staging/bcm/Bcmchar.c`	Fixed	CVE-2013-4515
Memory Disclosure	`drivers/staging/sb105x/sb_pci_mp.c`	Fixed	CVE-2013-4516
Memory Mapping	`drivers/video/au1200fb.c`	Fixed	CVE-2013-4511
Memory Mapping	`drivers/video/au1100fb.c`	Fixed	CVE-2013-4511
Memory Mapping	`drivers/uio/uio.c`	Fixed	CVE-2013-4511
Memory Mapping	`drivers/staging/.../drv_interface.c`	Fixed	-
Memory Mapping	`drivers/gpu/drm/i810/i810_dma.c`	Fix underway	-
Zero-byte Allocation	`fs/xfs/xfs_ioctl.c`	Fixed	CVE-2013-6382
Zero-byte Allocation	`fs/xfs/xfs_ioctl32.c`	Fixed	CVE-2013-6382
Zero-byte Allocation	`drivers/net/wireless/libertas/debugfs.c`	Fixed	CVE-2013-6378
Zero-byte Allocation	`drivers/scsi/aacraid/commctrl.c`	Fixed	CVE-2013-6380

» 18 zero-days, acknowledged /fixed by developers

Thank you! Questions?

- Tool:
  - http://mlsec.org/joern
  - http://github.com/fabsx00/joern
- Fabian Yamaguchi
  - fabs@goesec.de
  - http://codeexploration.blogspot.de
  - Twitter: @fabsx00