




THE PYPY SANDBOX

Using the PyPy Sandbox to Explore Mobile Code
Sandboxing



Seth James Nielson



- B.S./M.S. Computer Science from Brigham Young University
- Ph.D. Computer Science from Rice University
- Past Experience: Software Engineer, Security Analyst
- Director of Advanced Research Projects at
The Johns Hopkins University Information Security Institute
- Founder, Chief Scientist, Crimson Vista Inc.

The PyPy Sandbox (An Introduction)

- The PyPy Project
 - *Replacement for Cpython*
 - *Faster execution of most Python code*
 - *Current versions: 2.7.13 and 3.5.3*
- The PyPy Sandbox is a Secondary Feature
 - *Execution of **untrusted** Python scripts*



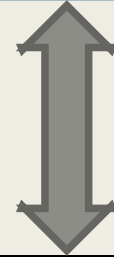
The Problem with Computers

- Computer processors do ***exactly*** what they're told
- They have no ability to decide if they ***should*** do what they're told
- What if they're told to do something ***harmful***?
- A lot of technology goes into figuring out what should be done
 - *Operating System*
 - *Anti-virus*
 - *Device permissions*

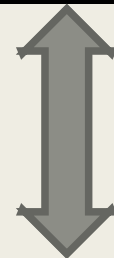
A Sandbox:

- The concept of a sandbox is an environment where destruction doesn't matter
- In practice, it is an interceptor between applications and the OS
- The interception layer enables:
 - *Policy Enforcement*
 - *“Sensor” Translations*
 - *“Command” Translations*

Untrusted
Application



SANDBOX



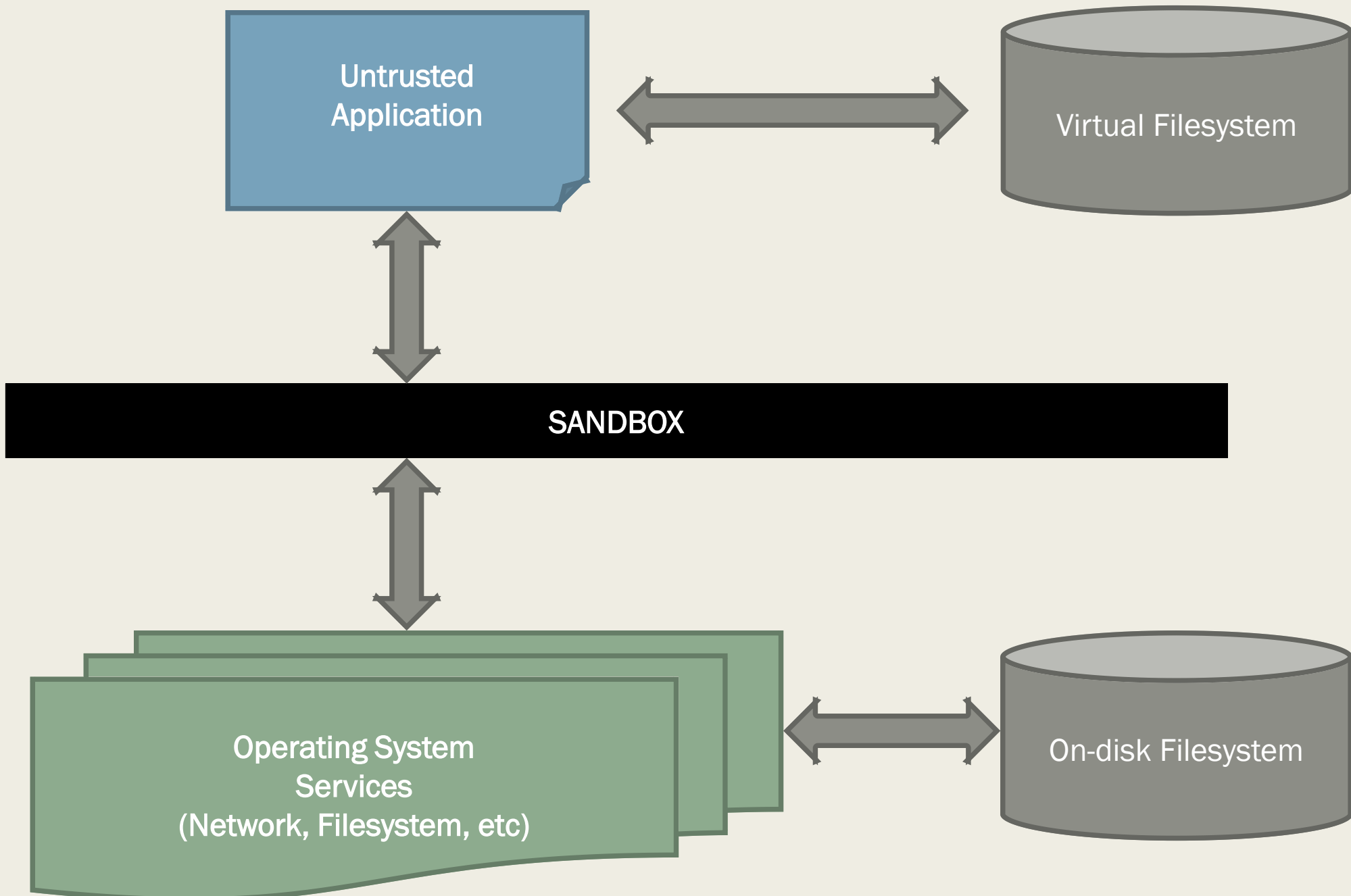
Operating System
Services
(Network, Filesystem, etc)

Policy Enforcement

- Most common use of a Sandbox
- Each incoming request to the operating system, *and response*, can be inspected
 - *Requests and responses can be allowed, denied, or modified*
 - *Policy based on request/response type, parameters, state of the system, etc*
 - *Examples:*
 - Network Access (Deny, Same-Origin Policy)
 - File Access (Read Only, Write-to-Temp)
 - Even memory allocations

Sensor/Command Translations

- Policy is not just about allow/deny but rewrite/modify
- Any risky call (e.g., syscall) can be rewritten with safer parameter
 - *(Or a risky call could be re-written to a safer call with similar semantics)*
- But a sometimes overlooked Sandbox capability is lightweight virtualization
 - *I like to call this “Sensor” translations*
 - *The Sandbox can control what the application “sees”*
 - *For example, it can present a virtual filesystem*
 - *Or provide alternative API replacements that are more secure*



Untrusted Application

Virtual Filesystem

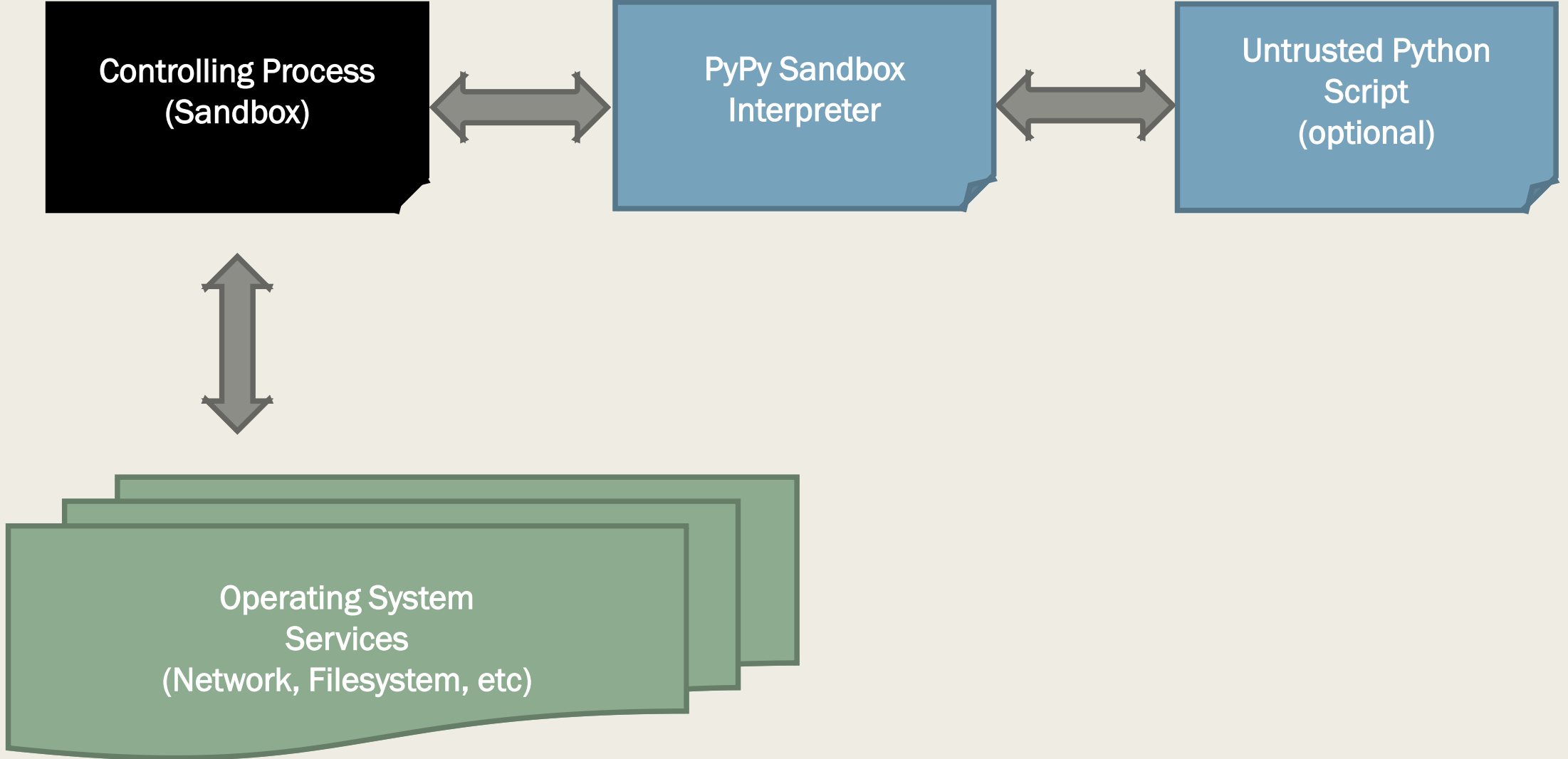
SANDBOX

Operating System Services
(Network, Filesystem, etc)

On-disk Filesystem

The PyPy Sandbox

- Creates a limited PyPy Interpreter
 - *No direct calls to the OS (system calls, etc)*
 - *Does not allow dynamic libraries, including compiled Python modules*
- Instead, a controlling process receives OS calls marshalled over a pipe
 - *This process provides the sandboxing and enforces security policies*
 - *For permitted calls, it performs the call itself and sends back the result*
 - *Or, it can modify the request and/or results*



Infinite Variety of Sandboxes

- Different controlling processes create different kinds of sandboxes
- Controlling process does not have to be Python
- The PyPy project provides a default controlling process called “pypy_interact.py”
 - *Can run a python “shell” or execute a script*
 - *Many OS subsystems completely disabled including network operations*
 - *Read only virtual file system*
 - /bin – virtual bin directory with pypy and a few required directories
 - /tmp – temp directory that potentially maps to a real directory
 - NOTE: the interpreter lives in the sandbox and executes the script from virtual /tmp!

```
$ mkdir my_sandbox_tmp
$ echo "this is a test" > my_sandbox_tmp/datafile.txt
$ ./pypy_interact.py --tmp=my_sandbox_tmp pypy3-c-sandbox
```

```
>>>> import os
>>>> os.listdir('/')
['dev', 'bin', 'tmp']
```

```
>>>> os.listdir('/tmp')
['datafile.txt']
```

```
>>>> f = open('/tmp/datafile.txt')
>>>> f.read()
'this is a test\n'
```

```
>>>> open('/tmp/newfile.txt', 'w+')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: '/tmp/newfile.txt'
>>>> open('/tmp/datafile.txt', 'a+')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
PermissionError: [Errno 1] Operation not permitted: '/tmp/datafile.txt'
```

Running an Untrusted Script

- Contents of “dangerous_script.py”

```
import os
print("Script Current Working Dir: {}".format(os.getcwd()))
print("Contents of root dir: {}".format(os.listdir('/')))
print("Try to delete /tmp dir with a system call.")
os.system('rm -rf /tmp')
```

```
$ ls my_sandbox_tmp/
```

```
dangerous_script.py  datafile.txt
```

```
./pypy_interact.py --tmp=my_sandbox_tmp pypy3-c-sandbox /tmp/dangerous_script.py
```

```
Script Current Working Dir: /tmp
```

```
Contents of root dir: ['bin', 'tmp', 'dev']
```

```
Try to delete /tmp dir with a system call.
```

```
Traceback (most recent call last):
```

```
  File "/tmp/dangerous_script.py", line 5, in <module>
```

```
    os.system('rm -rf /tmp')
```

```
RuntimeError
```

Sample Sandboxing Functions

```
def do_ll_os__ll_os_write(self, fd, data):
    if fd == 1:
        self._output.write(data.decode())
        self._output.flush()
        return len(data)
    if fd == 2:
        self._error.write(data.decode())
        return len(data)
    raise OSError("trying to write to fd %d" % (fd,))
```

```
def do_ll_os__ll_os_read(self, fd, size):
    f = self.get_file(fd, throw=False)
    if f is None:
        return super().do_ll_os__ll_os_read(fd, size)
    else:
        if not (0 <= size <= (2**64)):
            raise OSError(errno.EINVAL, "invalid read size")
        # don't try to read more than 256KB at once here
        return f.read(min(size, 256*1024))
```


Using PyPy in the Classroom

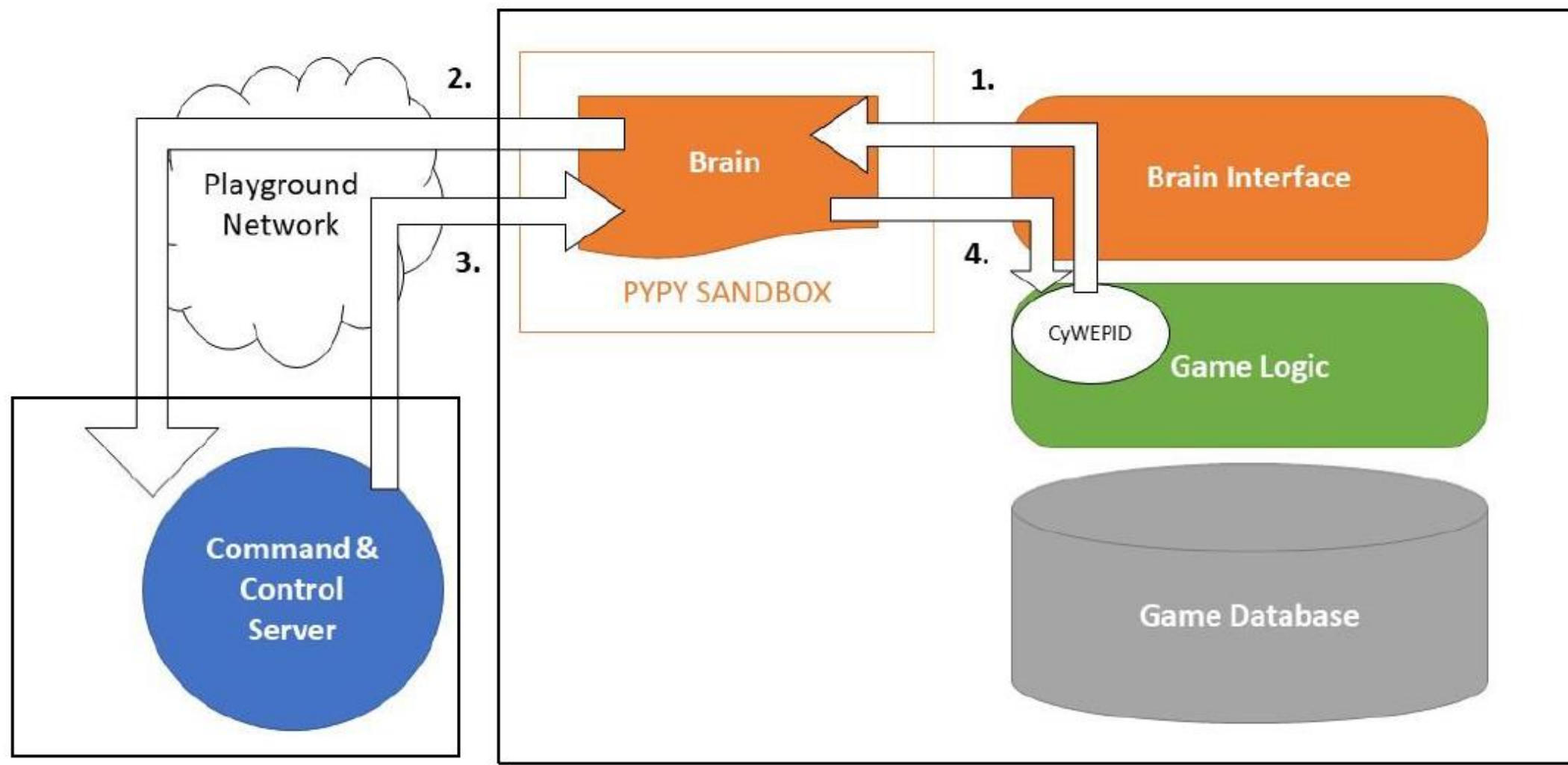
- Network Security at Johns Hopkins University
- Student Labwork:
 - *Uses “Playground,” an education overlay network created by the Author*
 - *Students create their own version of TCP within Playground*
 - *Students create their own version of TLS within Playground*
 - *Students build mobile code applications on top of Playground using PyPy*
 - Parallel processing (e.g., Traveling Salesman)
 - Adapt pypy_interact to support new features (writing to filesystem)

Modifying pypy_interact

- Requires students to carefully think about sandbox policies and features
- For example, implementing write
 - *Requires students to understand virtual file system*
 - *Implement policy for when writes are allowed*
 - Specific directories
 - Maximum size
 - *Argument sanitation (e.g., “../../..” doesn’t escape the sandbox)*
- Another example: implementing network operations

Developing Bot Brains

- Advanced Network Security
- “CyberWar_EDU” project
 - *Gameboard with semi-autonomous “bots”*
 - *Students can (re-)program the bots with a Python brain script*
 - *Each brain scripts run inside a PyPy sandbox instance*
 - *Each brain needs to connect to*
 - The Game Board (over TCP)
 - The Student’s command and control server (over Playground’s network)



Sandbox “Brain” Extensions

- Extended pypy_interact.py to brain_interact.py
- Virtual file system supports two special virtual files:
 - *“game://” which opens a socket to the gameboard*
 - *“<playground-protocol>://<host>:<port>” which connects to C&C*
- Allows writing within the /tmp directory so students can re-program their brains!

Sample Modified Sandbox Functions

```
def do_ll_os__ll_os_open(self, name, flags, mode):
    if name.startswith(b"game://"):
        host, port = '127.0.0.1', 10013
        try:
            protocol = asyncio_interface.sandbox_connect(host, port)
        except Exception as e:
            print("Exception on game connection: {}".format(e))
            raise RuntimeError("Could not open connection to game because {}".format(str(e)))
        fd = self.allocate_fd(protocol, ProtocolSocketWrapper())
        self.sockets[fd] = True
    return fd
```

The “Null” Brain

```
import time
import os

def brainLoop():
    gameSocket = open("game://", "rb+")
    ccSocket = open("default://20181.0.1.1:5000", "rb+")

    while True:
        gameData = os.read(gameSocket.fileno(), 1024)
        ccData = os.read(ccSocket.fileno(), 1024)

        if gameData: os.write(ccSocket.fileno(), gameData)
        if ccData: os.write(gameSocket.fileno(), ccData)

        if not gameData and not ccData:
            time.sleep(.5) # sleep half a second every time there's no data

if __name__=="__main__":
    try:
        brainLoop()
    except Exception as e:
        print("Brain failed because {}".format(e))
```

Eventual Goal for Lab Work

- Students reprogram bots over the network
- Students attempt to reprogram other student bots to take them over
- Eventually, want a student sandbox within the bot sandbox
 - *Bot sandbox is to protect the game from student malicious code*
 - *Student sandbox is to protect bot against false reprogramming*
 - *Give students a chance to create “firmware” that detects bad “software”*

Quick Review

- PyPy sandbox
 - *Provides lightweight Python sandboxing*
 - *A modified interpreter has no system calls*
 - *Dangerous calls are processed by a controlling process*
 - *Policy enforces allow, deny, and modify*
 - *Modify can be used to create a virtual system*
- Students can experiment extensively and gain insight into mobile code execution

Final Notes

- PyPy sandbox is a prototype. It is not ready for production code
- The current PyPy sandbox is somewhat broken for 2.7, inoperable for 3.5
- For my class, I fixed 3.5. I plan to submit my changes to PyPy shortly
- I have discussed providing on-going maintenance with the PyPy team

Thank You!

- Feel free to ask questions!

- Links:

- *The PyPy project:* <http://pypy.org>
- *Playground code:* <https://github.com/CrimsonVista/Playground3>
- *CyberWar EDU code:* <https://github.com/CrimsonVista/cybersecurity-war>
- *Playground paper:* <https://eric.ed.gov/?id=EJ1132824>
- *JHUISI:* <https://isi.jhu.edu/>