Hacking Like It's 2013
/* The Workshop */

#include “Itzik Kotler“
Agenda

- Pythonect
- Developing Domain-specific Language w/ Pythonect
- Hackersh
- Q&A
Pythonect

- *Pythonect* is a portmanteau of the words Python and Connect
- New, experimental, general-purpose dataflow programming language based on Python
- Current “stable“ version (True to Apr 9 2013): 0.4.2
- Made available under 'Modified BSD License'
- Influenced by: Unix Shell Scripting, Python, Perl
- Cross-platform (should run on any Python supported platform)
- Website: [http://www.pythonect.org/](http://www.pythonect.org/)
A few words on the Development

- Written purely in Python (2.7)
  - Works on CPython 2.x, and Jython 2.7 implementations
- Tests written in PyUnit
- Hosted on GitHub
- Commits tested by Travis CI
Installing and Using The Pythonect Interpreter

- Install directly from PyPI using easy_install or pip:
  - easy_install Pythonect
  OR
  - pip install Pythonect

- Clone the git repository:
  - git clone git://github.com/ikotler/pythonect.git
  - cd pythonect
  - python setup.py install
The Pythonect Interpreter

- Written and integrated with the Python environment:

  ```bash
  % pythonect
  Python 2.7.3 (default, Aug 1 2012, 05:14:39)
  [Pythonect 0.4.2] on linux2
  Type "help", "copyright", "credits" or "license" for more information.
  >>>
  ```
Dataflow Programming

Programming paradigm that treats data as something originating from a source, flows through a number of components and arrives at a final destination - most suitable when developing applications that are themselves focused on the "flow" of data.
Dataflow Example

A video signal processor which may start with video input, modifies it through a number of processing components (i.e. video filters), and finally outputs it to a video display.
Dataflow Example

Want to change a feed from a local file to a remote file on a website?

No problem!
Dataflow Example

Want to write the Video B&W Frame Processor output to both a screen and a local file?

No problem!
Dataflow Programming Advantages

- Concurrency and parallelism are natural
- Data flow networks are natural for representing process
- Data flow programs are more extensible than traditional programs
Dataflow Programming Disadvantages

- The mindset of data flow programming is unfamiliar to most programmers
- The intervention of the run-time system can be expensive
Dataflow Programming Languages

- Spreadsheets are essentially dataflow (e.g. Excel)
- VHDL, Verilog and other hardware description languages are essentially dataflow
- XProc
- Max/Msp
- ... Etc.
<Pythonect Examples>
'Hello, world' -> print
What do we have here?

- `->` is a Pythonect Control Operator, it means async forward.
- There's also `|` (i.e. Pipe) which means sync forward.
- `'Hello, world'` is a literal string
- `print` is a function
"Hello, world" -> [print, print]
["Hello, world", "Hello, world"] -> print
range(99, 0, -1) \
| [ _ % 2 == 0 ] \
-> str \n-> _ + " bottle(s) of beer on the wall," \n-> print \n-> _.split(' on')[0] + "." \n-> print \n-> print("Take one down, pass it around,"
Basic Pythonect Syntax Summary

- `->` is async forward.
- `|` (i.e. Pipe) is sync forward.
- `_` (i.e. Underscore) is current value in flow
<Pythonect Security Scripts/Examples>
ROT13 Encrypt & Decrypt

raw_input() -> _.encode('rot13') -> print
Check if FTP Server Supports Anonymous Login

'ftp.gnu.org' \ 
  -> ftplib.FTP \ 
  -> _.login() \ 
  -> print("Allow anonymous")
sys.argv[1] \n-> [str(_ + '/' + x) for x in open(sys.argv[2], 'r').read().split('\n')] \n-> [(_, urllib.urlopen(_))] \n-> _[1].getcode() != 404 \n-> print "%s returns %s" % (_[0], _[1], _[1].getcode())
Command line Fuzzer

```
['%s', '%n', 'A', 'a', '0', '!', '$', '%', '*', '+', ',', '-', '.', '/', ':']
| [_ * n for n in [256, 512, 1024, 2048, 4096]]
| os.system('/bin/ping ' + _)
```
(Multi-thread) Generic File format Fuzzer

open('dana.jpg', 'r').read() \
    -> itertools.permutations \
    -> open('output_' + hex(__hash__()) + '.jpg', 'w').write(''.join(_))
Compute MALWARE.EXE's MD5 & SHA1

"MALWARE.EXE" -> [os.system("/usr/bin/md5sum " + _), os.system("/usr/bin/sha1sum " + _)]
Compute MALWARE.EXE's Entropy

- **Entropy.py:**
  
  ```python
  import math
  def entropy(data):
      entropy = 0
      if data:
          for x in range(2**8):
              p_x = float(data.count(chr(x))) / len(data)
              if p_x > 0:
                  entropy += - p_x * math.log(p_x, 2)
      return entropy
  ```

- **Pythonect:**
  
  ```python
  "MALWARE.EXE" \
  -> open(_, 'r').read() \
  -> entropy.entropy \
  -> print
  ```
References / More Examples

- **My Blog**
  - Scraping LinkedIn Public Profiles for Fun and Profit
  - Fuzzing Like A Boss with Pythonect
  - Automated Static Malware Analysis with Pythonect

- **LightBulbOne (Blog)**
  - Fuzzy iOS Messages!
Pythonect Roadmap

- Support Python 3k
- Support Stackless Python
- Support IronPython
- Support GPU Programming
- Fix bugs, etc.
Questions?
Moving on!

Developing Domain-specific Language (DSL) with Pythonect
Domain-specific Language

- Domain-specific language (DSL) is a mini-language aiming at representing constructs for a given domain.
- DSL is effective if the words and idioms in the language adequately capture what needs to be represented.
- DSL can also add syntax sugar.
Why?

Why create a custom tag or an object with methods?

Elegant Code Reuse

Instead of having to recode algorithms every time you need them, you can just write a phrase in your DSL and you will have shorter, more easily maintainable programs.
Example for DSL's

- Programming Language R
- XSLT
- Regular Expression
- Graphviz
- Shell utilities (awk, sed, dc, bc)
- Software development tools (make, yacc, lex)
- Etc.
<DSL/Examples>
Example #1: XSLT 'Hello, world'

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="p">
        Hello world! - From hello.xsl.
    </xsl:template>
</xsl:stylesheet>
```
Example #2: Graphviz/DOT 'Hello, world'

digraph G
{
  Hello → World
}

Domain-specific Language with Pythonect

• Pythonect provides various features to let you easily develop your own DSLs:
  – Built-in Python module Autoloader
  – Concurrency (Threads & Processes)
  – Abstract Syntax (i.e. Generic Flow Operators)
Built-in Python AutoLoader

• The AutoLoader loads Python modules from the file system when needed.
• In other words, no need to `import` modules explicitly.
• The sacrifice is run-time speed for ease-of-coding and speed of the initial `import() ing`. 
'Hello, world' -> \texttt{string.split}

\begin{itemize}
  \item i.e.
  \begin{verbatim}
  import string
  return string.split
  \end{verbatim}
\end{itemize}
Concurrency (Threads & Processes)

- **Multi-threading:**
  - 'Hello, world' -> [print, print]

- **Multi-processing:**
  - 'Hello, world' -> [print, print]

- **Mix:**
  - 'Hello, world' -> [print, print &]
Abstract Syntax

- Brackets for Scope:
  - [ ]

- Arrows and Pipes for Flows:
  - | and ->

- Dict and Logical Keywords for Control Flow:
  - {} and not/or/and
So, imagine the following is a real script:

```python
from_file('malware.exe') \ 
    -> extract_base64_strings \ 
    -> to_xml
```
IT IS!
(with Pythonect)
Meet SMALL

Simple Malware AnaLysis Language

- Toy language for analyzing malware samples
- Single Python file (14 functions, 215 lines of text)
- Runs on top of Pythonect
SMALL Features

- Extract IPv4 Addresses from Binaries
- Extract Base64 Strings from Binaries
- Calculate MD5/SHA1/CRC32
- Determine File Type (via /usr/bin/file)
- Create XML Reports
How Does SMALL Work?

- SMALL functions are divided into two groups:
  - Root, these functions start a flow
  - Normal, these functions continues or closes the flow

- Root functions accept `String` and return `dict`
  - e.g. `from_file()`

- Normal functions accept `dict` and return `dict`
  - e.g. `extract_base64_strings()`
<Pythonect/Security DSL (i.e. SMALL) Examples>
How to Start the SMALL Interpreter

```
pythonect -m SMALL -i
```

- The '-m' means - run library module as a script
- The '-i' means - inspect interactively after running script
- Just like Python :)
Extract Base64 Strings and Save As XML

```python
from_file('malware.exe') \n    -> extract_base64_strings \n        -> to_xml
```
from_file('malware.exe') \ 
   -> extract_ipv4_addresses \ 
      -> to_xml
Compute MD5, SHA1, CRC32, and File Type

from_file('malware.exe') \ 
  -> md5sum \ 
    -> sha1sum \ 
      -> crc32 \ 
        -> file_type \ 
          -> to_xml
Other (Potential) Security Domains:

- Reverse Engineering
- Malware Analysis
- Penetration Testing
- Intelligence Gathering
- Fuzzing
- Etc.
Questions?
Moving on!

Hackersh
Hackersh

- *Hackersh* is a portmanteau of the words Hacker and Shell
- Shell (command interpreter) written with Pythonect-like syntax, built-in security commands, and out of the box wrappers for various security tools
- Current “stable” version (True to Apr 1 2013): 0.1.0
- Made available under GNU General Public License v2 or later
- Influenced by: Unix Shell Scripting and Pythonect
- Cross-platform (should run on any Python supported platform)
- Website: [http://www.hackersh.org](http://www.hackersh.org)
A few words on the Development

- Written purely in Python (2.7)
- Hosted on GitHub
Motivation

• Taking over the world

• Automating security tasks and reusing code as much as possible
Problems

• There are many good security tools out there...
  - but only a few can take the others output and run on it
  - but only a few of them give you built-in threads/processes controlling for best results

• No matter how well you write your shell script, the next time you need to use it - for something slightly different - you will have to re-write it
Hackersh – The Solution

• Hackersh provides a “Standard Library“ where you can access your favorite security tools (as Components) and program them as easy as a Lego
• Hackersh lets you automagically scale your flows, using multithreading, multiprocessing, and even a Cloud
• Hackersh (using Pythonect as it's scripting engine) gives you the maximum flexibility to re-use your previous code while working on a new slightly-different version/script
Installing and Using The Hackersh

• Install directly from PyPI using easy_install or pip:
  - easy_install Hackersh
  OR
  - pip install Hackersh

• Clone the git repository:
  - git clone git://github.com/ikotler/hackersh.git
  - cd hackersh
  - python setup.py install
Implementation

- Component-based software engineering
  - External Components
    - Nmap
    - W3af
    - Etc.
  - Internal Components
    - URL (i.e. Convert String to URL)
    - IPv4_Address (i.e. Convert String to IPv4 Address)
    - Etc.
Component as Application

• Components accepts command line args:
  - "localhost" -> hostname -> nmap("-P0")

• They also accept internal flags options as:
  - "localhost" -> hostname -> nmap("-P0", debug=True)
Input/Output: Context

• Every Hackersh component (except the Hackersh Root Component) is standardized to accept and return the same data structure – Context.

• Context is a dict (i.e. associative array) that can be piped through different components
Same Context, Different Flow

- "http://localhost" -> url -> nmap -> ping
  - Port scan a URL, if *ANY* port is open, ping it
- "http://localhost" -> url -> ping -> nmap
  - Ping the URL, if pingable, scan for *ANY* open ports
Ask The Context

- Context stores both Data and Metadata
- The Metadata aspect enables potential AI applications to fine-tune their service selection strategy based on service-specific characteristics
Conditional Flow

"http://localhost" \n -> url \n    -> nmap \n        -> [_['PORT'] == '8080' and _['SERVICE'] == 'HTTP'] \n            -> w3af \n                -> print
Hackersh High-level Diagram

Literal (e.g. String) → Root Component (e.g. URL) → Context → Component → ...
<Hackersh Scripts/Examples>
TCP & UDP Ports Scanning

"localhost" -> hostname -> nmap
Class C (256 Hosts) Ping Sweep

'192.168.1.0/24' -> ipv4_range -> ping
Web Server Vulnerability Scanner

'127.0.0.1' -> ipv4_address -> nmap -> nikto
Fork: Target as Hostname + Target as IP

"localhost" \n   -> hostname \n      -> [nslookup, pass] -> ...

Target -> Built-in Component
         ↓
   Target as Hostname
         ↓
   Target as IPv4 Addr.
         ↓
      …
Black-box Web App Pentration Testing

"http://localhost"
  -> url 
  -> nmap 
  -> browse 
  -> w3af 
  -> print

Target ➔ Built-in Component ➔ External Component ➔ Built-in Component ➔ External Component ➔ Built-in Component
Hackersh Roadmap

- Unit Tests
- Documentation
- More Tools
  - Metasploit
  - OpenVAS
  - TheHarvester
  - Hydra
  - ...
- Built-in Commands
- Plugins System
- <YOUR IDEA HERE>
Hackersh Official TODO

Questions?
Thank you!

My Twitter: @itzikkotler
My Email: ik@ikotler.org
My Website: http://www.ikotler.org

Pythonect Website: http://www.pythonect.org
Hackersh Website: http://www.hackersh.org

Feel free to contact me if you have any questions!