Abuse of trust

Exploiting our relationship with public repositories

Pat H
path/to/file @github
@gmail.com

Dev4Crowd

- Me: Low-level programmer by day
  - (make joke about the “C”)
- You: Interested in the software supply chain
- Focus on Python and Pypi
  - But applicable to many languages

Problem :

The software supply chain

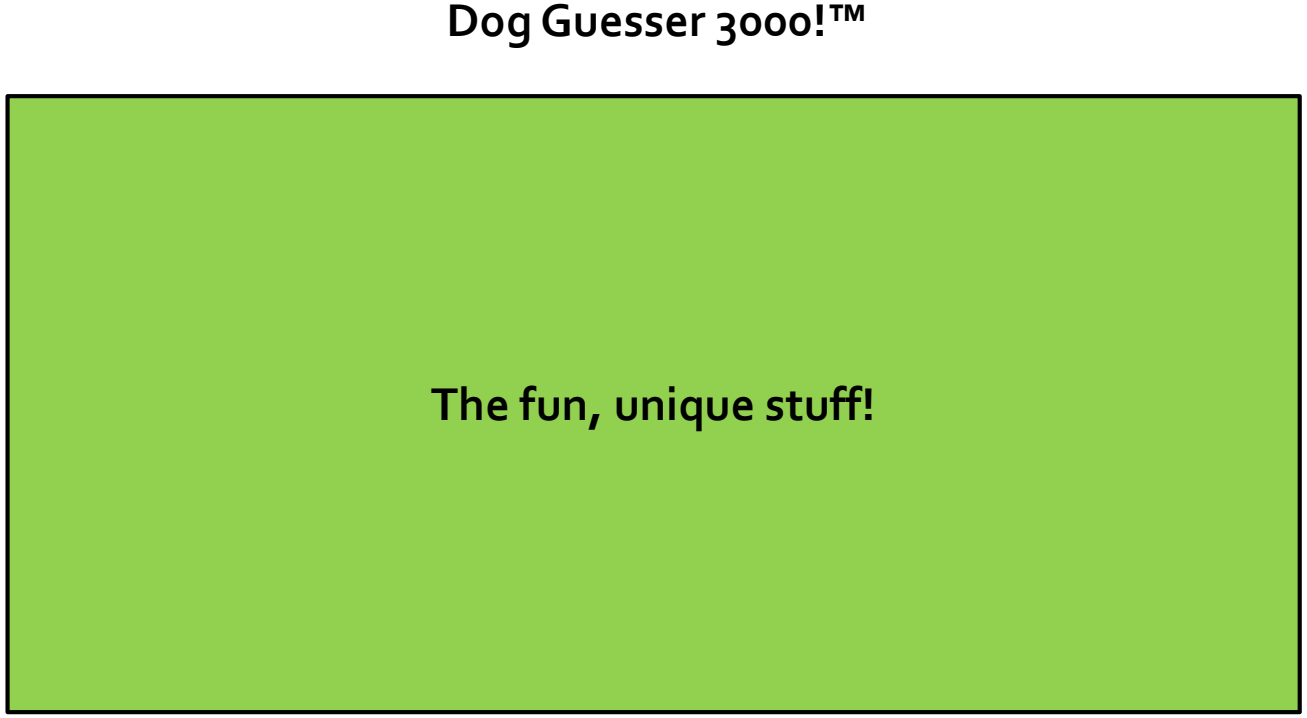
- Modern software development is hard!
- Many moving parts:
  - Multiple Operating Systems / Browsers
  - Plug into 3rd Party authentication
  - Host code on 3rd Party servers
  - . . .

Problem :

The software supply chain

Problem :

The software supply chain



Problem :

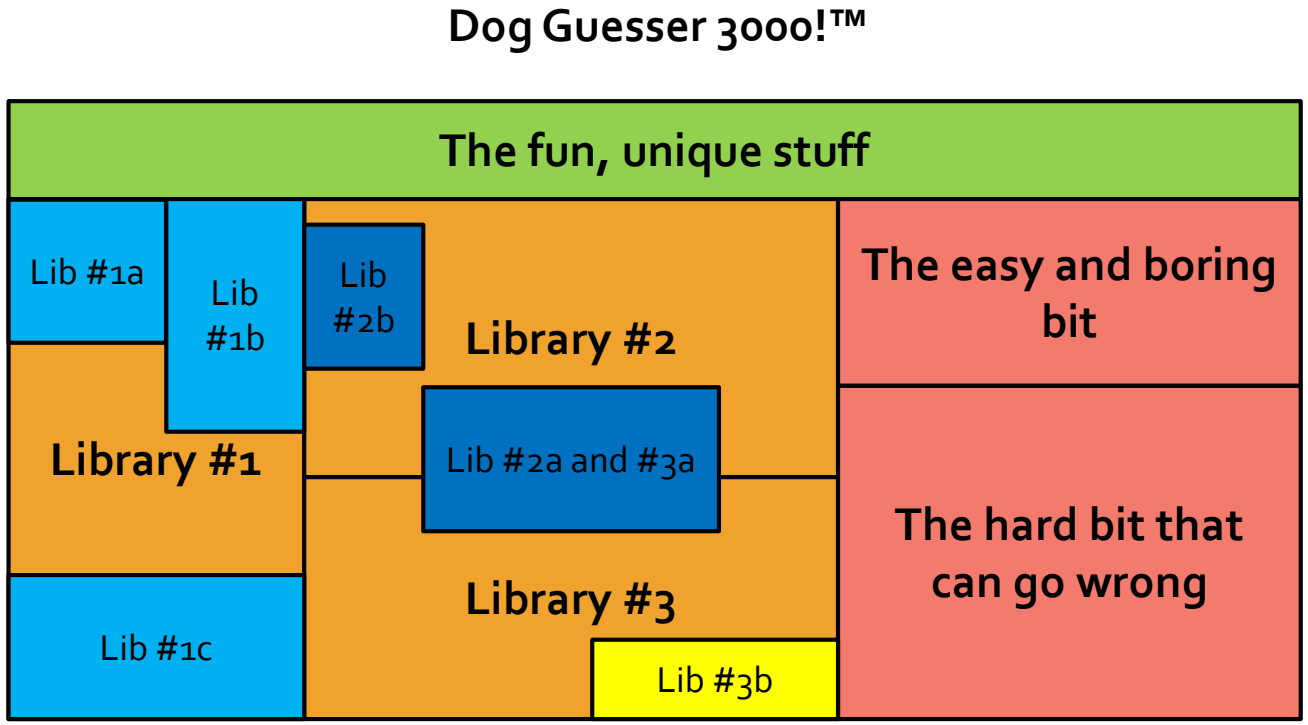
The software supply chain

Problem :

The software supply chain

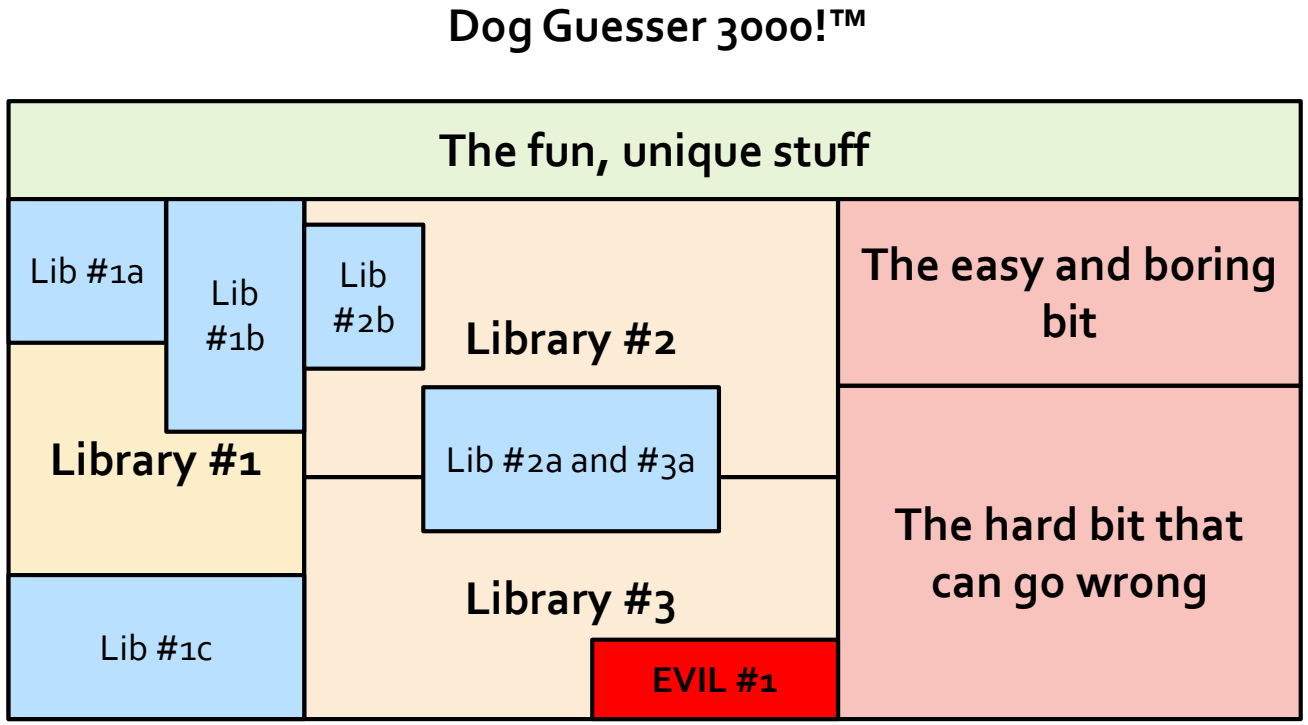
Problem :

The software supply chain



Problem :

The software supply chain



Lose your secrets in 3 steps:

Spear phishing to
Code execution to
Data exfiltration oh my

Villain4Victim

- Walk through a hypothetical scenario
- CoolCorp, uses Python in accounting
  - Use “big-money” app from PyPi
- Attack goal: steal SECRETS.txt

```
x = open('SECRETS.txt').read()
# TODO: exfil
```

Step #1:

Phishing Expedition

- Phish developer of library of a package you use
  - App: “big-money”
  - Depends on: “lib-lotsa-money”
- Public Python projects have a setup.py listing
  - Creator’s name
  - Email addresses
  - Home pages
  - URLs containing vendor names
  - Links to Gitlab instances with keys
  - Links to unsecured wikis with private discussions
- Once credentials are obtained, they are no longer protected by 2FA, notifications, etc

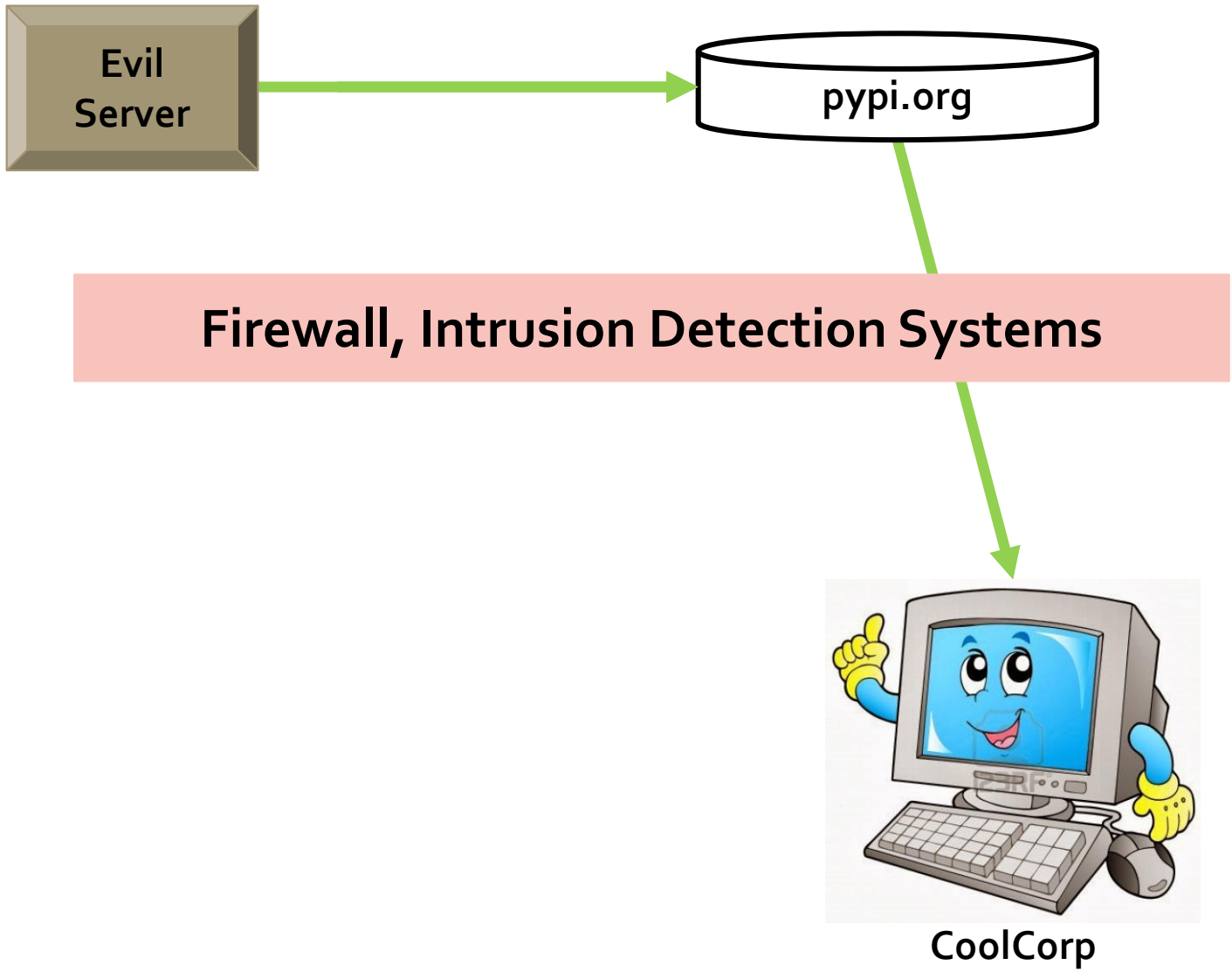
Step #2:

Code execution at install time

- No obvious code changes needed
  - Just a single line to insert a dependency
  - totes_not_evil==0.0.2
- Possible to run arbitrary code at install time
  - Need to publish as “legacy” sdist
  - Almost all other languages have this ability
- Technique used by many legitimate packages
  - Also lots of illegitimate ones
- Code runs as local user
  - But we don’t need admin to steal user data!

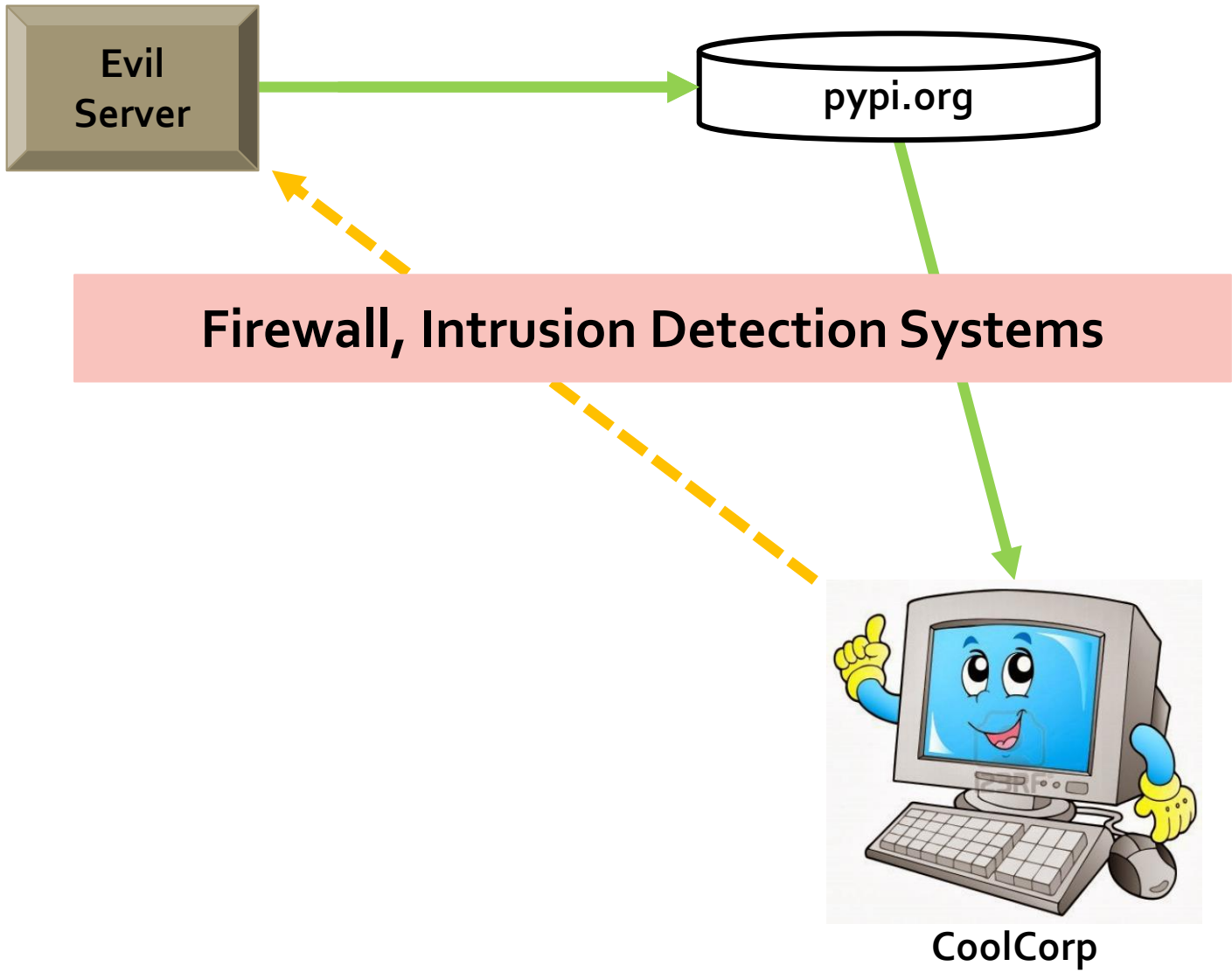
Step #3:

Data Exfiltration



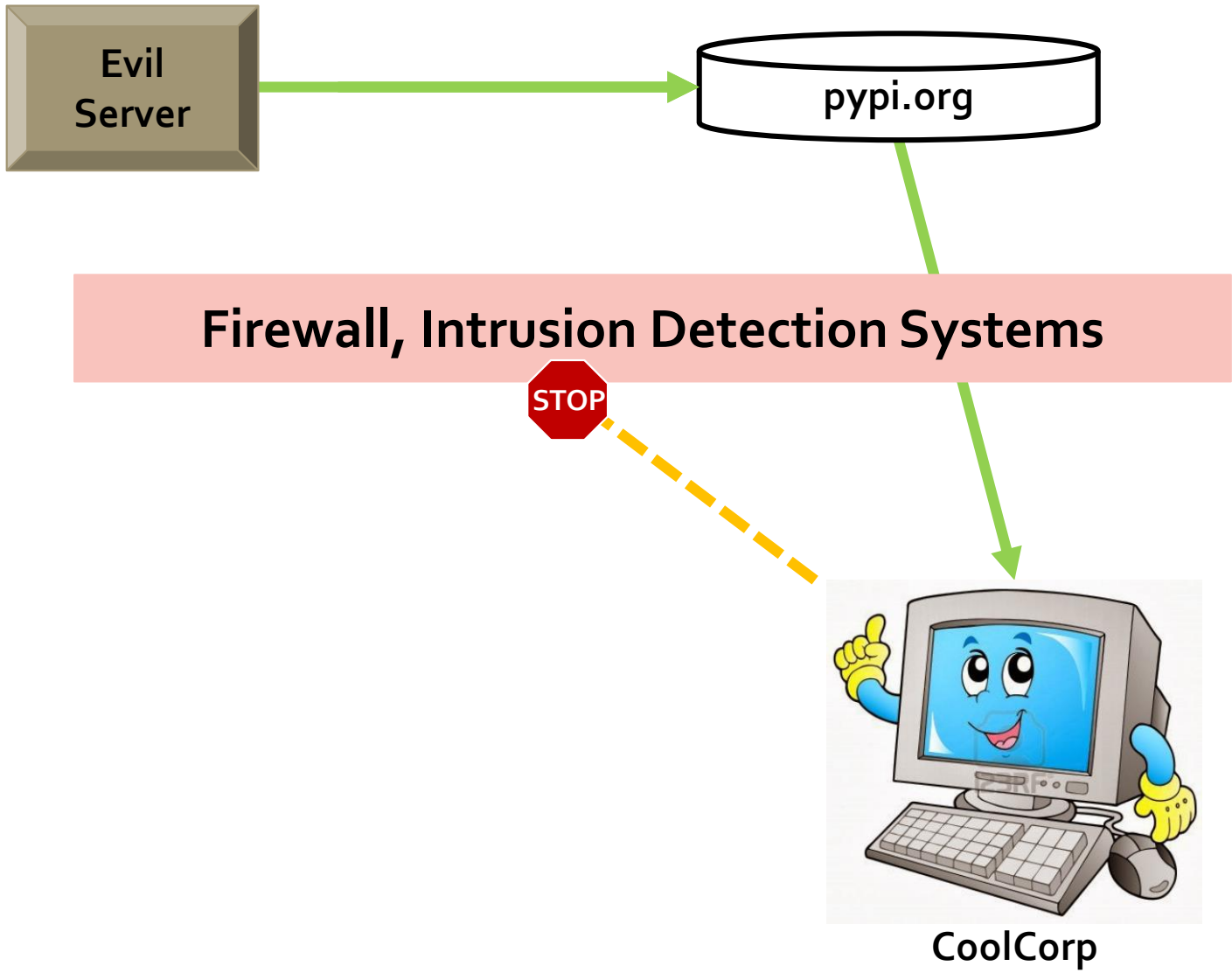
Step #3:

Data Exfiltration



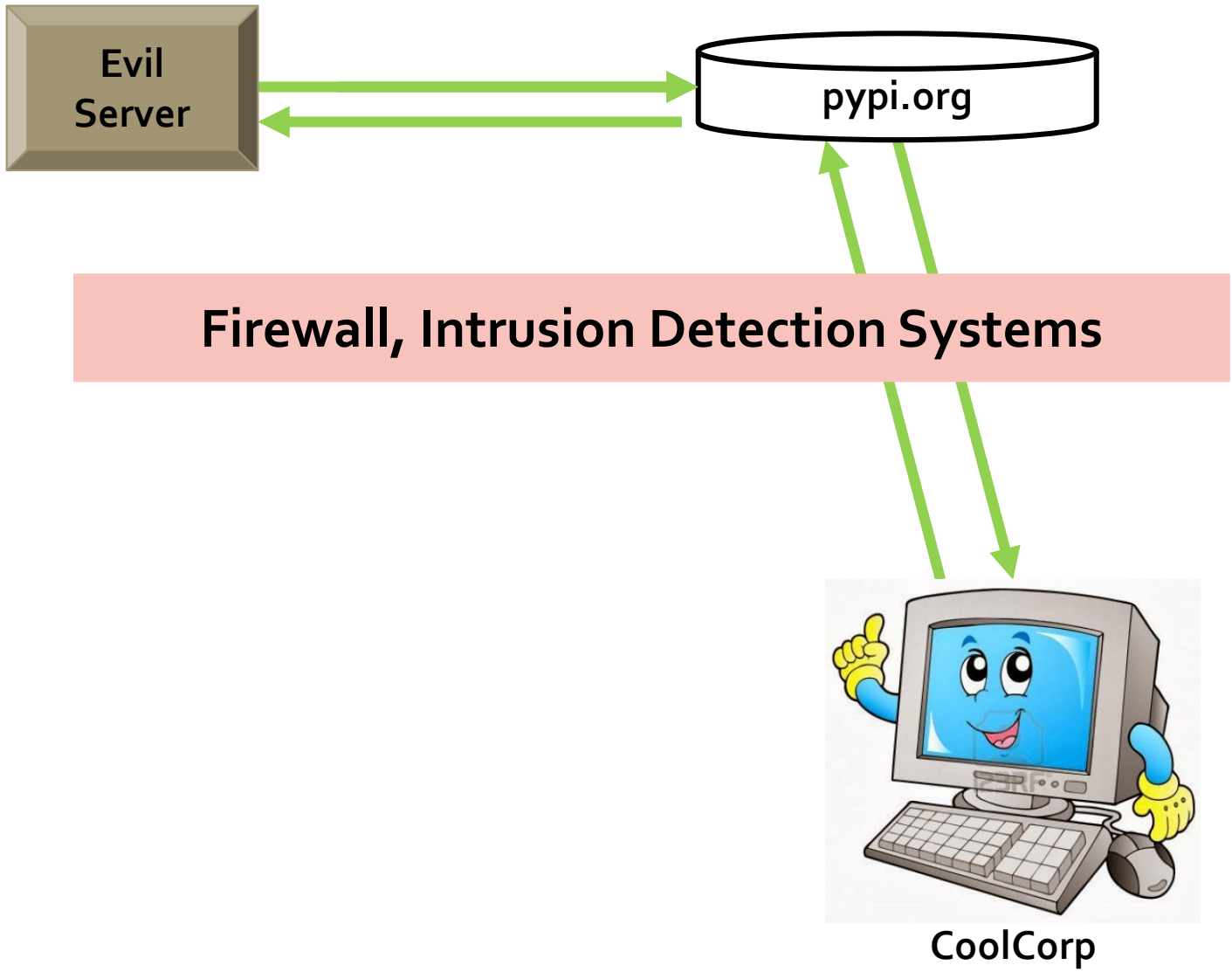
Step #3:

Data Exfiltration



Step #3:

Data Exfiltration



Step #3:

Data Exfiltration

- Network traffic logged/audited? Use PyPi!
- Traffic sent to Pypi domain under TLS, trusted cert
- Let's create a self-publishing package
- Re-upload our dependency with SECRETS.txt inside to PyPi

Real attacks in real life (for real)

In the real world:

event-stream

- Attack November 2018
  - Did #1 and #2
- Popular Node.JS library
  - Millions of daily downloads
- Social engineering to gain control of project
- Injected a single malicious dependency
- Targeted bitcoin wallets

Mitigations

Aka what can be done about it?

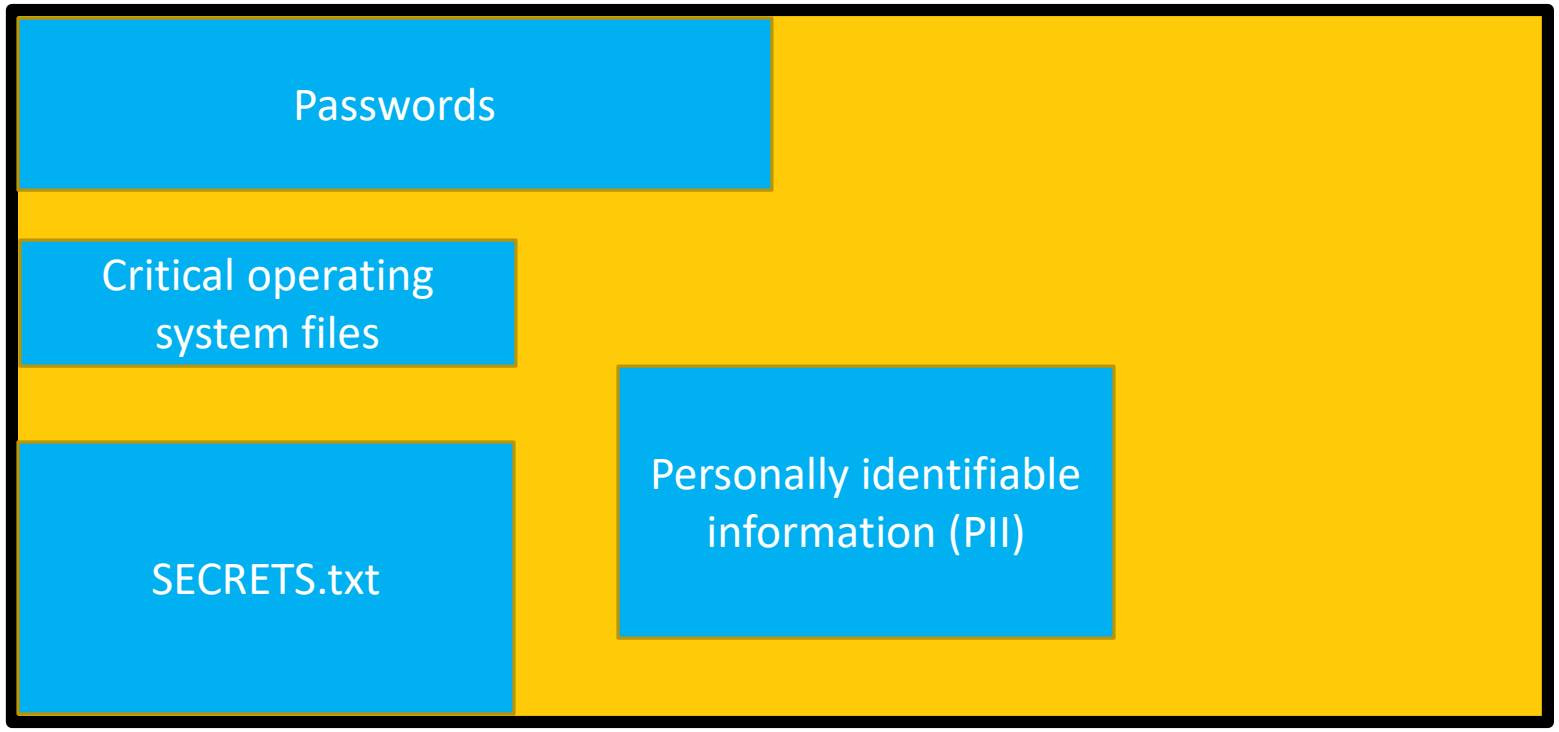
Mitigations

- Pyup.io Safety & JFrog X-Ray
  - Creates dependency map and flags “bad” packages
  - No code analysis
    - Relies on people to register “bad” packages
- PyCQA Bandit
  - Finds unsafe code by parsing AST
  - Designed for finding security issues within *your* code
    - Need to download dependencies first
    - >1000 legit packages throw “high” warnings

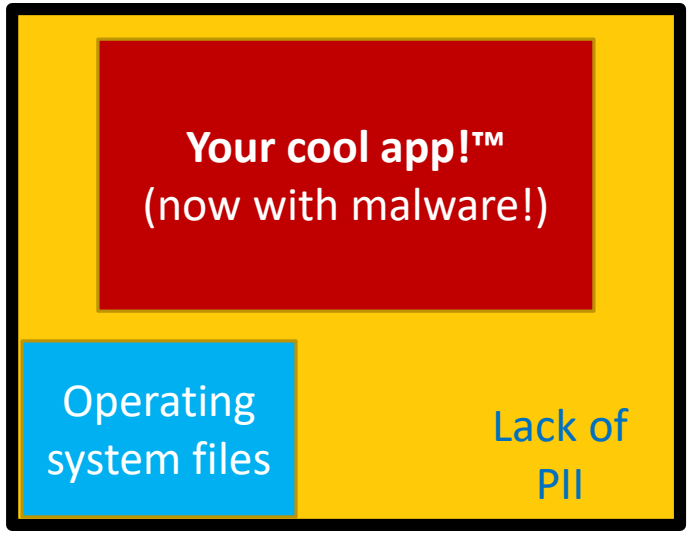
But, what if…

- You’re a small team without time to code-review the world?
  - Or knowledge
- You just need a one-time quick solution?
- You want to prototype alternative packages?

Your Machine



Contained Environment



DockEnv

VIRTUAL ENVIRONMENTS FOR PYTHON USING DOCKER:

THE EASY WAY!

DockEnv – Virtual Environments using Docker

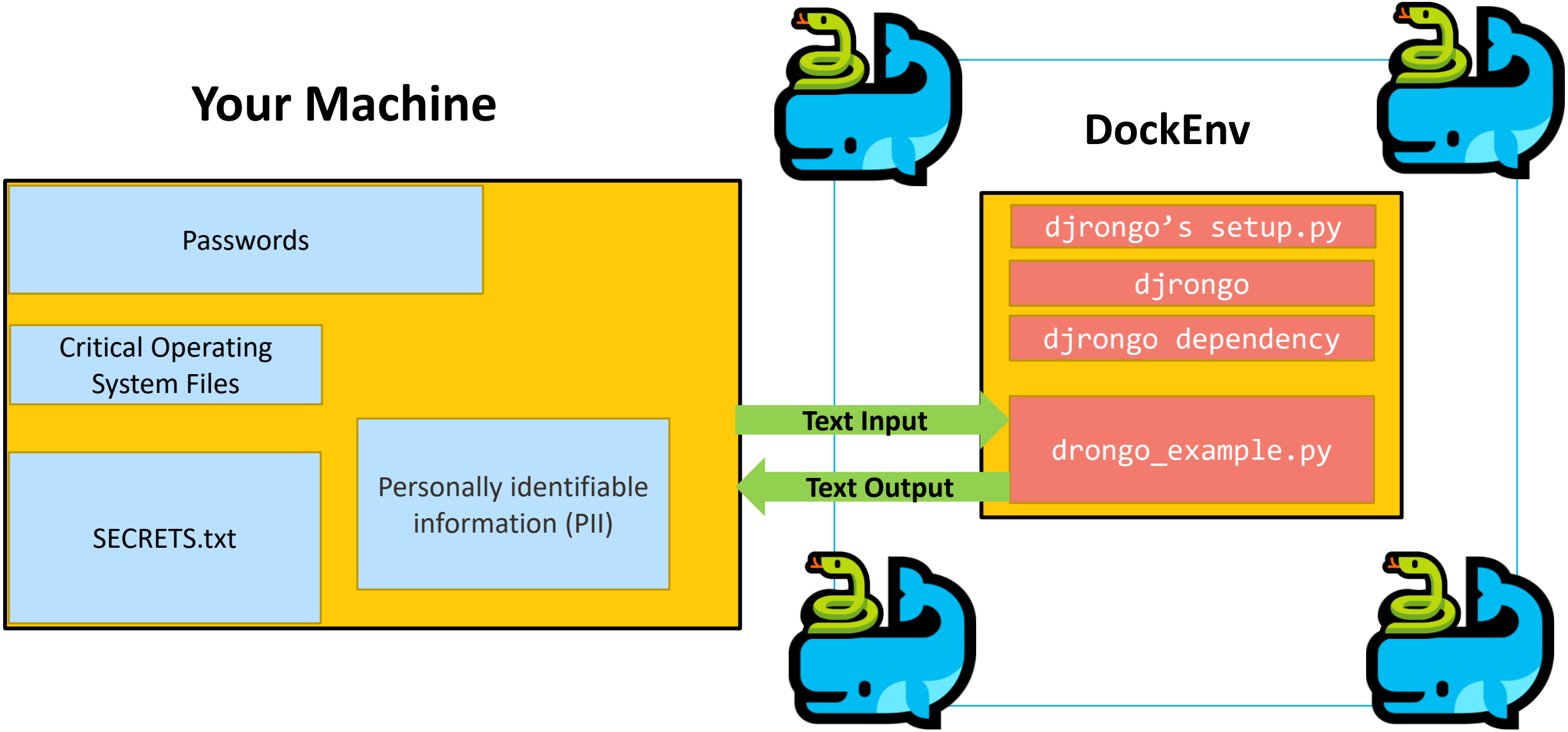
- Command line tool to safely install and run Python code
- All code runs inside a segregated Docker container, including installation
- Code cannot access any other part of your system
- After installation, virtual filesystem is read-only and has no network access
  - Unless you allow it to
- Containers only last as long as the code does

DockEnv – Create and Run

- Create a virtual environment, then run scripts inside

```
$> dockenv new my_env --package djrongo==0.0.1

$> dockenv run my_env djrongo_example.py --run-demo
This is some output from djrongo
```


Your Machine
Passwords
Critical Operating System Files
SECRETS.txt
Personally identifiable information (PII)
Text Input
Text Output
DockEnv
djrongo's setup.py
djrongo
djrongo dependency
drongo_example.py

DockEnv – Layered Security

Create environment with multiple packages:

```
$> dockenv new my_env –r requirements.txt
```

Pass in a folder of config files:

```
$> dockenv run my_env --mount ./config_dir test.py
```

Expose a network port for connection:

```
$> dockenv run my_env --expose-port 80 test.py
```

Enable writing to virtual disk

```
$> dockenv run my_env --writable-filesystem test.py
```

DockEnv adds security

- Prevents code execution on local machine
  - Code only executes inside the container, not on your machine
- Blocks creation of persistent backdoors
  - Code only exists for as long as it takes to run your script
- Prevents exfiltration of SECRETS.txt to Pypi
  - Only allows network access if enabled

DockEnv use cases

- Suitable for testing, quick solutions, and development
- Code running in DockEnv has access to as much data as you give it
  - In production, this means production data
  - Not a substitution for a full audit of your supply chain

Conclusion

Conclusion

- The Software Supply chain? Try Software Supply web
- Attacks on the software supply chain can and do happen
- If you don’t fully understand all you dependencies, this leaves you vulnerable
- Constant vigilance is needed to identify potential malicious activity
- Using tools such as Safety, X-Ray, Bandit, and DockEnv can help alleviate some of the risks
  - But isn’t a replacement for an in-depth understanding of your supply chain

QUESTIONS

- DockEnv available now

  https://github.com/pathtofile/dockenv

- Talk to me, please

  path.to.file[at]gmail

- References:

  https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident

  https://jfrog.com/xray/

  https://github.com/pyupio/safety

  https://github.com/PyCQA/bandit