

r2c

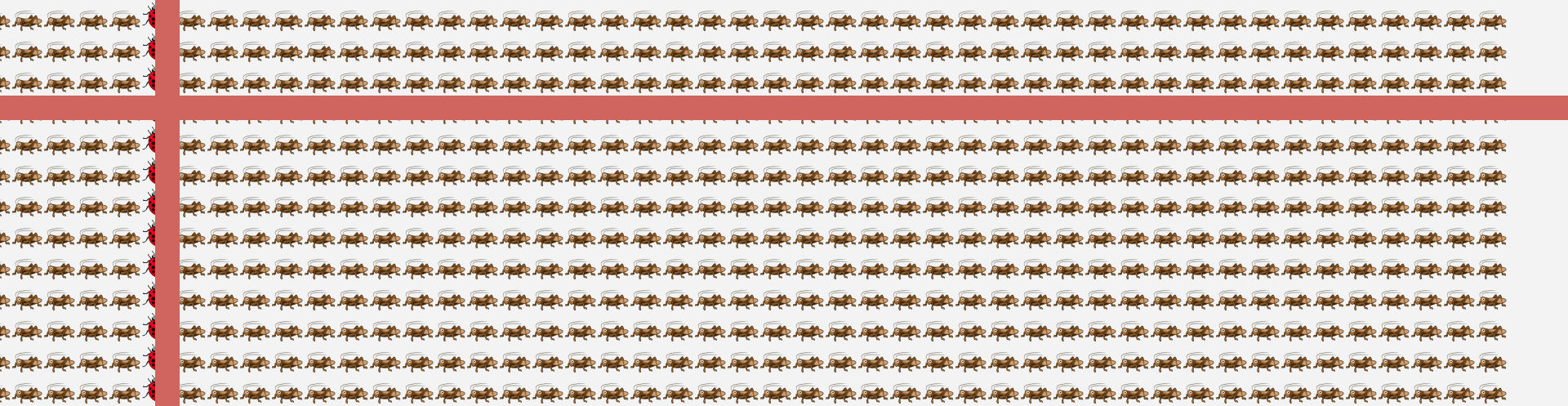
(lost? behind? r2c.dev/defcon or ask a co-instructor)

@r2cdev

r2c.dev

discover, exploit & eradicate

entire vulnerability classes at scale



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Let’s get you on the platform!

Please go to the following Google form and let us know your GitHub username (and your real email if you’d like to be added to our Slack):

https://r2c.dev/defcon-signup

You will need to **sign in to a GitHub account later** (if you didn’t bring a 2FA token -- make a burner GitHub account now)

(lost? behind? r2c.dev/defcon or ask a co-instructor)

$ whois r2c

Team of 15 at a security/developer tools startup

Help developers code more confidently and write more secure code

Relevant focus: find large numbers of vulnerabilities (where others might not be looking!)

r2c = return-to-corp (a.l.a. return-to-libc)

$ whois “Isaac Evans”

- CEO @ r2c
- binary exploitation bypasses for techniques like control-flow integrity and hardware defenses on new architectures like RISC-V @ US DoD + MIT Lincoln Laboratory
- Made small contributions to Ghidra (ghidra-sre.org)
- BS+MS EECS at MIT

(lost? behind? r2c.dev/defcon or ask a co-instructor)

$ Co-instructors and TAs

Don’t hesitate to ask us questions!

Look for the shirts with R’s on them!

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Let’s talk!

Join the workshop discord channel to discuss problems with other attendees and the TAs!

r2c.dev/defcon-discord

(lost? behind? r2c.dev/defcon or ask a co-instructor)

agenda

15min: Short demo of what's possible: idea to bug bounty report

45min: What is program analysis?
What are the current tools available to analyze source?

45min: Writing your first program analysis

15min: Fetch lunch

1hr: Writing a program analysis that actually looks for something interesting

1hr: More complex analysis, refine the analysis TP/FP rate

Till end: Hunting time :)

(lost? behind? r2c.dev/defcon or ask a co-instructor)

part 1/5

Demo: analysis on ecosystems

(lost? behind? r2c.dev/defcon or ask a co-instructor)

cd ~/demo

setup: let's precache

If you miss something, refer to r2c.dev/defcon

If you are running Windows or Linux*:

r2c.dev/defcon-vm **or get a flash drive from a co-instructors**

(*Ubuntu 18 or Mac OSX are officially supported: you won't need a VM)

part 2/5

What is program analysis?

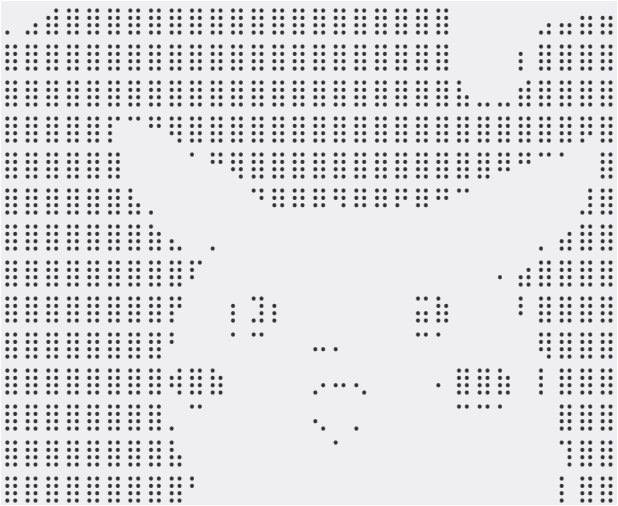
(lost? behind? r2c.dev/defcon or ask a co-instructor)

“What’s the worst that can happen?”



(lost? behind? r2c.dev/defcon or ask a co-instructor)

“What~~’s the worst that~~ can happen?”



(lost? behind? r2c.dev/defcon or ask a co-instructor)

What kind of questions?

- What values can this variable have?
- What types happen here?
- Who can influence this variable?
- How can the program get here?
- Who influences the program getting here?
- “What does this code do?”
- “What doesn’t this code do?”

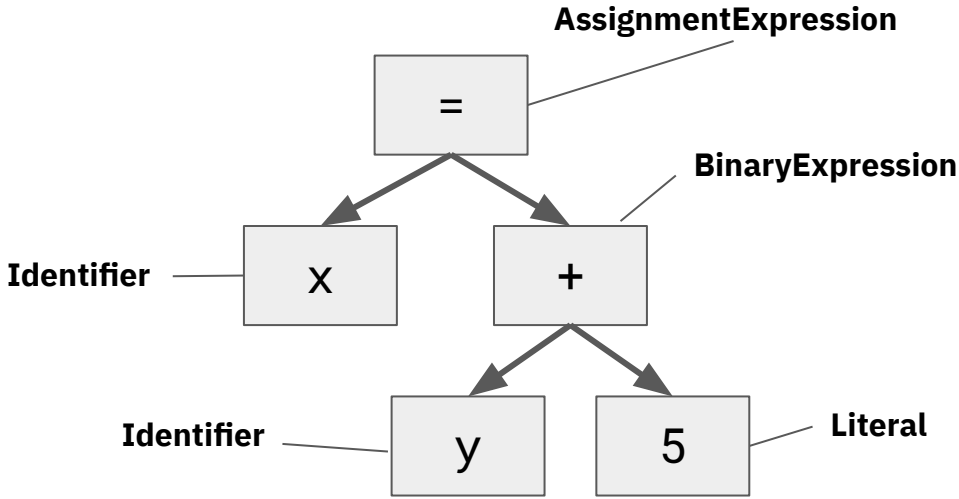
What kind of problems?

- Validation, escaping sandbox, ACLs
- API usage, ACLs, correctness
- User information leakage, buffer overflows
- ACLs, error detection, find usages
- Return to libc, bypassing validation
- Hard crashes

What is program analysis?

- Answer some question, measure some property about a program
- Programs are trees
- Many analyses reduce to *graph problems* with *domain-specific adaptations*
- Let's look closely at a tree and then write a simple analysis

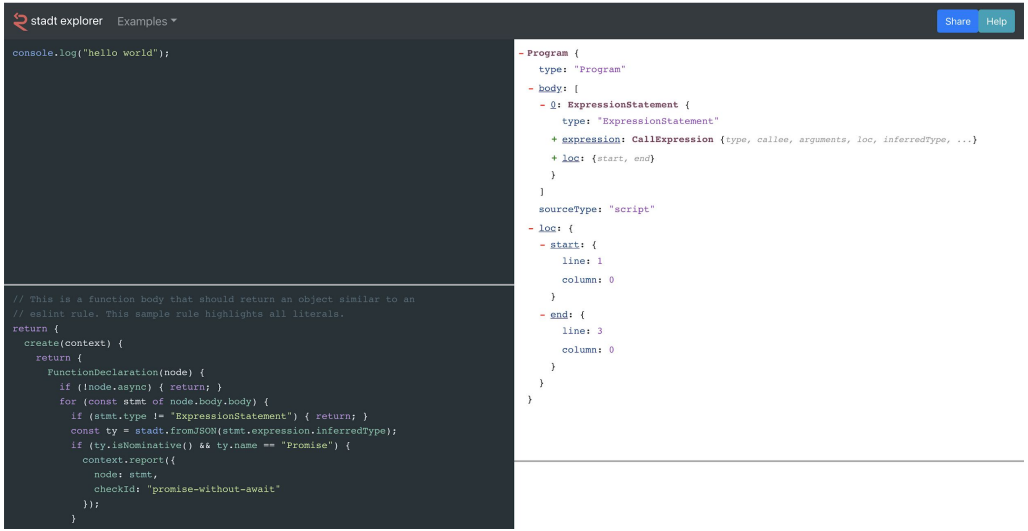
x = y + 5;



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Lab 0: **Abstract Syntax Trees**

Step 1: open https://r2c-explorer.dev



You are looking at a JSON representation of an *estree*, the representation used by Mozilla's Javascript parser (Javascript is formally specified as ECMAScript -- hence *es*)

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Lab 0: **Abstract Syntax Trees**

Step 1: pull up https://r2c-explorer.dev

Step 2: `x = y + 5;`

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Lab 0: **Abstract Syntax Trees**

Step 1: pull up https://r2c-explorer.dev

Step 2: `x = y + 5;`



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Lab 0: **Abstract Syntax Trees**

Step 1: pull up https://r2c-explorer.dev

Step 2: `x = y + 5;`

Step 3: `console.log("Hello, world!");`

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Lab 0: **Abstract Syntax Trees**

Step 1: pull up https://r2c-explorer.dev

Step 2: `x = y + 5;`

Step 3: `console.log("Hello, world!");`



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Lab 0: **Abstract Syntax Trees**

Step 1: pull up https://r2c-explorer.dev

Step 2: `x = y + 5;`

Step 3: `console.log("Hello, world!");`

Step 4: `x.map(i => i + 1)`

(valid AST, but not a valid program!)

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Lab 0: **Abstract Syntax Trees**

Step 1: pull up https://r2c-explorer.dev

Step 2: `x = y + 5;`

Step 3: `console.log("Hello, world!");`

Step 4: `x.map(i => i + 1)`

(valid AST, but not a valid program!)



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Lab 0: **Abstract Syntax Trees**

Step 1: pull up https://r2c-explorer.dev

Step 2: `x = y + 5;`

Step 3: `console.log("Hello, world!");`

Step 4: `x.map(i => i + 1)`

Step 5: choose "async await" under Examples

```
const fs = require("fs");
async function f() {
  await fs.promises.writeFile("out.txt", "contents");
  fs.promises.writeFile("out.txt", "blah");
}
```

Lab 0: **Abstract Syntax Trees**

Step 1: pull up https://r2c-explorer.dev

Step 2: `x = y + 5;`

Step 3: `console.log("Hello, world!");`

Step 4: `x.map(i => i + 1)`

Step 5: choose "async await" under Examples

```
const fs = require("fs");
async function f() {
  await fs.promises.writeFile("out.txt", "contents");
  fs.promises.writeFile("out.txt", "blah");
}
```



Prereqs: OS

Follow along at: r2c.dev/defcon-slides

Tested/Supported OS:
macOS 10.14, Ubuntu 18.10

Ask instructors for a VM if you need it!

You can download it from r2c.dev/defcon-vm

You'll have to use Virtualbox. Login to the VM with:

Username: defcon password: defcon

Prereqs: tools

docker: a tool for running software in isolated containers

python3.6+: the programming language needed to run (but not to develop) analyzers

pip3: the Python package manager used to install and update r2c-cli

If you're on Ubuntu, make sure to follow these instructions to use Docker as non-root:

https://docs.docker.com/install/linux/linux-postinstall/#manage-docker-as-a-non-root-user

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Make sure everything is working

```
$ docker run hello-world
```

```
$ python3 --version # make sure its 3.6+
```

```
$ pip3 --version
```

N-min break!

What are ASTs good for? **Type Checking**

Simplest approach: bottom-up recursive AST type decoration

```
- typeAnnotation: TSTypeAnnotation {
    type: "TSTypeAnnotation"
  + loc: {start, end}
  + range: [2 elements]
  - typeAnnotation: TSNumberKeyword {
      type: "TSNumberKeyword"
    + range: [2 elements]
    + loc: {start, end}
    }
  }
+ inferredType: {kind}
  inferredTypeString: "number"
}
- init: BinaryExpression {
  type: "BinaryExpression"
  operator: "+"
+ left: Identifier {type, name, loc, inferredType, inferredTypeString}
+ right: Literal {type, value, raw, loc, inferredType, ...}
+ loc: {start, end}
- inferredType: {
    kind: "string"
  }
  inferredTypeString: "string"
```

```
1  const message: string = 'hello world';
2
3       var x: number
4       Type 'string' is not assignable to type 'number'.
5       Quick Fix...  Peek Problem
6  var x: number = message + 5
7  console.log(message);
```

(lost? behind? r2c.dev/defcon or ask a co-instructor)

What are ASTs good for? **Type Checking**

Simplest approach: bottom-up recursive AST type decoration

```
- typeAnnotation: TSTypeAnnotation {
    type: "TSTypeAnnotation"
  + loc: {start, end}
  + range: [2 elements]
  - typeAnnotation: TSNumberKeyword {
      type: "TSNumberKeyword"
    + range: [2 elements]
    + loc: {start, end}
    }
  }
+ inferredType: {kind}
  inferredTypeString: "number"
}
- init: BinaryExpression {
  type: "BinaryExpression"
  operator: "+"
+ left: Identifier {type, name, loc, inferredType, inferredTypeString}
+ right: Literal {type, value, raw, loc, inferredType, ...}
+ loc: {start, end}
- inferredType: {
    kind: "string"
  }
  inferredTypeString: "string"
```



```
1  const message: string = 'hello world';
2
3       var x: number
4       Type 'string' is not assignable to type 'number'.
5       Quick Fix...  Peek Problem
6  var x: number = message + 5
7  console.log(message);
```

What are ASTs good for? **Type Checking**

Simplest approach: bottom-up recursive AST type decoration

```
- typeAnnotation: TSTypeAnnotation {
    type: "TSTypeAnnotation"
  + loc: {start, end}
  + range: [2 elements]
  - typeAnnotation: TSNumberKeyword {
      type: "TSNumberKeyword"
    + range: [2 elements]
    + loc: {start, end}
    }
  }
+ inferredType: {kind}
  inferredTypeString: "number"
}
- init: BinaryExpression {
  type: "BinaryExpression"
  operator: "+"
+ left: Identifier {type, name, loc, inferredType, inferredTypeString}
+ right: Literal {type, value, raw, loc, inferredType, ...}
+ loc: {start, end}
- inferredType: {
    kind: "string"
  }
  inferredTypeString: "string"
```





```
1  const message: string = 'hello world';
2
3       var x: number
4       Type 'string' is not assignable to type 'number'.
5       Quick Fix...  Peek Problem
6  var x: number = message + 5
7  console.log(message);
```

Why Program Analysis: **Type Inference**

Much more powerful; TypeScript type inference:

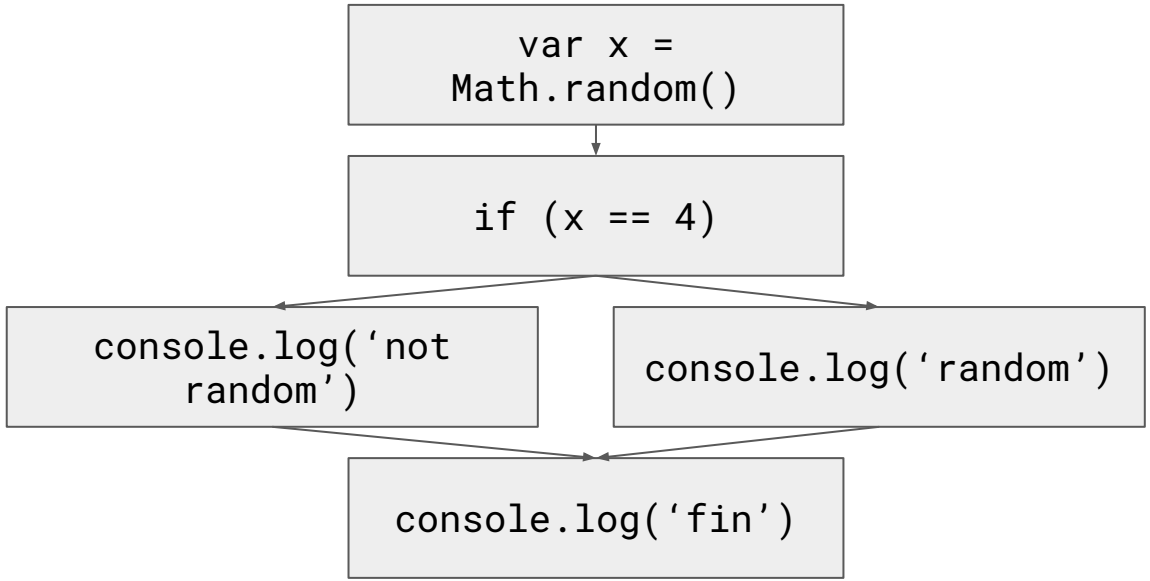
```
type S = { done: boolean, value: number }
type T =
    | { done: false, value: number }
    | { done: true, value: number };

declare let source: S;
declare let target: T;

target = source;
```

(from https://devblogs.microsoft.com/typescript/announcing-typescript-3-5/)

Kinds of Program Analysis: **Control-Flow Graph**

A graph where nodes are instructions and edges are possible transitions of the instruction pointer (execution)

```
var x = Math.random()
if (x == 4) {
  console.log('not random')
} else {
  console.log('random')
}
console.log('fin')
```



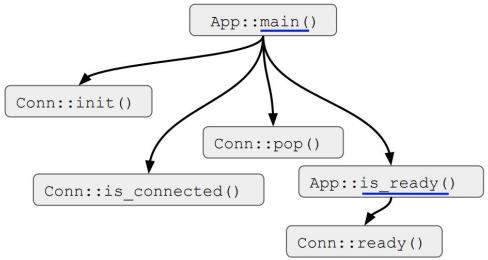
(lost? behind? r2c.dev/defcon or ask a co-instructor)

Kinds of Program Analysis: **Call Graph**

Nodes are functions; an edge exists from every function to the functions it calls

Why it's useful: does anyone even call this vulnerable function?

```
1  extern crate accepts;
2  extern crate send;
3
4  use accepts::Conn;
5  let is_ready = |conn: _| conn.ready();
6  fn main () {
7   let pipe = Conn::init();
8   let mut notifications = Vec::new();
9   while !is_ready(pipe);
10  // setup stuff
11  while pipe.is_connected() {
12   notifications.push(pipe.pop());
13  }
14 }
```



https://fosdem.org/2019/schedule/event/rust_praezi/

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Kinds of Program Analysis: **Data-Flow**

Data-flow analysis: state and information flow on top of the control flow graph

Why it's useful: Detect Heartbleed (OpenSSL bug from 2014)

- Can user-controlled input reach *write_length*?
- Where is *write_length* defined?
- What variables does its definition depend on?

```
+                                                    payload + padding;
                int r;

+               if (write_length > SSL3_RT_MAX_PLAIN_LENGTH)
+                       return 0;
+
                /* Allocate memory for the response, size is 1 byte
```

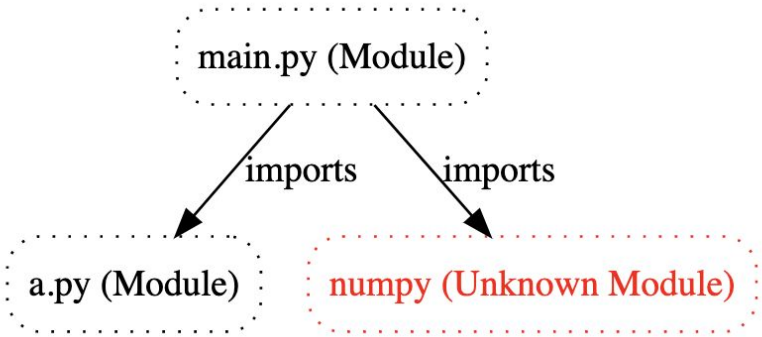
https://github.com/openssl/openssl/commit/96db9023b881d7cd9f379b0c154650d6c108e9a3

Kinds of Program Analysis: **Imports/Exports**

imports/exports analysis tells us:

- what functions in the entire module are *exported* (available to others)
- or what exported functions from other modules it references (*imports*)

Usage example: just because we depend on a library, doesn't mean we use the vulnerable function! (and we might even use the function without exercising the vulnerable behavior)



(from https://github.com/github/semantic/blob/master/docs/examples.md)

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Why Program Analysis: **Vulnerabilities!**

Find vulnerable patterns:

```
strcpy(a, b)
```

Prove that vulnerable patterns are exploitable:

```
strcpy(a, b)
          ^ b comes from user input
```

In some cases, construct PoC automatically! (Usually dynamic analysis does this)

```
Fuzzer: Attekett_surku_fuzzer
Crash Type: Heap-use-after-free READ 1
Crash Address: 0x6150003faab0
 - crash stack -
 syncer::SyncBackupManager::Init
 browser_sync::SyncBackendHostCore::DoInitialize
 - free stack -
 browser_sync::SyncBackendHostCore::DoDestroySyncManager
 browser_sync::SyncBackendHostCore::OnInitializationComplete
```

(from https://bugs.chromium.org/p/chromium/issues/detail?id=395410)

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Why use program analysis to find vulnerabilities at scale?

(lost? behind? r2c.dev/defcon or ask a co-instructor)

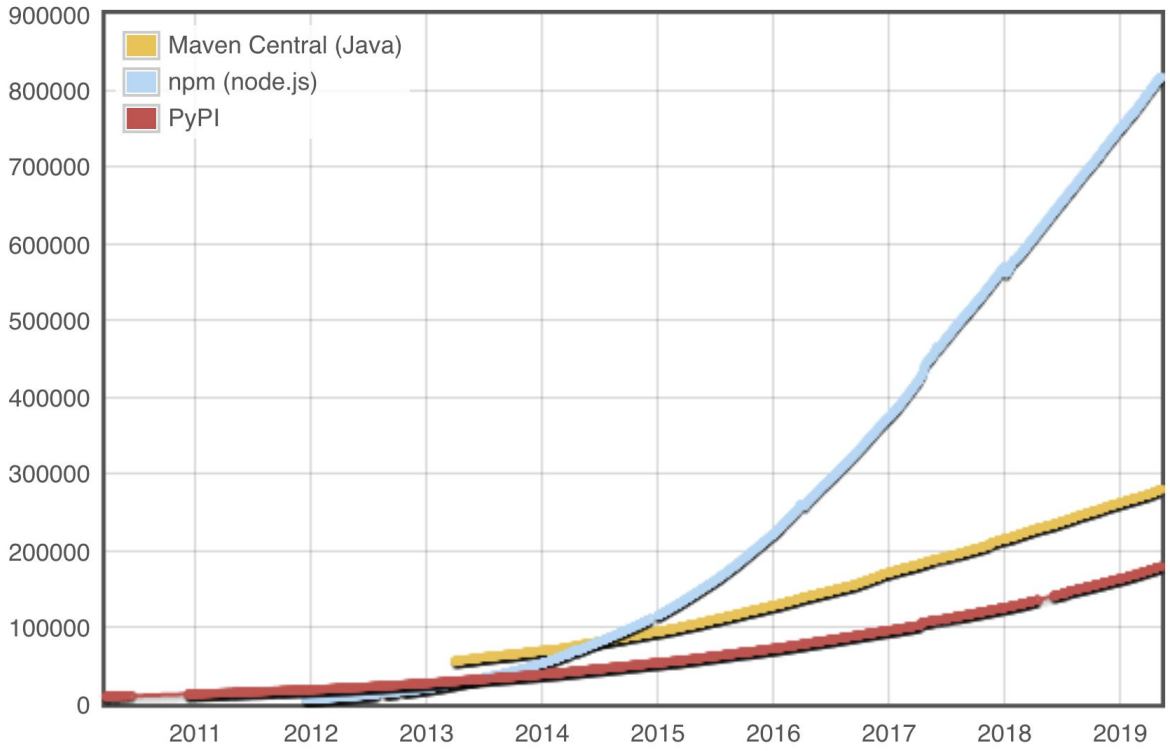
Standard libraries are going away

@dstufft, PyPi maintainer →



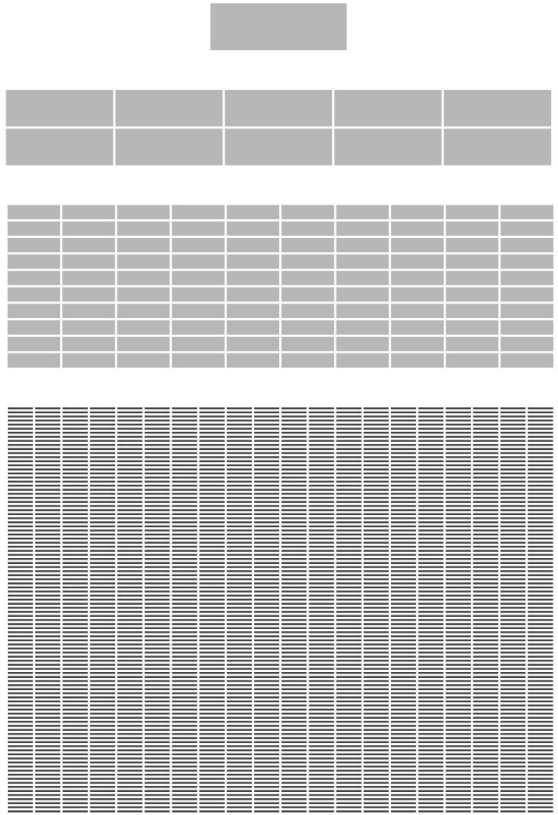
(lost? behind? r2c.dev/defcon or ask a co-instructor)

published packages vs. time



trust explosion

10x surface area growth per level



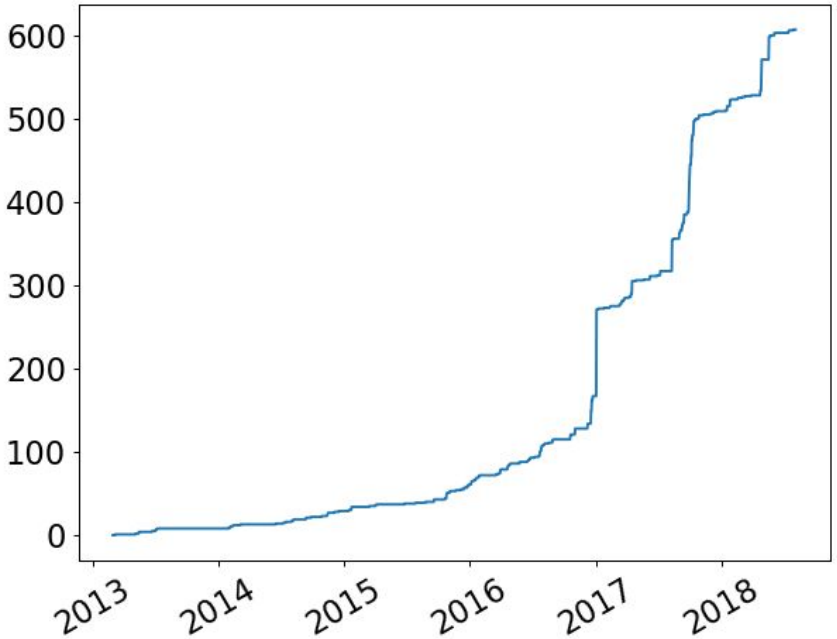
237
direct JS dependencies

1,918
Transitive JS dependencies

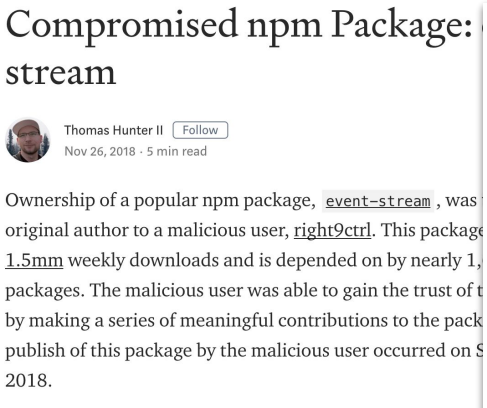
23,271
Independent code contributors

advisories published for npm vs. time

security analysis of these packages is nascent and one-off



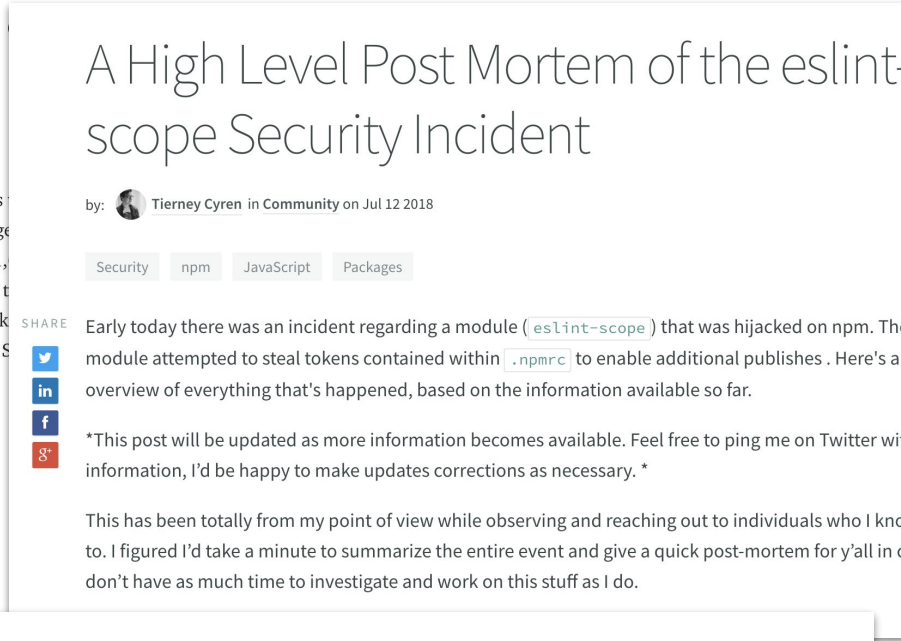
with real-world implications for targeted attacks



Compromised npm Package: stream

Thomas Hunter II Follow
Nov 26, 2018 · 5 min read

Ownership of a popular npm package, `event-stream`, was
original author to a malicious user, right9ctrl. This package
1.5mm weekly downloads and is depended on by nearly 1,
packages. The malicious user was able to gain the trust of t
by making a series of meaningful contributions to the pack
publish of this package by the malicious user occurred on S
2018.



A High Level Post Mortem of the eslint-scope Security Incident

by: Tierney Cyren in Community on Jul 12 2018

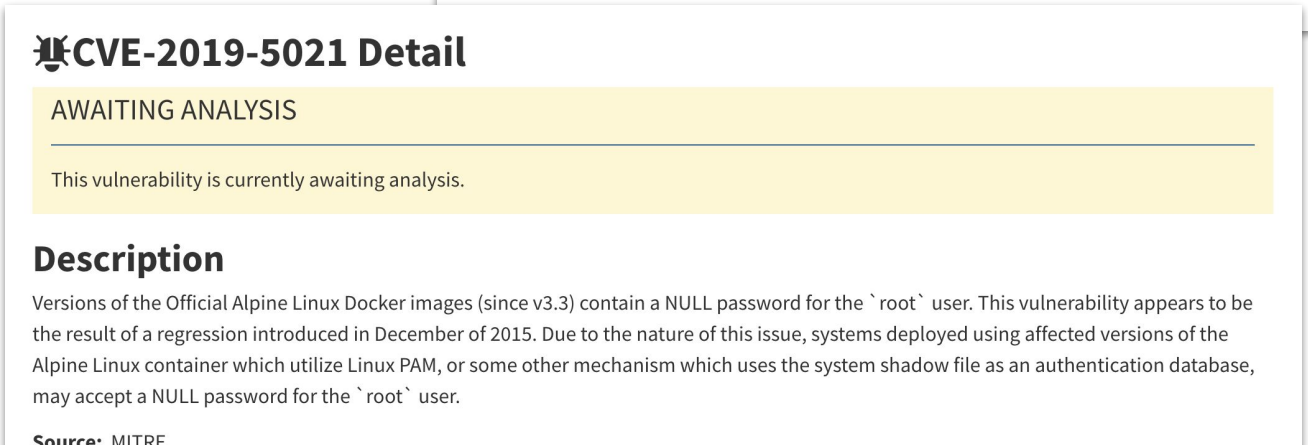
Security npm JavaScript Packages

SHARE

Early today there was an incident regarding a module (`eslint-scope`) that was hijacked on npm. Th
module attempted to steal tokens contained within `.npmrc` to enable additional publishes . Here's a
overview of everything that's happened, based on the information available so far.

*This post will be updated as more information becomes available. Feel free to ping me on Twitter wi
information, I'd be happy to make updates corrections as necessary. *

This has been totally from my point of view while observing and reaching out to individuals who I kn
to. I figured I'd take a minute to summarize the entire event and give a quick post-mortem for y'all in
don't have as much time to investigate and work on this stuff as I do.



CVE-2019-5021 Detail

AWAITING ANALYSIS

This vulnerability is currently awaiting analysis.

Description

Versions of the Official Alpine Linux Docker images (since v3.3) contain a NULL password for the `root` user. This vulnerability appears to be the result of a regression introduced in December of 2015. Due to the nature of this issue, systems deployed using affected versions of the Alpine Linux container which utilize Linux PAM, or some other mechanism which uses the system shadow file as an authentication database, may accept a NULL password for the `root` user.

Source: MITRE

(lost? behind? r2c.dev/defcon or ask a co-instructor)

part 3/5

Let’s write some analysis!



security & privacy caveats

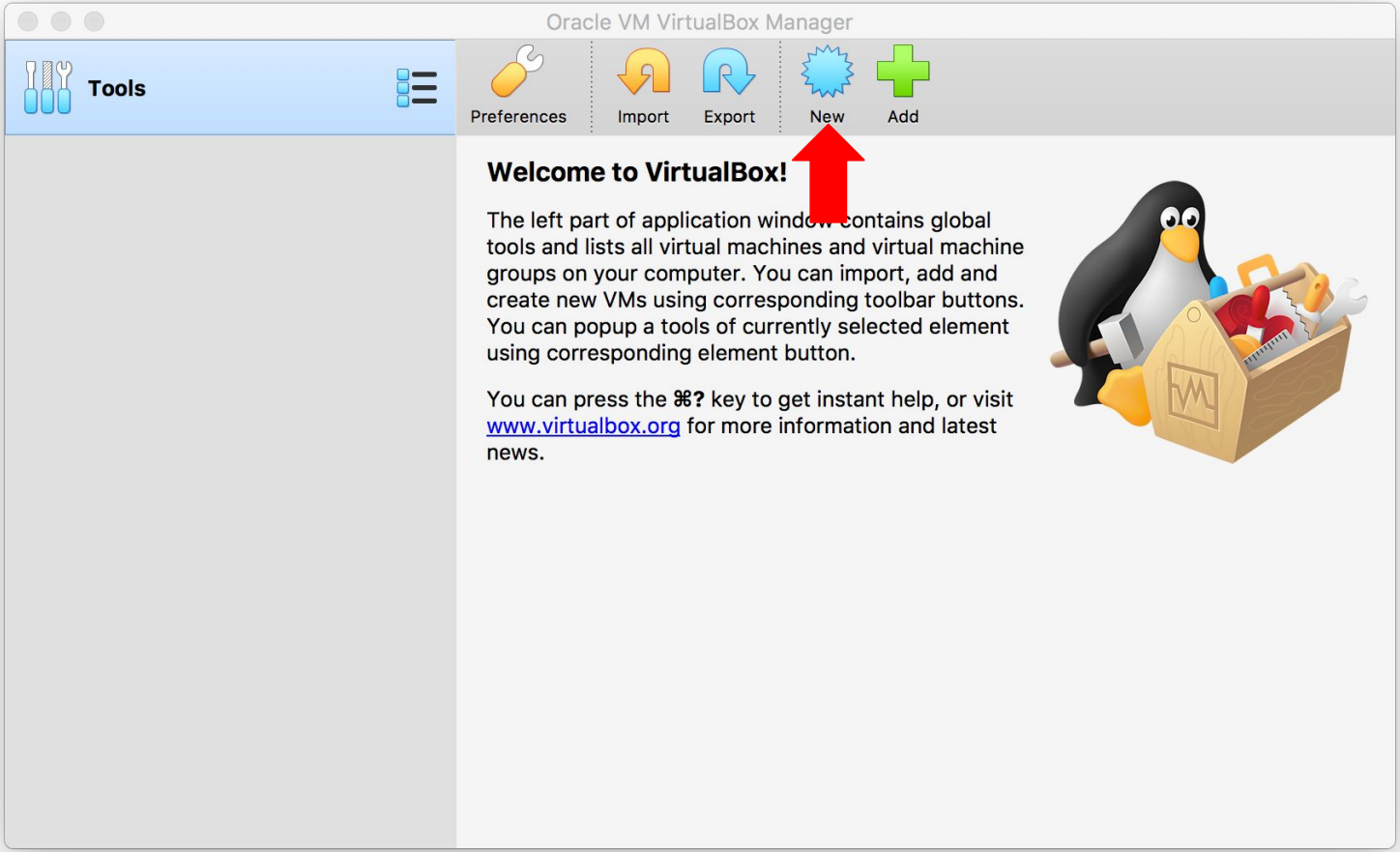
r2c is in beta; so at the moment please assume the following:

- no expectation of privacy for pushed code
- running an analyzer = arbitrary code execution (maybe)

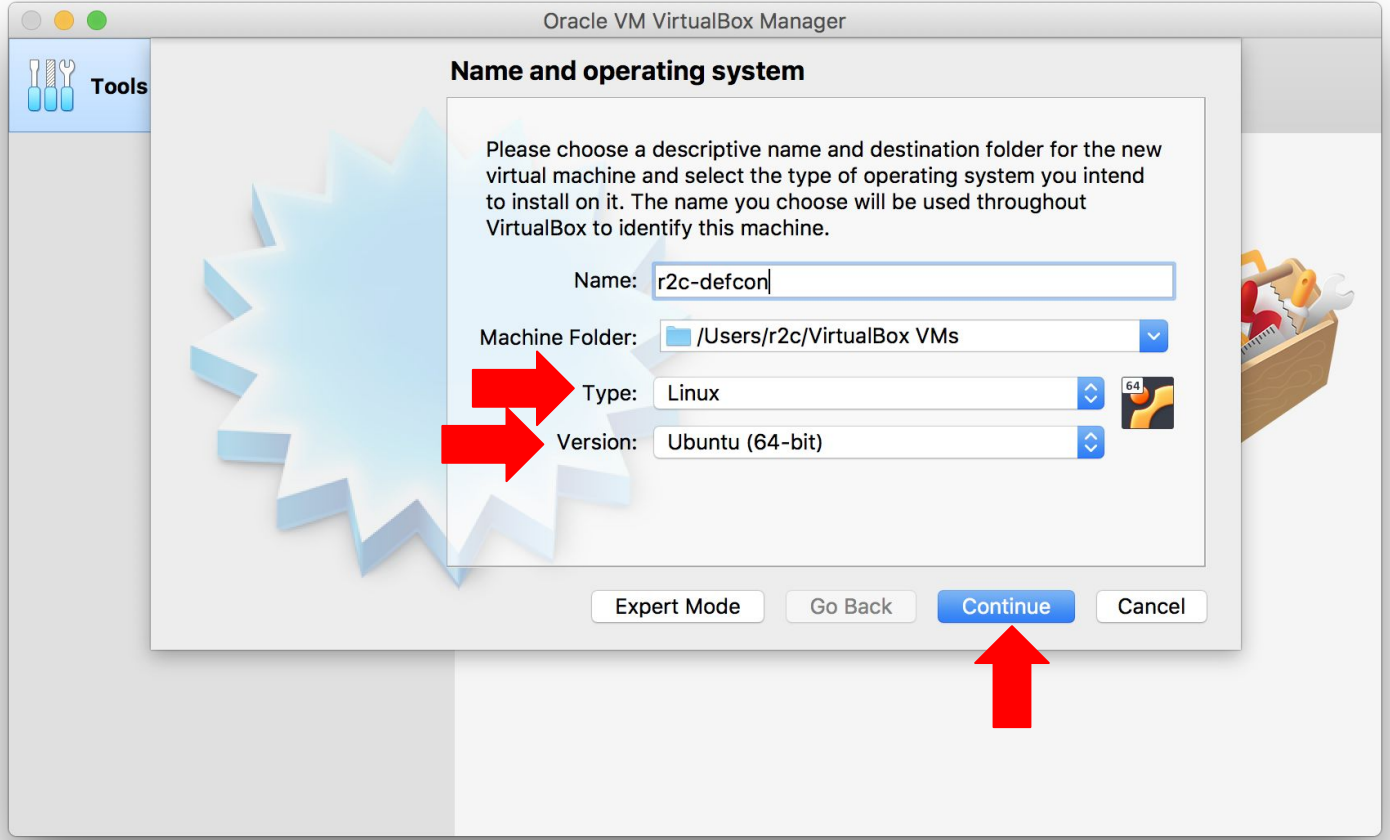
(lost? behind? r2c.dev/defcon or ask a co-instructor)

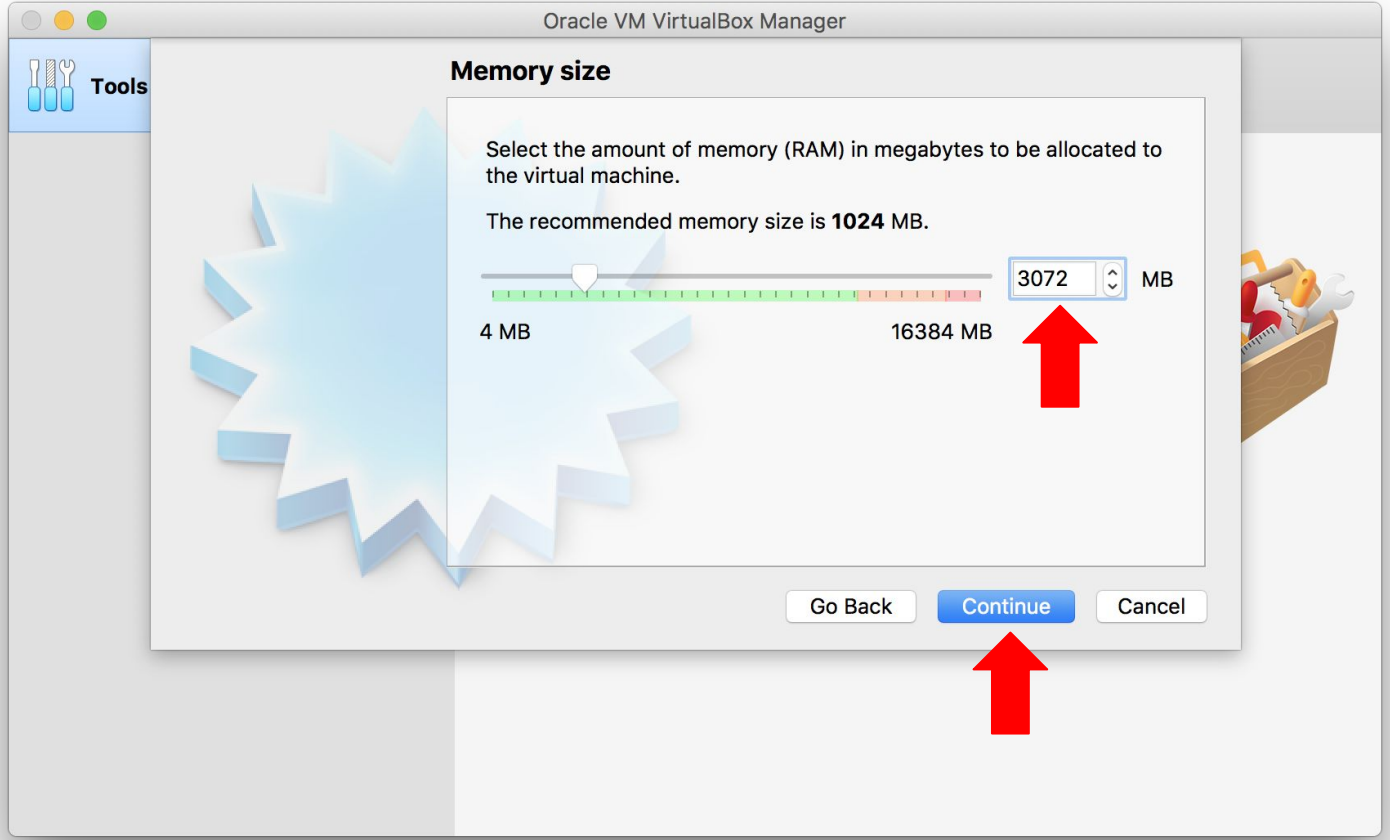
why build analyzers in r2c format?

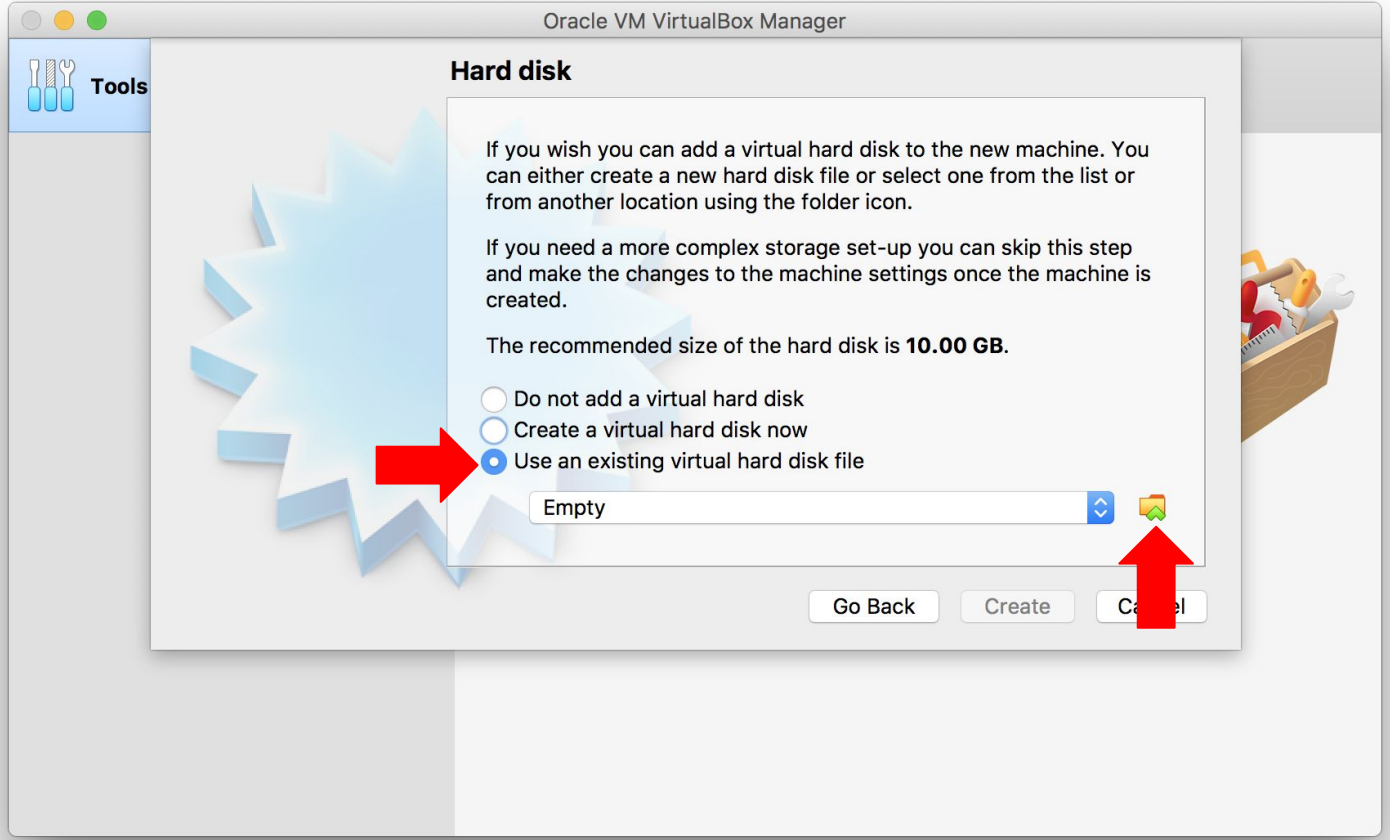
- **open source** CLI, tools, and docs
- easy to be **deterministic** (Docker!)
- infra for running at **scale**
- **composable** outputs with analyzers written by others


Oracle VM VirtualBox Manager
Tools
Preferences
Import
Export
New
Add
Welcome to VirtualBox!
The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.
You can press the ⌘? key to get instant help, or visit www.virtualbox.org for more information and latest news.

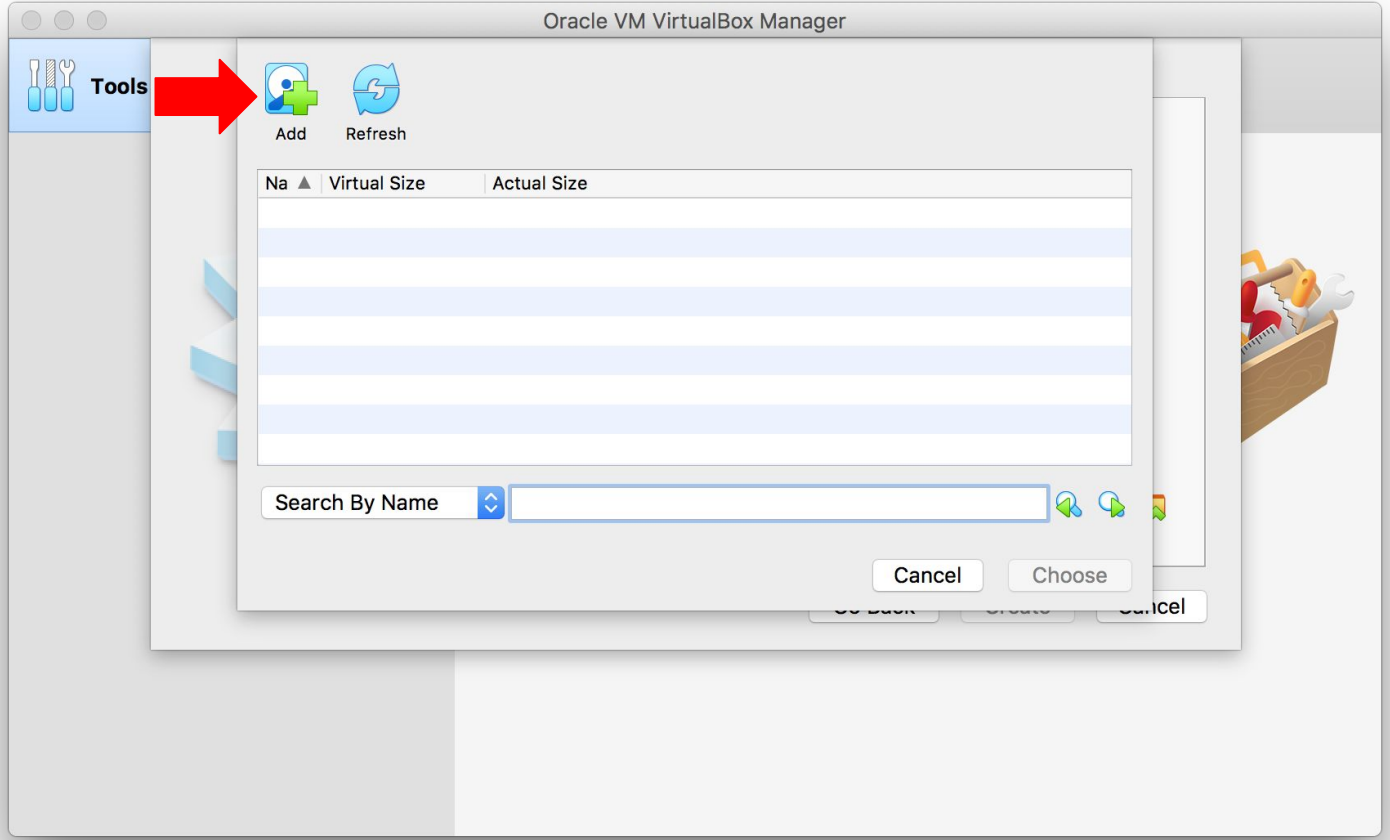
(lost? behind? r2c.dev/defcon or ask a co-instructor)



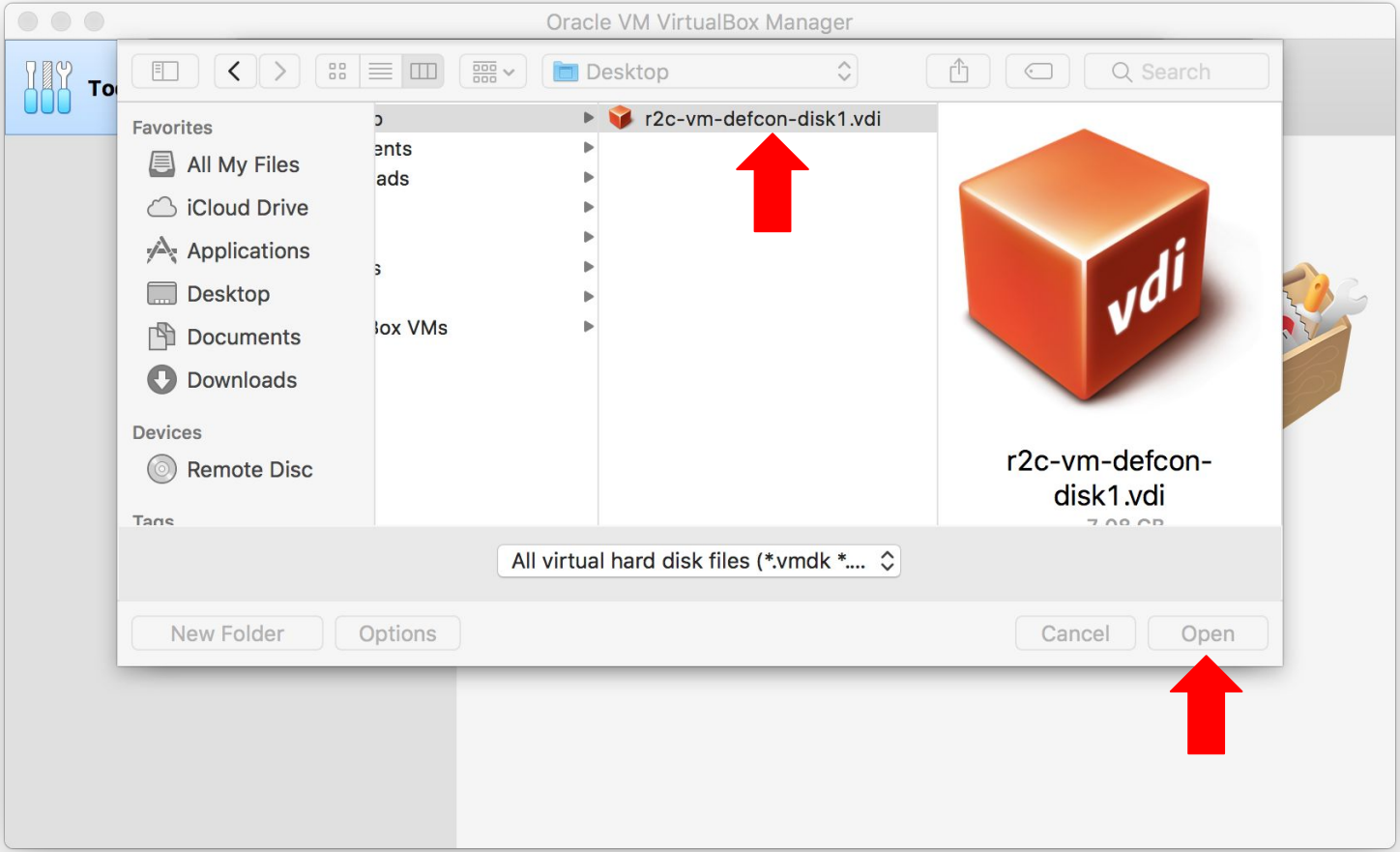

Oracle VM VirtualBox Manager
Tools
Memory size
Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.
The recommended memory size is 1024 MB.
3072
MB
4 MB
16384 MB
Go Back
Continue
Cancel


Oracle VM VirtualBox Manager
Tools
Hard disk
If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.
The recommended size of the hard disk is 10.00 GB.
Do not add a virtual hard disk
Create a virtual hard disk now
Use an existing virtual hard disk file
Empty
Go Back
Create

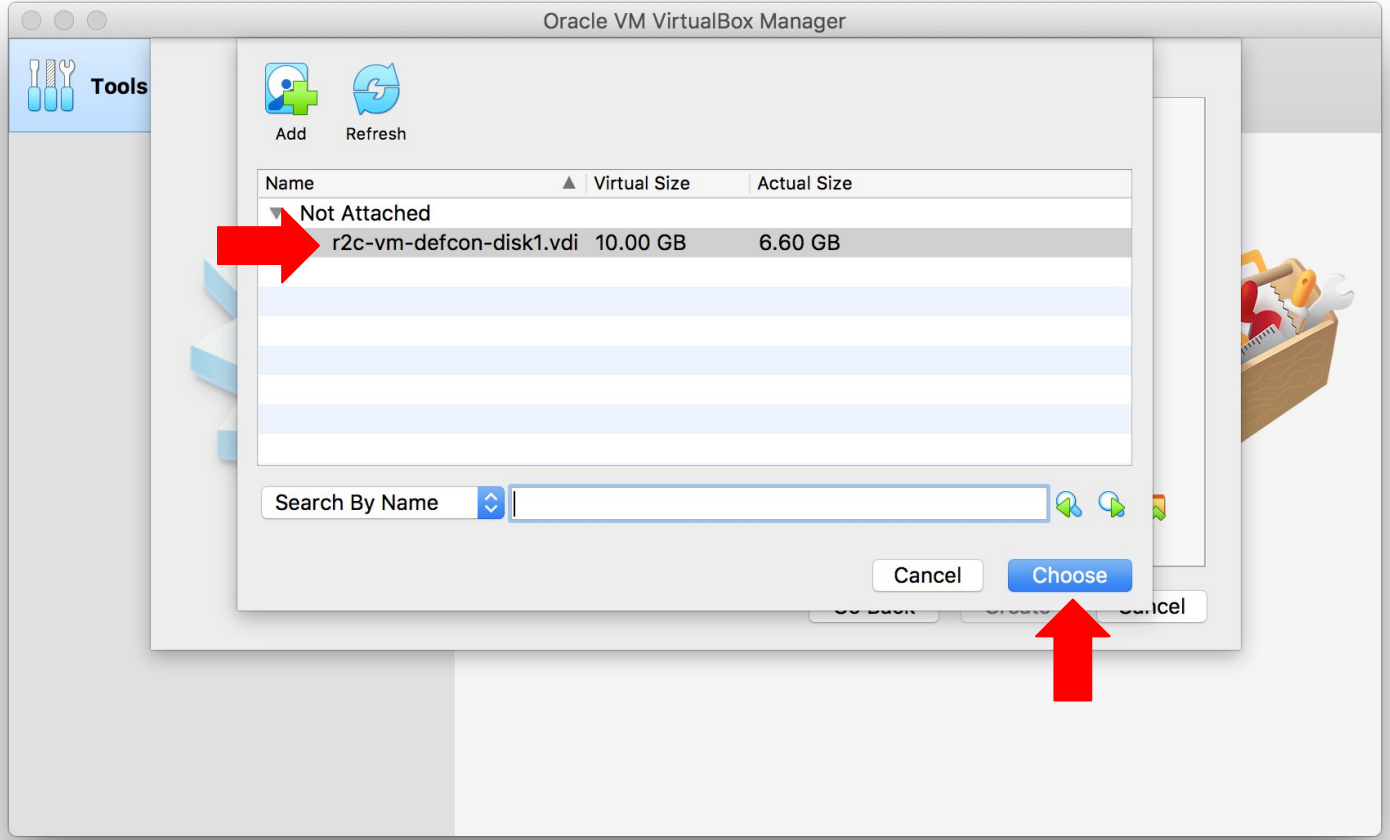
(lost? behind? r2c.dev/defcon or ask a co-instructor)



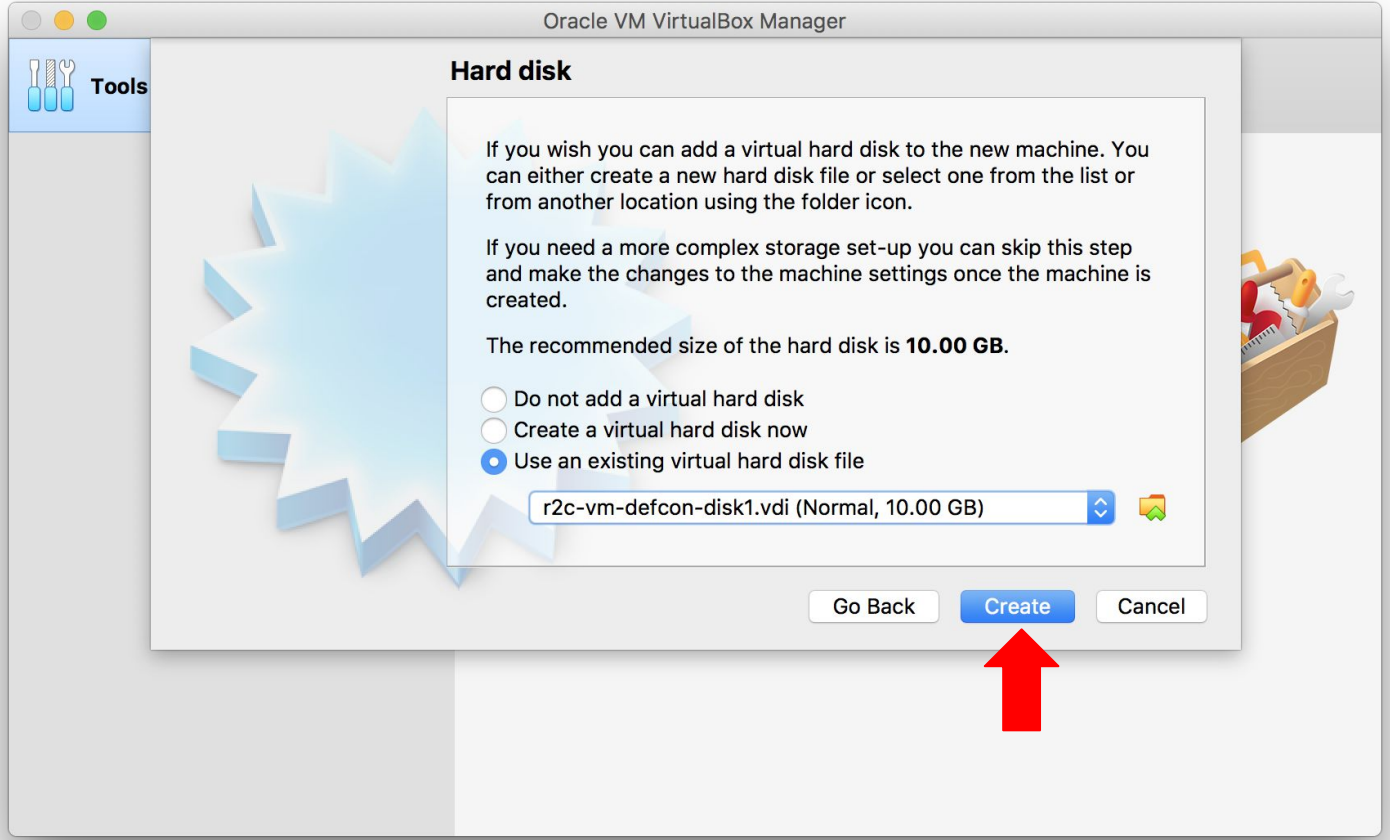
(lost? behind? r2c.dev/defcon or ask a co-instructor)



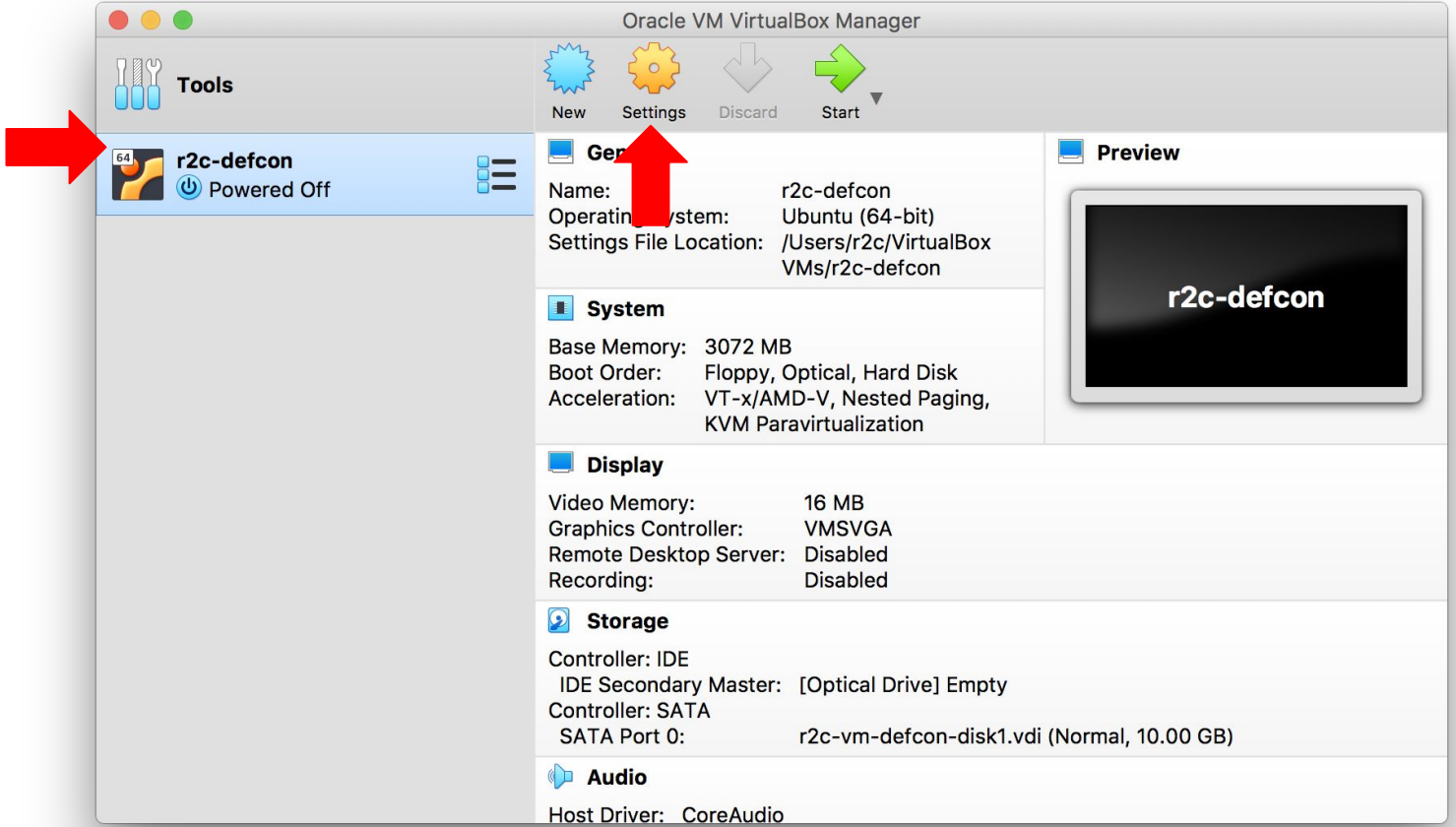
(lost? behind? r2c.dev/defcon or ask a co-instructor)



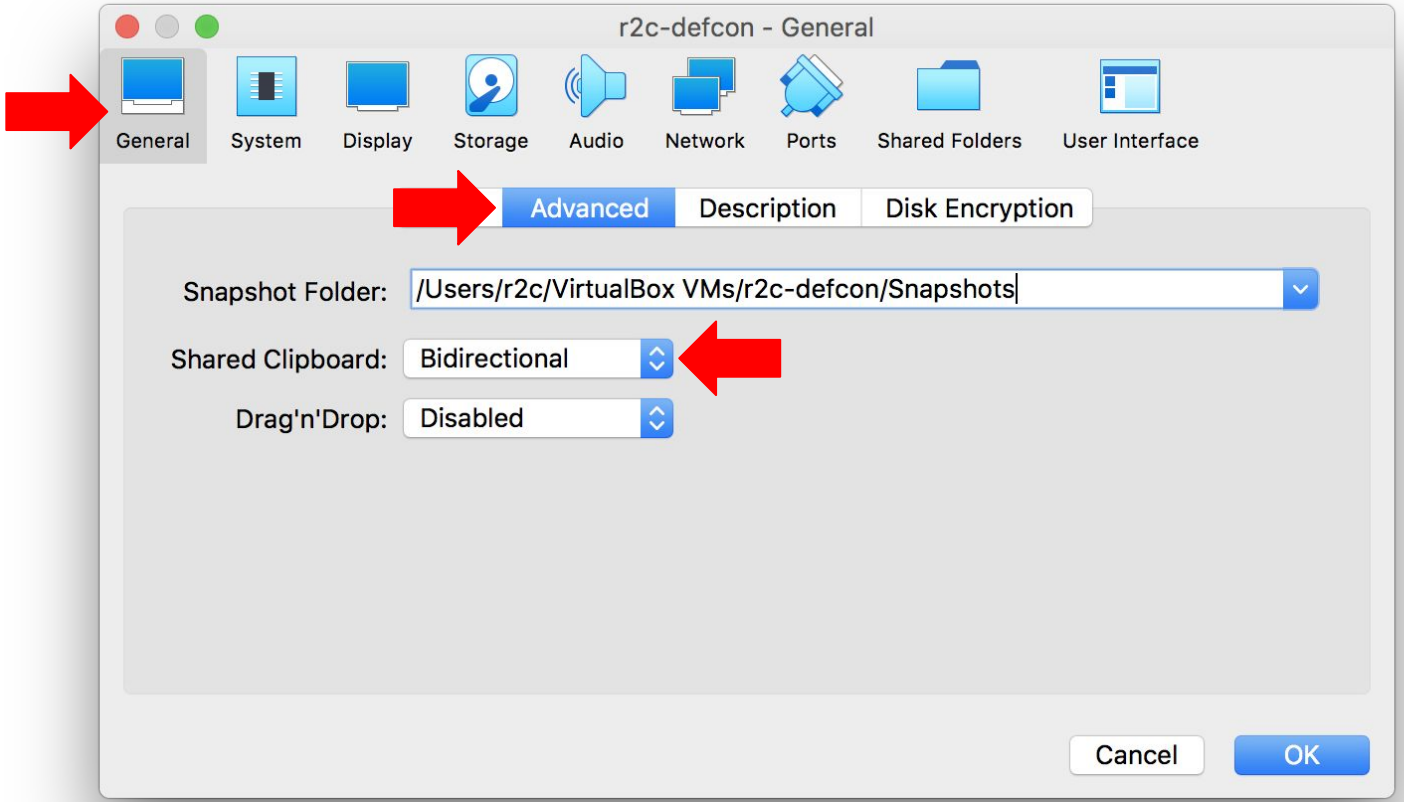
(lost? behind? r2c.dev/defcon or ask a co-instructor)



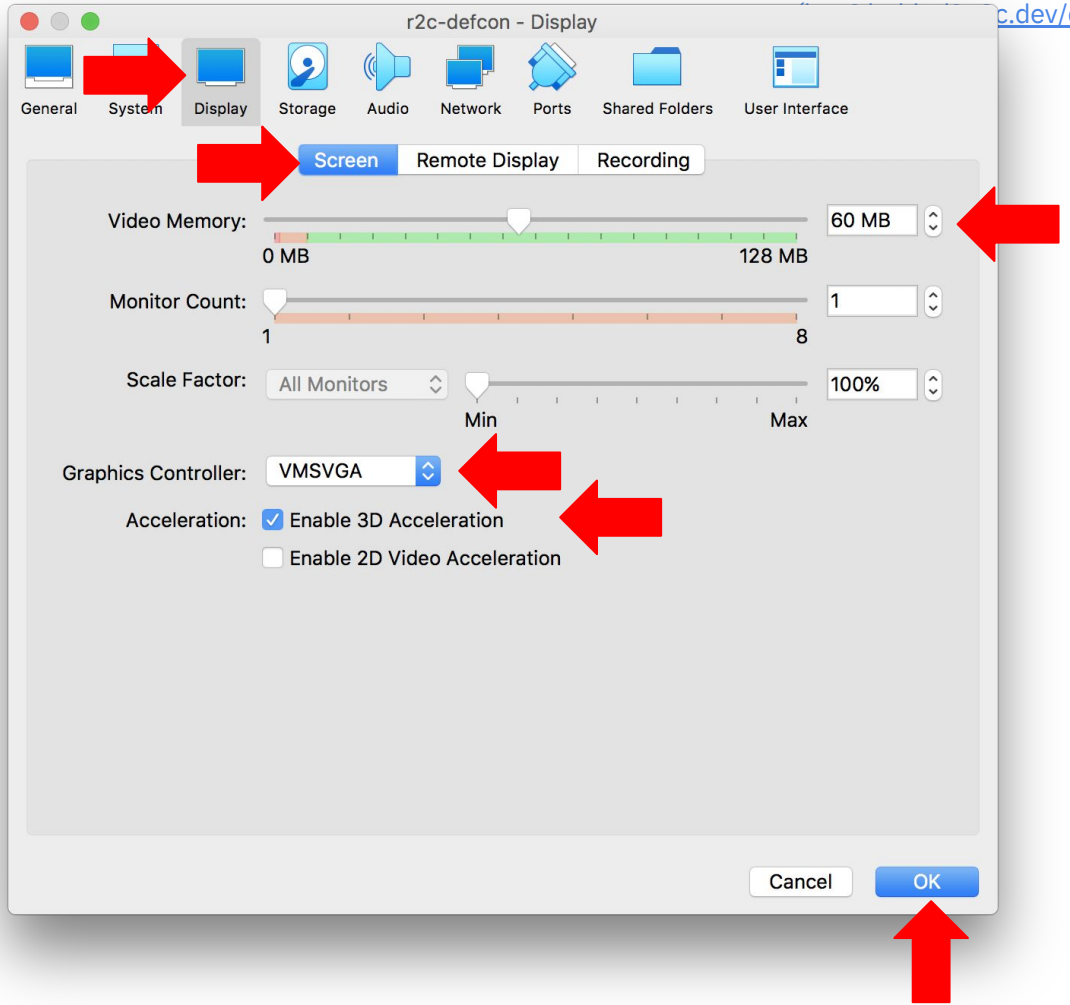
(lost? behind? r2c.dev/defcon or ask a co-instructor)



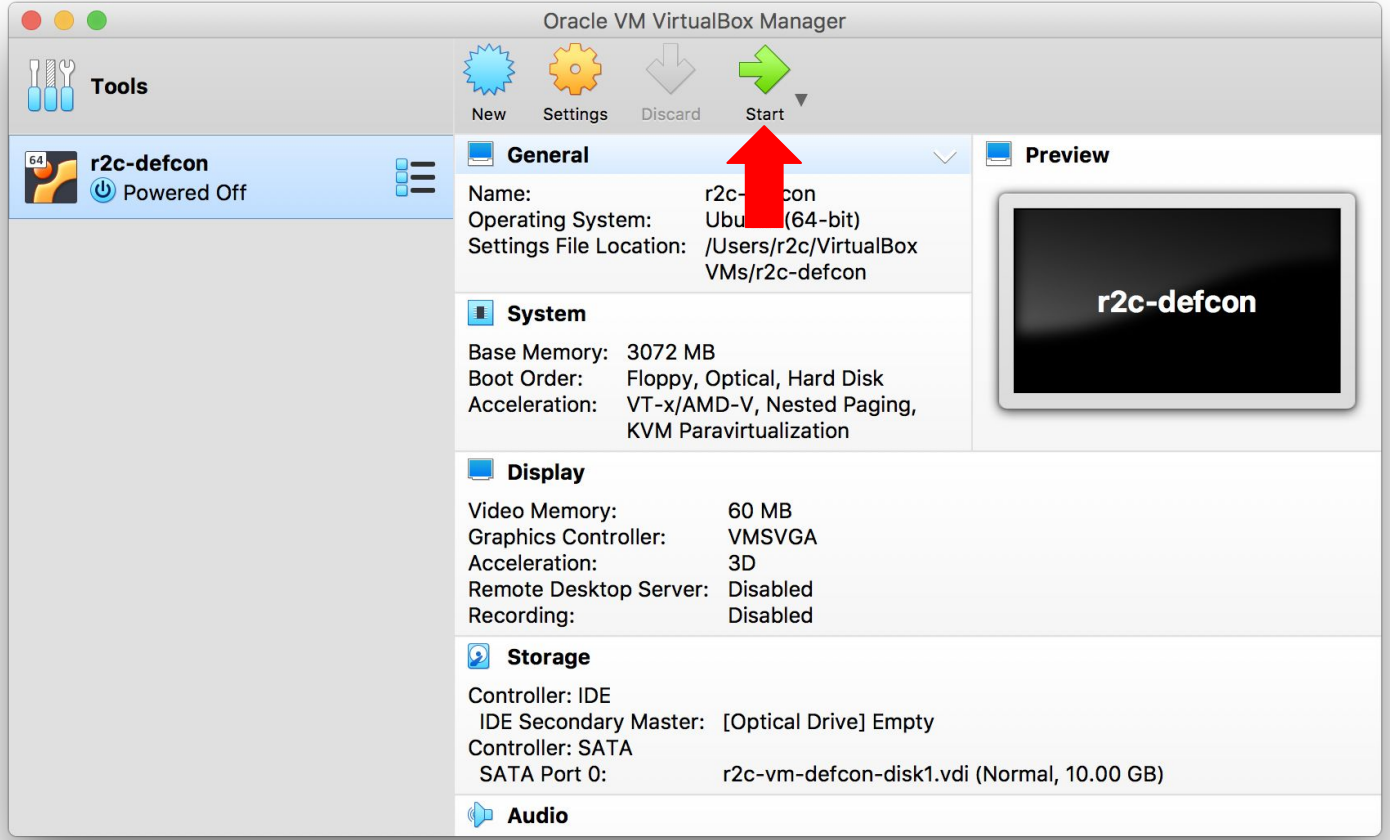
(lost? behind? r2c.dev/defcon or ask a co-instructor)



c.dev/defcon or ask a co-instructor)



(lost? behind? r2c.dev/defcon or ask a co-instructor)



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Install the r2c cli!

```
pip3 install r2c-cli
```

```
r2c
```

```
(cli-dev) bash-3.2$ r2c
Usage: r2c [OPTIONS] COMMAND [ARGS]...

Options:
  --debug / --no-debug
  --help                  Show this message and exit.

Commands:
  init      Creates an example analyzer for your choice of languages.
  login     Log into the R2C analysis platform.
  logout    Log out of the R2C analysis platform.
  push      Push the analyzer in the current directory to the R2C analysis...
  run       Run the analyzer in the current directory over a code directory.
  test      Locally run integration tests for the current analyzer...
  unittest  Locally unit tests for the current analyzer directory You can...
```

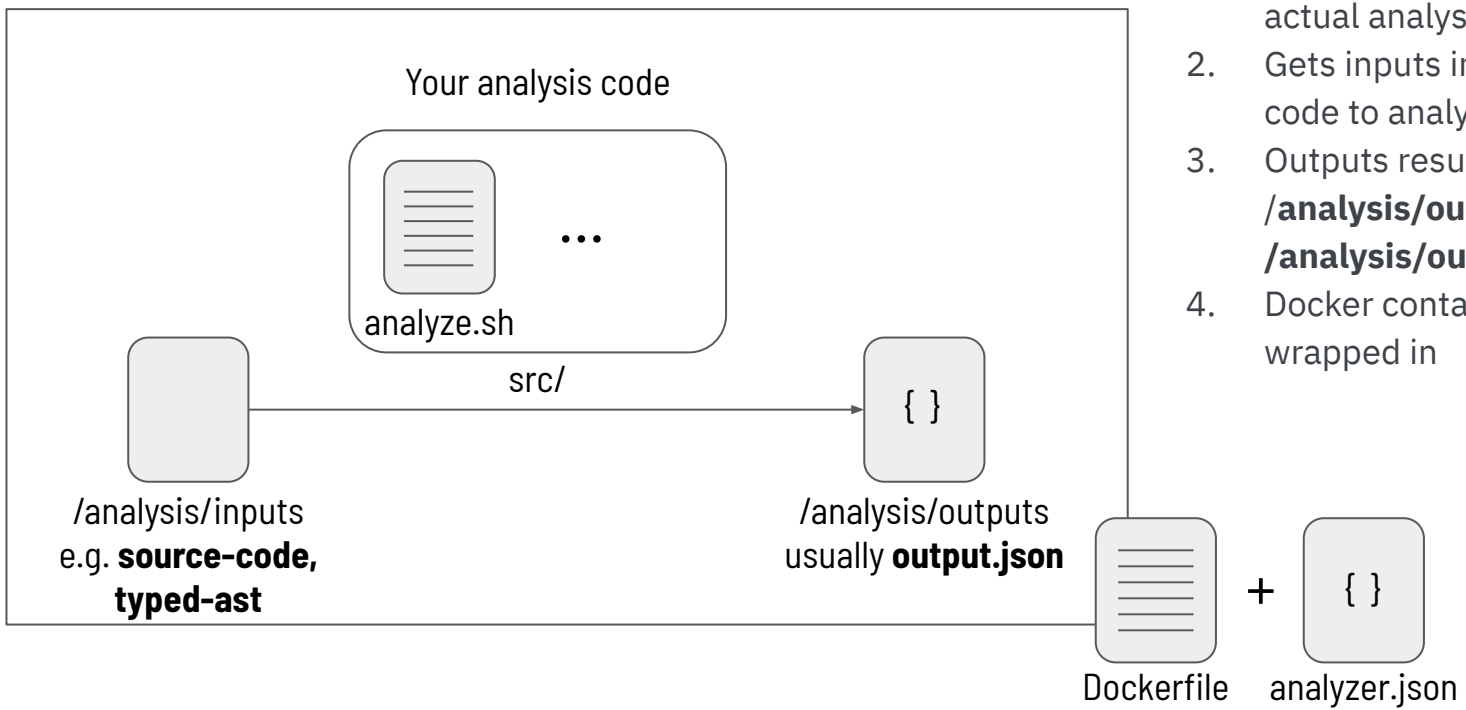
(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 1: **Badwords**

Let's find people who are writing code and angry about it...

```
git clone https://github.com/returntocorp/badwords
```

What is an analyzer?



1. An `analyze.sh` script that kicks off the actual analysis
2. Gets inputs in **/analysis/inputs** (e.g. code to analyze)
3. Outputs results to **/analysis/outputs/output.json** or **/analysis/outputs/fs**
4. Docker container that analysis is wrapped in

(lost? behind? r2c.dev/defcon or ask a co-instructor)

What happens under the hood?

- Dockerfile is used to build a docker image
- Docker image is run for each input (e.g., GitHub repo)
- First, all inputs of an analyzer are fetched and mounted onto the container in an `/analysis/inputs/` directory
- Container is run by running its `analyze.sh` file
- Output is read out of an `/analysis/outputs/` directory and saved

Understanding analyzer files, cont'd:

`analyzer.json`:

`analyzer_name`: A namespaced analyzer name. For this workshop, all analyzers should be namespaced with "defcon" automatically.

`version`: The version of the analyzer. We follow semantic versioning.

`dependencies`: This analyzer depends on the most basic component, public/source-code. It specifies that it depends on any version of it by using "*" as the version. For more complicated analysis, we will depend on components such as r2c/transpiler or r2c/typed-ast later.

Understanding analyzer files (cont'd):

./Dockerfile:

- Responsible for the container's setup and configuration.
- Docker lets us use any language because each image can have its own set of software installed
- Installs software to build and run your analysis
- Specifies `analyze.sh` as an entrypoint for when the Docker container is run

To learn more about Dockerfiles in general, see Docker's tutorial.

Understanding analyzer files (cont'd):

./src:

- `src/analyze.sh` is the main entry point. From this file, we'll run your program that performs analysis!
- `src/unittest.sh` run your analyzer's unit tests, if it has them, inside the container by calling them from this file.
- Everything else in `src/` is your code, and you want to run it from `analyze.sh`

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 1: **Badwords**

- Let’s find people who are writing code and angry about it...
- We’ve written this analyzer in Python
- We’re using the bad words from Google bad words list!

Analyzer 1: **Badwords**

What is a badword?

A trimmed Google bad words list we're using is under `src/bad-words`

Context? Only if used as a *word*[1]

If (ab**butt**ed) <- not bad
x.**ass**igned' <- not bad
["fuck"] <- bad

1: word definition from grep's --word-regexp flag: as if surrounded by `[[:<:]]` and `[[:>:]]` or newer syntax: `\b(?=\w)` and `\b(?<=\w)`

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Understanding analyzer files

```
.
├── Dockerfile
├── README.md
├── analyzer.json
└── src
    ├── analyze.sh
    └── unittest.sh
```

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 1: **Badwords**

First, the Dockerfile, responsible for the container's setup and configuration.

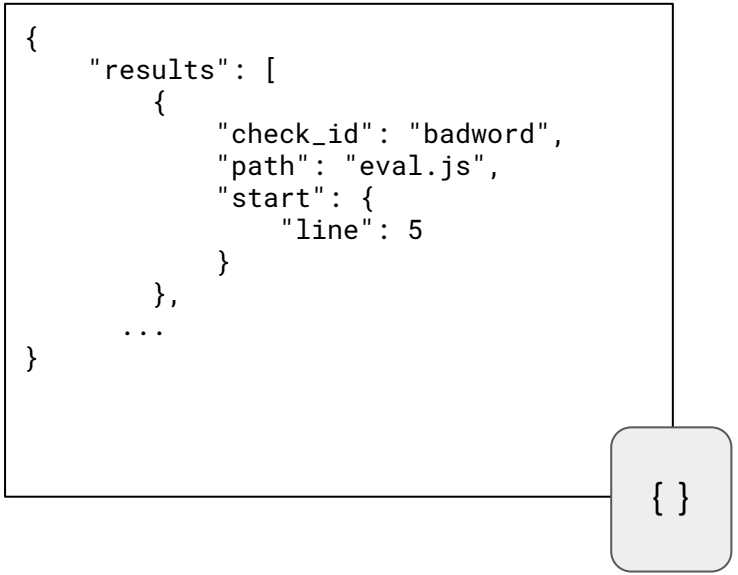
Note that we're installing Python:

```
FROM python:3.7.2-slim-stretch

# TODO: Analyzer specific packages below

# Setup analysis user for docker
RUN groupadd -r analysis && useradd -m --no-log-init --gid analysis analysis
USER analysis
COPY src /analyzer


# Setup entrypoint into the analysis code logic
WORKDIR /
CMD ["/analyzer/analyze.sh"]
```

r2c.dev/defcon-links (last3 behind? r2c.dev/defcon or ask a co-instructor) docs.r2c.dev

Analyzer 1: **Badwords**

- we expect input in `/analysis/inputs/public/source-code/`
- because we depend on the `source-code` analyzer which clones repos for us
- we write to `/analysis/outputs/output.json`
- we're trying to produce results and save them in the following schema →



```
{
    "results": [
        {
            "check_id": "badword",
            "path": "eval.js",
            "start": {
                "line": 5
            }
        },
      ...
}
```

{ }

Analyzer 1: **Badwords**

`analyze.sh` is the entrypoint specified our Dockerfile. Let's see what it does:

1. First, we navigate to where the code we're trying to analyze is:
2. Our analyzer's code lives at `/analyzer/badwords.py` because in our Dockerfile, we copied the `src` folder to the `/analyzer` folder inside the container.
3. Recursively (-r) look for words (-w), ignore case (-i), and print the filename (-H) and line number (-n). Include (--include) JS/TS/Python files.

```
#!/bin/bash

set -e
CODE_DIR="/analysis/inputs/public/source-code"

grep -w -Hnir --include \*.ts --include \*.tsx --include \*.js --include \*.jsx --include \*.py -f /analyzer/bad-words "${CODE_DIR}" |
    python3 /analyzer/badwords.py "${CODE_DIR}" /analysis/output/output.json

exit 0
```

Analyzer 1: **Badwords**

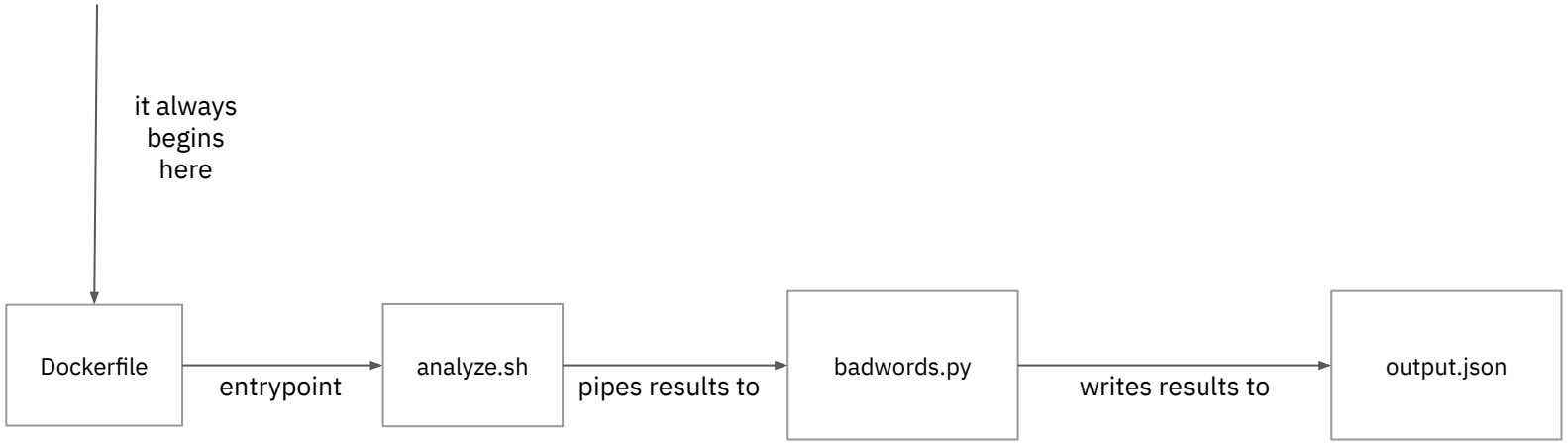
`badwords.py` actually saves the results with the right format.

```
{
    "results": [
        {
            "check_id": "badword",
            "path":
"test/backbone-koans/js/koans/aboutModels.js",
            "start": {
                "line": 22
            }
        },
        {
            "check_id": "badword",
            "path": "test/backbone-koans/js/todos.js",
            "start": {
                "line": 5
            }
```

```python
#!/usr/bin/env python3
import json
import os
import sys

def make_results(grep_lines, base_path):
    output = []
    for line in grep_lines:
        if not (len(line.strip())):
            continue
        print(line)

        filename = line.split(":")[0]
        filename = os.path.relpath(filename, base_path)
        line_num = int(line.split(":")[1])
        if filename.startswith("./"):
            filename = filename[2:]

        obj = {"check_id": "badword", "path": filename, "start": {"line": line_num}}
        output.append(obj)
    return output


def format_json(grep_lines, stream, base):
    results = make_results(grep_lines, base)
    obj = {"results": results}
    json.dump(obj, stream, indent=4, separators=(",", ": "))


if __name__ == "__main__":
    print("starting now")
    base = sys.argv[1]
    fout_name = sys.argv[2]
    grep_output = sys.stdin.readlines()
    with open(fout_name, "w") as fout:
        format_json(grep_output, fout, base)
```

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 1: **Badwords**

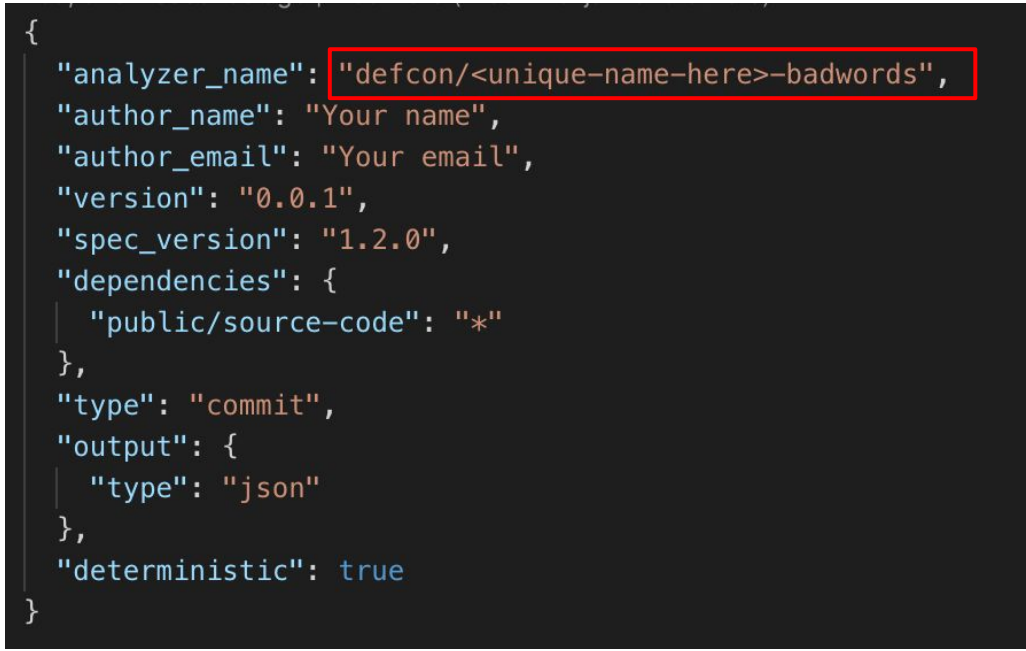


Analyzer 1: **Badwords**

`analyzer.json` defines the r2c-specific manifest.

Edit `analyzer.json`, change analyzer_name to:

`defcon/<your-username>-badwords`



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 1: **Badwords**

Let's run this on an actual repo:

```
git clone https://github.com/returntocorp/blanket /tmp/analysis/blanket
```

Go back into the badwords analyzer directory and run:

```
r2c login --org defcon
r2c run /tmp/analysis/blanket
```

Let us know if your login failed! Submit your username at r2c.dev/defcon-signup

Analyzer 1: **Badwords**

In `src/badwords.py`**, the meat of the analyzer:** read file at the given path, produce results, write them to `output.json`

There should be 6 entries in the output!


```
{
    "results": [
        {
            "check_id": "badword",
            "path": "test/backbone-koans/js/koans/aboutModels
            "start": {
                "line": 22
            }
        },
        {
            "check_id": "badword",
            "path": "test/backbone-koans/js/todos.js",
            "start": {
                "line": 5
            }
        },
```

Analyzer 1: **Badwords**

In `src/badwords.py`**, the meat of the analyzer:** read file at the given path, produce results, write them to `output.json`

```
{
    "results": [
        {
            "check_id": "badword",
            "path": "test/backbone-koans/js/koans/aboutModels.js",
            "extra": { "flagged-word": "fuck" },
            "start": {
                "line": 22
            }
        },
   ...
}
```

Analyzer 1: **Badwords**

`analyzer.json` defines the r2c-specific manifest.

1. Edit `analyzer.json`, change analyzer_name to:

`defcon/<your-username>-badwords`

2. Now run:

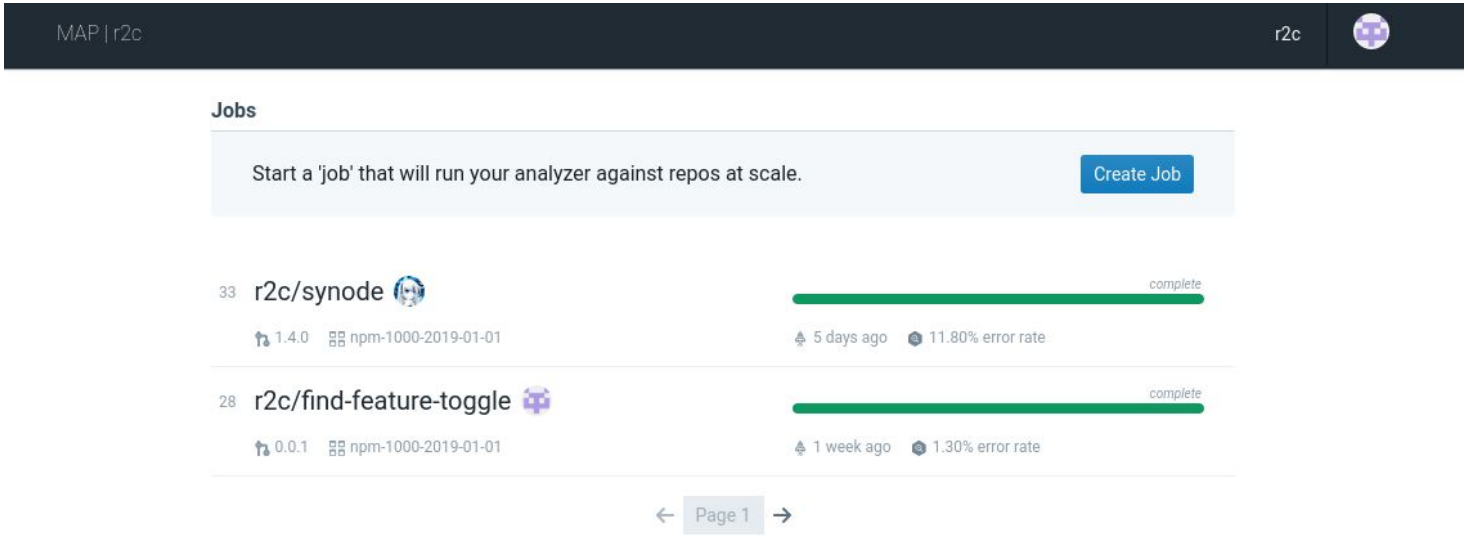
`r2c push`

```
{
  "analyzer_name": "defcon/<unique-name-here>-badwords",
  "author_name": "Your name",
  "author_email": "Your email",
  "version": "0.0.1",
  "spec_version": "1.2.0",
  "dependencies": {
    "public/source-code": "*"
  },
  "type": "commit",
  "output": {
    "type": "json"
  },
  "deterministic": true
}
```

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 1: **Badwords**

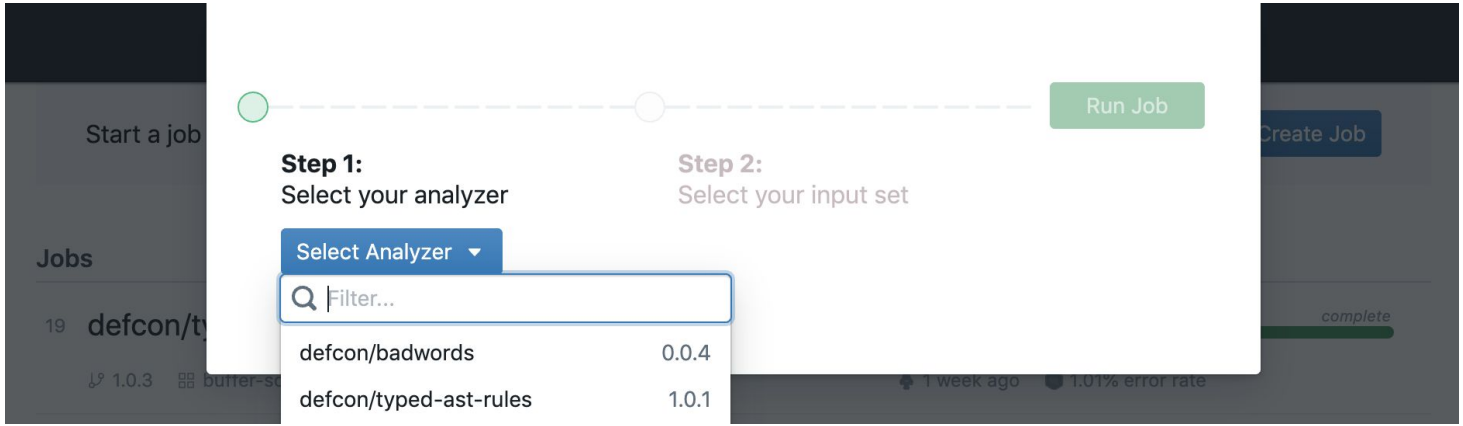
Let's go over to the web UI at https://defcon.r2c.dev and run our analyzer on lots of projects:



(lost? behind? r2c.dev/defcon or ask a co-instructor)

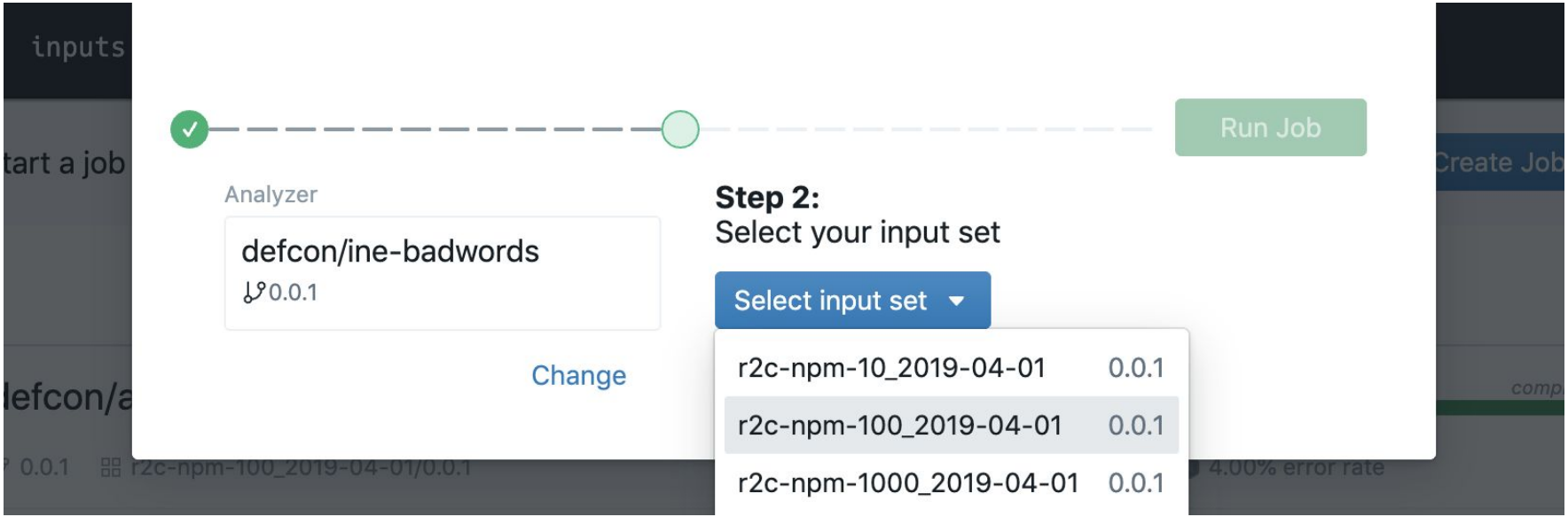
Analyzer 1: **Badwords**

Let’s select our analyzer from the list of all analyzers on the platform:



Analyzer 1: **Badwords**

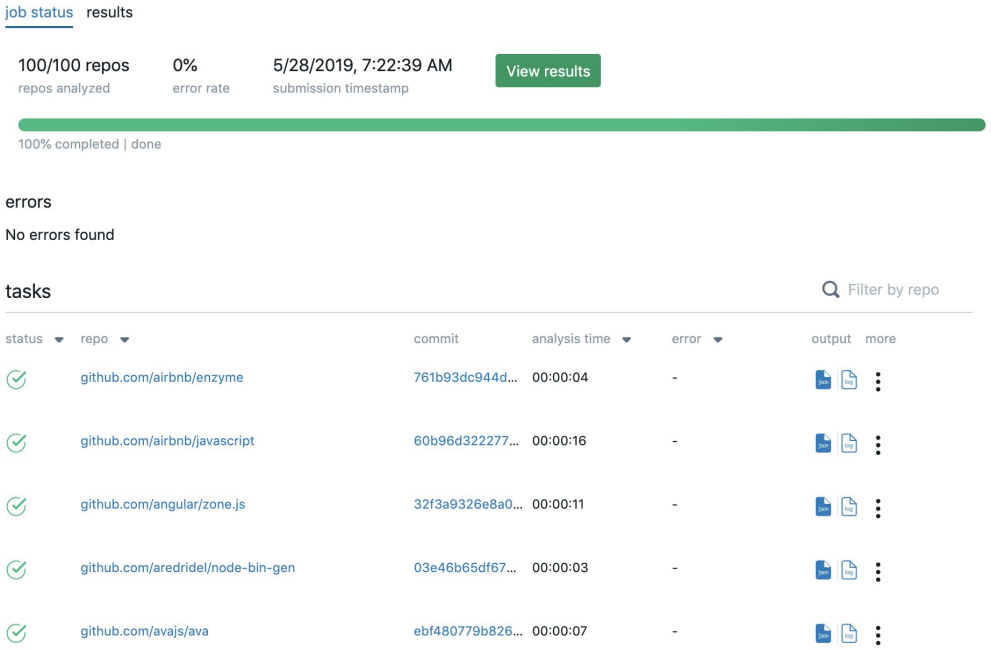
And select the set of projects we want to run it on! Let's choose the top 100 projects on NPM.



(lost? behind? r2c.dev/defcon or ask a co-instructor)

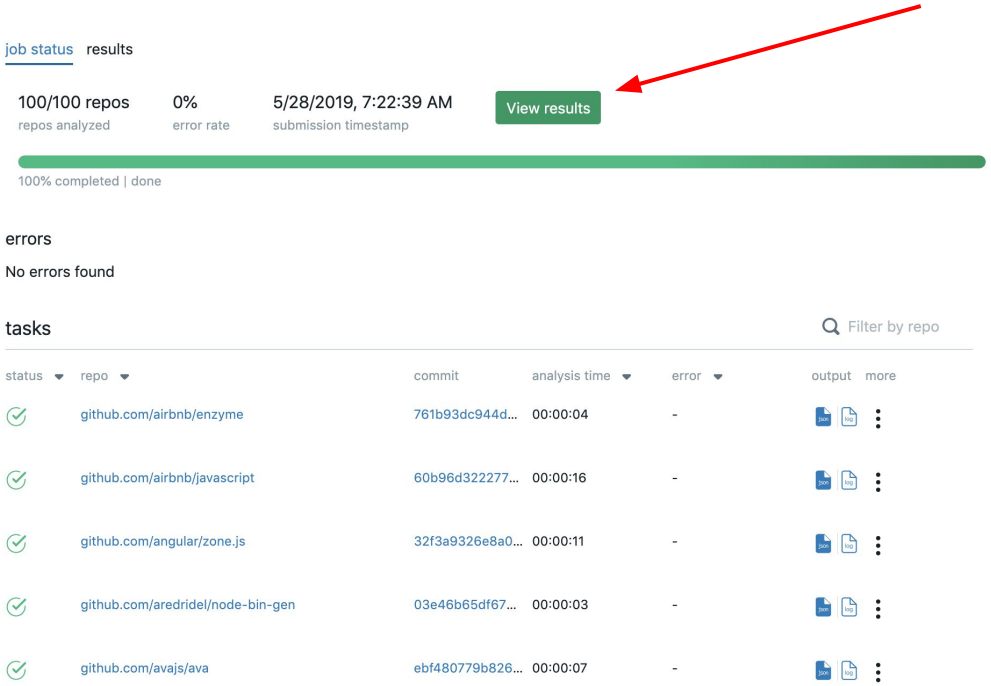
Analyzer 1: **Badwords**

You can download the results and logs of each run:



Analyzer 1: **Badwords**

You can download the results and logs of each run:



(lost? behind? r2c.dev/defcon or ask a co-instructor)

15min break for lunch

ret-2-workshop @ 12

while (is_hungry) { is_hungry = eat(); }

(lost? behind? r2c.dev/defcon or ask a co-instructor)

part 4/5

I’m a real analyzer now!

(At least, for JavaScript)

Analyzer 2: **Javascript is a terrible language**

Vulnerability in the Buffer API

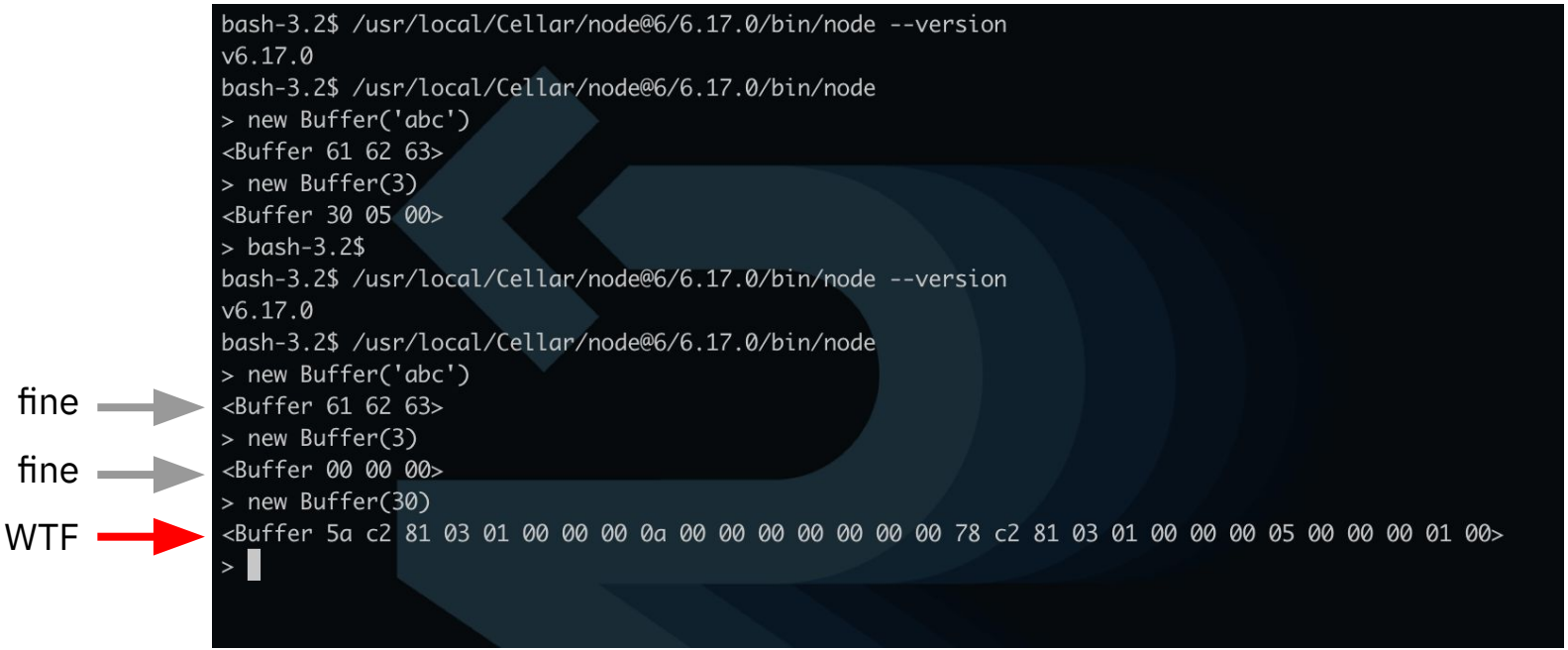
- New Buffer API in NodeJS
- Node is common in server-side/desktop apps
- Buffers in NodeJS can be constructed in the following ways:

```
const buf1 = new Buffer([1,2,3]);
// creates a buffer containing [01, 02, 03]
const buf2 = new Buffer('test');
// creates a buffer containing ASCII bytes [74, 65, 73, 74]
const buf3 = new Buffer(10);
// creates a buffer of length 10
```

On versions of Node prior to Node 6: when passed a number, the buffer constructor allocates memory without initializing (zeroing) it out!

Analyzer 2: **Javascript is a terrible language**

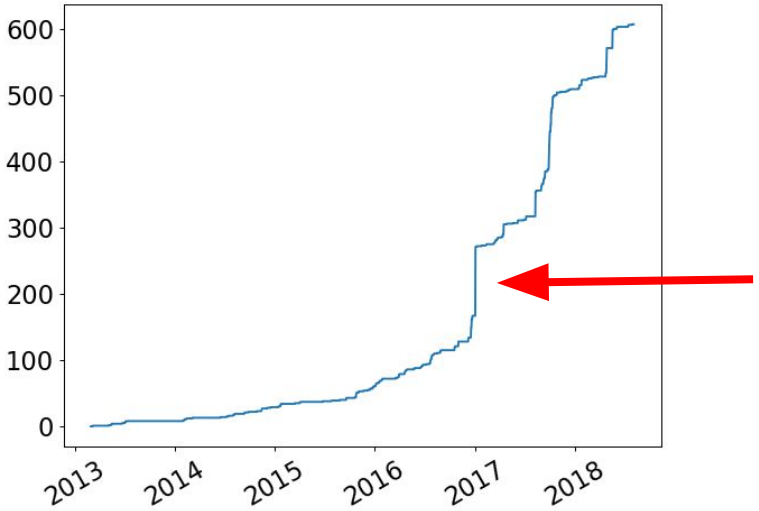
Vulnerability in the new Buffer API



Analyzer 2: **Javascript is a terrible language**

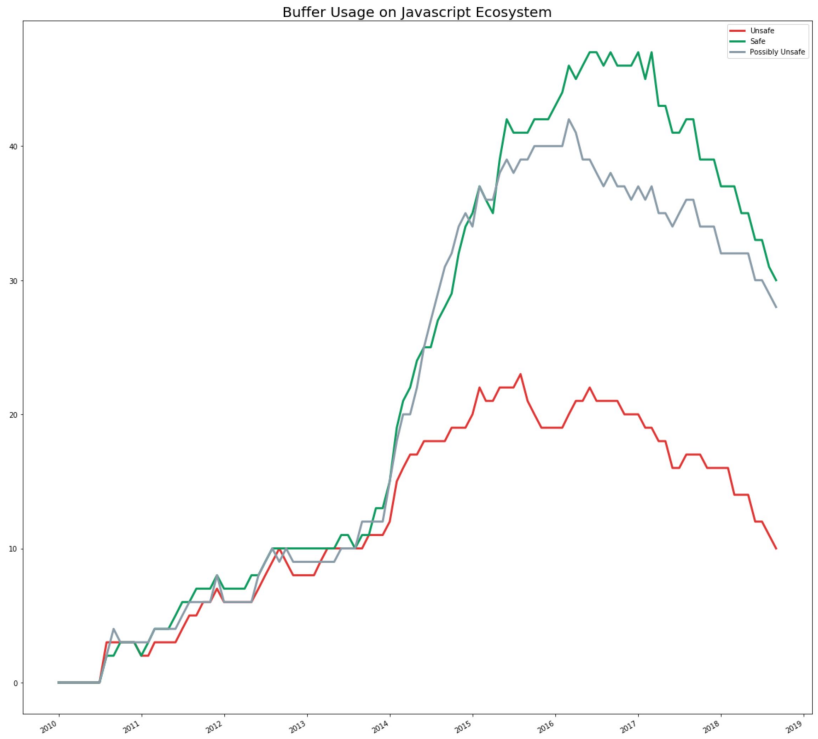
Vulnerability in the new Buffer API

- Discovered almost 3 years ago! Several instances reported
- Led to the deprecation of the API
- Vulns in mongoose, ws, request and most recently sequelize
- Remember that graph of vulnerabilities over time?



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 2: **Javascript is a terrible language**



Analyzer 2: **Javascript is a terrible language**

Vulnerability in the new Buffer API

```
21  function MongooseBuffer(value, encode, offset) {
22    var length = arguments.length;
23    var val;
24
25    if (length === 0 || arguments[0] === null || arguments[0] === undefined) {
26      val = 0;
27    } else {
28      val = value;
29    }
30
31    var encoding;
32    var path;
33    var doc;
34
35    if (Array.isArray(encode)) {
36      // internal casting
37      path = encode[0];
38      doc = encode[1];
39    } else {
40      encoding = encode;
41    }
42
43    var buf = new Buffer(val, encoding, offset);
44    utils.decorate(buf, MongooseBuffer.mixin);
45    buf.isMongooseBuffer = true;
```



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 2: **Javascript is a terrible language**

The analyzer will depend on *r2c/typed-ast*

1. Check out the example analyzer:

   ```
   git clone https://github.com/returntocorp/buffer-template
   ```

2. Edit the `analyzer_name` field in `analyzer.json`:

   ```
   defcon/<your-username>-buffer
   ```

3. Test on this repo:

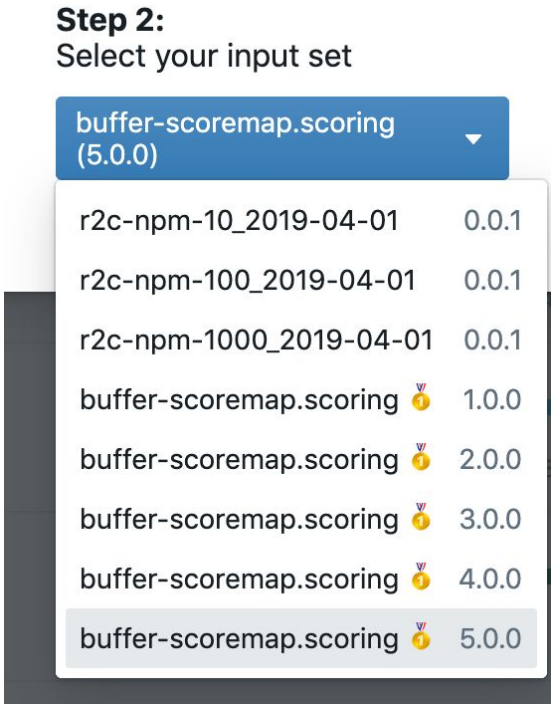
   ```
   git clone https://github.com/returntocorp/buffer-rule-tests /tmp/XYZ
   r2c run /tmp/XYZ
   ```

Analyzer 2: **Getting started**

This analyzer will be an AST visitor that reports every instance of the API -- safe or not (will have a lot of false positives -- we'll fix that later)

Your goal: detect all the positive cases in our test set via an ast-based analysis.

Hint: only fire when the constructor is called with a value that could possibly be a number as its argument. Output `bad_buffer_constructor` as check_id. Make sure to include the `start_line`!



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 2: **Dependencies**

You can explore the output of *r2c/typed-ast* at **r2c-explorer.dev**



(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 2: **Getting started**

To check your logic, enter some new Buffer statements (select the Buffer example) and see what the tree looks like.

Remember to use `bad_buffer_constructor` as check_id

Ask the co-instructors if you need help!

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 2: **Javascript is a terrible language**

Starting again with **https://github.com/returntocorp/buffer-template**

```
.
├── CHANGELOG.md
├── Dockerfile
├── README.md
├── analyzer.json
└── src
    ├── analyze.sh
    ├── package-lock.json
    ├── package.json
    ├── src
    │   ├── index.ts
    │   └── rules
    │       └── buffer.ts          ← example usage: start here
    ├── tsconfig.json
    └── unittest.sh
```

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 3: **types → dataflow**

Let's use some of the program analysis tooling we talked about earlier to eliminate the false positives!

What we need is a dataflow analysis that can answer the question of whether a specified identifier refers to a *number* or a *string*

```
new Buffer(x: number) // bad
```

```
new Buffer(x: string) // ok
```

```
new Buffer(x: not a number or a string) // ??
```

Analyzer 3: **what about those false positives?**

```
function alloc(safe_var) {
  if (typeof safe_var === 'string' || safe_var instanceof
String) {
     return new Buffer(safe_var)
  }
}

function broken_alloc(unsafe_var) {
  x = new Buffer(unsafe_var)
  return 4;
}

function unsafe_but_filled(unsafe_var) {
  x = new Buffer(unsafe_var)
```

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzer 3: **AST visiting**

```
function isRequire(ty) {
  return ty.isNominative() && ty.name == "NodeRequire";
}

return {
  create(context) {
    return {
      CallExpression(node) {
        if (isRequire(getType(node.callee))) {
          const ty = getType(node.arguments[0]);
          if (!ty.mustSatisfy(t => t.isLiteral())) {
            context.report({
              node,
              checkId: "non-literal require",
              extra: node.value
            });
          }
        }
      }
    }
  }
}
```

typed-ast: **JS AST with type annotations**

- JSON: estree + extra inferredType property
- Consume in any language!
- Inference: TypeScript + @types packages
  - Works even with invalid JS, with non-typed JS, etc.
- 56.4% AST nodes with nontrivial types over top 100K packages on npm
- Separate ‘definitions’ file for classes to avoid repetition

Why types?

- Can’t know if this is safe otherwise!
- string is safe, number isn’t (uninitialized memory)

```
const arg = getSomeArgument();
const buf = new Buffer(arg);
```

stadt: **serialization for TypeScript types**

- github.com/returntocorp/stadt
- MIT-licensed!
- It's just JSON: anyone can use this for type serialization/deserialization


```
{
  "returnType": {
    "kind": "nominative",
    "name": "Promise",
    "fullyQualifiedName": {
      "builtin": true,
      "name": "Promise"
    },
    "typeArguments": [
      {
        "kind": "nominative",
        "name": "Array",
        "fullyQualifiedName": {
          "builtin": true,
          "name": "Array"
        },
        "typeArguments": [
          {
            "kind": "nominative",
            "name": "Dirent",
            "fullyQualifiedName": {
              "builtin": false,
              "fileName": "fs.d.ts",
              "packageName": "@types/node",
              "name": "Dirent"
            },
            "typeArguments": []
          }
        ]
      }
    ]
  }
}
```

(lost? behind? r2c.dev/defcon or ask a co-instructor)

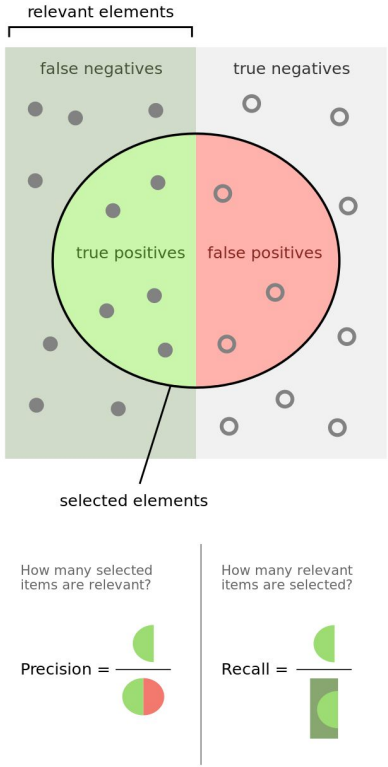
For Analyzers 3: we have test cases!

Maximize **True Positives** and **True Negatives**

Minimize **False Positives** and **False Negatives**

Our online scoring is based on the F-1 score:
$F_1 = 2 \times (\text{precision} \times \text{recall}) / (\text{precision} + \text{recall})$

(en.wikipedia.org/wiki/F1_score)



(lost? behind? r2c.dev/defcon or ask a co-instructor)

part 5/5

is your analyzer...web-scale?

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Analyzers can be collaborated on!

Both by sharing the source, and by depending on them in the r2c registry

Collaborate work with your workshop partners & please share your results

(chat with us if you're working on something more heavyweight, we'd love to help!)

(lost? behind? r2c.dev/defcon or ask a co-instructor)

What are the available analyzers?

(lost? behind? r2c.dev/defcon or ask a co-instructor)

What are the available analyzers?

github/

semantic

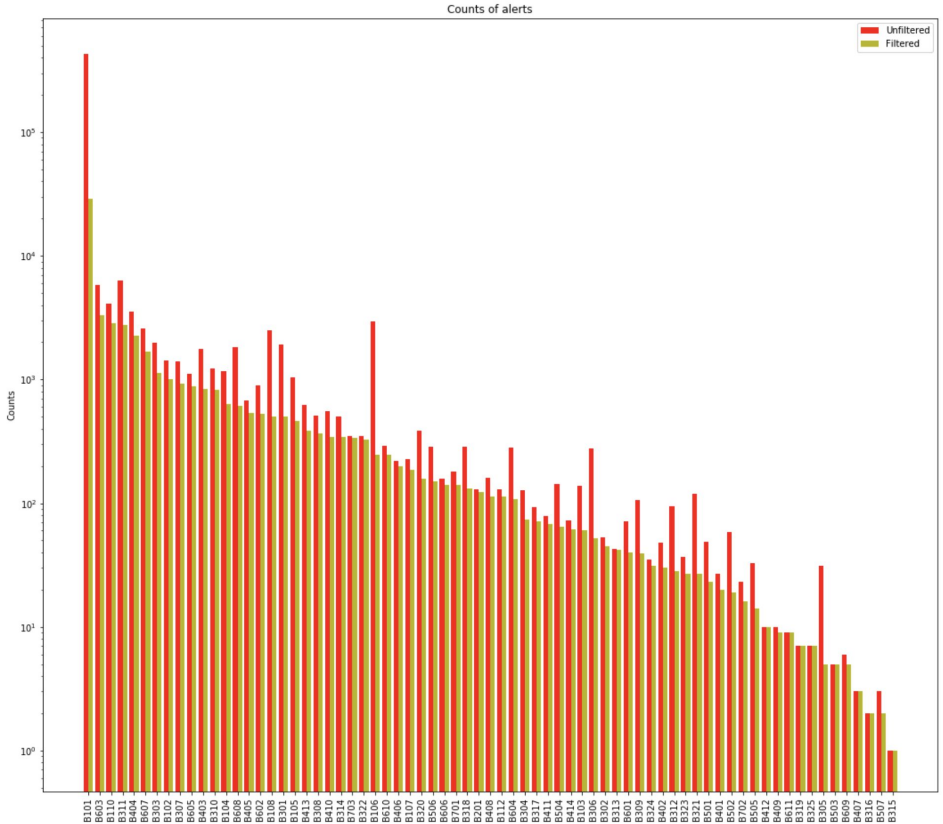
Language support

Priority	Language	Parse	Assign	Diff	ToC	Symbols	Import graph	Call graph	Control flow graph
1	Ruby	✅	✅	✅	✅	✅	✅	🚧	
2	JavaScript	✅	✅	✅	✅	✅	✅	🚧	
3	TypeScript	✅	✅	✅	✅	✅	✅	🚧	
4	Python	✅	✅	✅	✅	✅	✅	🚧	
5	Go	✅	✅	✅	✅	✅	✅	🚧	
	PHP	✅	✅	✅	✅	✅			
	Java	✅	✅	✅	🔶	✅			
	JSON	✅	✅	✅	N/A	N/A	N/A	N/A	
	JSX	✅	✅	✅	🔶				
	Haskell	✅	✅	✅	🔶	✅			
	Markdown	✅	✅	✅	🔶	N/A	N/A	N/A	

(lost? behind? r2c.dev/defcon or ask a co-instructor)

What are the available analyzers?

r2c/bandit on PyPi



(lost? behind? r2c.dev/defcon or ask a co-instructor)

What are the available analyzers?

r2c/js-permissions

(lost? behind? r2c.dev/defcon or ask a co-instructor)

What are the available analyzers?

r2c/js-test-finder*

*worked on by several groups; currently invitation-only

(lost? behind? r2c.dev/defcon or ask a co-instructor)

What are the available analyzers?

collab/js-callgraph*

*worked on by several groups; currently invitation-only

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Why all the collaborative + large scale features?

1) Build an “app store” for program analysis
2) Use it to find large numbers of vulnerabilities (where others might not be looking!)
3) Profit! Report the vulns!
4) Give feedback to the authors of the analyzers

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Hunting time!

Outputs from some of the analyzers just shown are cached on 100K-1M repos/packages on r2c infrastructure:
defcon.r2c.dev/triager

Specifically, these results might be more interesting:
https://r2c.dev/defcon-triage

Only ask: please credit r2c.dev if you file an issue/bug bounty!

Or, write a new analyzer / improve an existing one!



r2c

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Stay in touch!

Feedback	r2c.dev/defcon-feedback <- req. for a shirt!
User Research	r2c.dev/ur
Join our team!	r2c.dev/team
Email	hello@r2c.dev



r2c

(lost? behind? r2c.dev/defcon or ask a co-instructor)

Stay in touch!

Lots of people writing analysis and some of them hanging out on our slack!

Email **hello@r2c.dev** for an invite!