Gatelogic FRP framework

...200 lines of Python I still regret...

Marek Majkowski  @majek04
Who we are?
Reverse proxy

- Optimizations
- Caching
- DDoS protection
- Security
Attacked
Signal

<table>
<thead>
<tr>
<th>Mpps</th>
<th>rule hint</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.160</td>
<td>dnsbpf --</td>
<td>ip=173.245.59.</td>
</tr>
<tr>
<td>3.892</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>4.169</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>3.750</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>4.098</td>
<td>dnsbpf --</td>
<td>ip=173.245.59.</td>
</tr>
<tr>
<td>0.965</td>
<td>dnsbpf --</td>
<td>ip=173.245.59.</td>
</tr>
<tr>
<td>3.910</td>
<td>dnsbpf --</td>
<td>ip=173.245.59.</td>
</tr>
<tr>
<td>0.973</td>
<td>dnsbpf --</td>
<td>ip=173.245.59.</td>
</tr>
<tr>
<td>3.811</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>3.770</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>3.879</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>3.857</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>3.975</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>0.545</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>0.315</td>
<td>dnsbpf --</td>
<td>ip=173.245.58.</td>
</tr>
<tr>
<td>0.415</td>
<td>dnsbpf --</td>
<td>ip=173.245.59.</td>
</tr>
</tbody>
</table>

INCOMING/UDP samples=11836 pps={total= 65.176M, accounted= 65.174M}
Mitigation

servers

iptables rules

mitigation

command line

Operator
Mitigation

$ gatekeeper dnsbpf list

[ ] Imported: 0 removed: 0 trusted: 25
--expiry=365d --ip=173.245.
--expiry=7d --ip=
--expiry=7d --ip=
--expiry=7d --ip=
--expiry=7d --ip=
--expiry=7d --ip=
--ip=173.245.
--ip=173.245.
--ip=173.245.
--ip=173.245.
--ip=173.245.
--ip=173.245.
"Business logic"

switch → signal → mitigation

switch → signal

servers → iptables rules → mitigation
12 months later...
--ip=1.2.3.4 example.com

--ip=1.2.3.4 example.com --qps=100

Business logic
--ip=1.2.3.4 example.com

--ip=1.2.3.4 example.com --qps=500

example.com = FREE | PAID

Business logic
example.com subdomains: (www, ns1, ns2)

--ip=1.2.3.4 example.com

example.com = FREE | PAID

--ip=1.2.3.4 example.com --except www,n1,ns2 --qps=500

Business logic
def dns_mitigation(attack, plan, subdomains, toggles):
    domain = attack['domain']

    if toggles['all_mitigations_disabled']: return

    qps = 100
    if plan[domain] == 'business':
        qps = 500

    mitigation =
        attack['description'] + \
        ' --qps=%s' % qps + \
        ' --except=%s'.join(subdomains[domain])

    return mitigation
def dns_mitigation(attack, plan, subdomains, toggles):
    domain = attack['domain']

    if toggles['all mitigations disabled']: return

    qps = 100
    if plan[domain] == 'business':
        qps = 500

    mitigation =
        attack['description'] + \
        ' --qps=%s' % qps + \
        ' --except=%s'.join(subdomains[domain])

    return mitigation
Business logic

• Hard problem!
• Multiple DB lookups
• Wait for operator confirmation
• Critical path
Functional reactive programming
Data transformation pipe (dataflow)

Side effects

Input #a

Input #b

a

b

c

render

Input #c

interaction

reaction
models - Excel

<table>
<thead>
<tr>
<th>Stock Name</th>
<th>Symbol</th>
<th>Shares</th>
<th>Purchase Price</th>
<th>Cost Basis</th>
<th>Current Price</th>
<th>Market Value</th>
<th>Gain/Loss</th>
<th>Dividend/Share</th>
<th>Annual Yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>AAPL</td>
<td>100</td>
<td>$90.00</td>
<td>$9,000.00</td>
<td>$144.13</td>
<td>$14,413.27</td>
<td>$14,269.14</td>
<td>$2.28</td>
<td>1.58%</td>
</tr>
<tr>
<td>Microsoft</td>
<td>MSFT</td>
<td>200</td>
<td>$32.00</td>
<td>$6,400.00</td>
<td>$65.57</td>
<td>$13,114.14</td>
<td>$13,048.57</td>
<td>$1.56</td>
<td>2.38%</td>
</tr>
<tr>
<td>Salesforce</td>
<td>CRM</td>
<td>150</td>
<td>$25.00</td>
<td>$3,750.00</td>
<td>$82.57</td>
<td>$12,385.50</td>
<td>$12,302.93</td>
<td>$0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>Oracle</td>
<td>ORCL</td>
<td>250</td>
<td>$50.00</td>
<td>$12,500.00</td>
<td>$44.56</td>
<td>$11,138.75</td>
<td>$11,094.20</td>
<td>$0.64</td>
<td>1.44%</td>
</tr>
<tr>
<td>Hewlett Packard Enterprise</td>
<td>HPE</td>
<td>500</td>
<td>$18.00</td>
<td>$9,000.00</td>
<td>$17.69</td>
<td>$8,842.50</td>
<td>$8,824.82</td>
<td>$0.26</td>
<td>1.47%</td>
</tr>
<tr>
<td>Alphabet</td>
<td>GOOG</td>
<td>100</td>
<td>$225.00</td>
<td>$22,500.00</td>
<td>$833.36</td>
<td>$83,336.00</td>
<td>$82,502.64</td>
<td>$0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>Intel</td>
<td>INTC</td>
<td>200</td>
<td>$22.00</td>
<td>$4,400.00</td>
<td>$36.07</td>
<td>$7,213.00</td>
<td>$7,176.94</td>
<td>$1.09</td>
<td>3.02%</td>
</tr>
<tr>
<td>Cisco</td>
<td>CSCO</td>
<td>225</td>
<td>$18.00</td>
<td>$4,050.00</td>
<td>$33.24</td>
<td>$7,478.78</td>
<td>$7,445.54</td>
<td>$1.16</td>
<td>3.49%</td>
</tr>
<tr>
<td>Qualcomm</td>
<td>QCOM</td>
<td>185</td>
<td>$65.00</td>
<td>$12,025.00</td>
<td>$56.48</td>
<td>$10,447.88</td>
<td>$10,391.40</td>
<td>$2.12</td>
<td>3.75%</td>
</tr>
<tr>
<td>Amazon</td>
<td>AMZN</td>
<td>50</td>
<td>$800.00</td>
<td>$40,000.00</td>
<td>$879.64</td>
<td>$43,882.00</td>
<td>$43,984.36</td>
<td>$0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>Redhat</td>
<td>RHT</td>
<td>100</td>
<td>$95.00</td>
<td>$9,500.00</td>
<td>$86.26</td>
<td>$8,626.00</td>
<td>$8,539.74</td>
<td>$0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>Facebook</td>
<td>FB</td>
<td>1000</td>
<td>$17.00</td>
<td>$17,000.00</td>
<td>$141.64</td>
<td>$141,640.00</td>
<td>$141,498.36</td>
<td>$0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>Twitter</td>
<td>TWTR</td>
<td>500</td>
<td>$45.00</td>
<td>$22,500.00</td>
<td>$14.61</td>
<td>$7,302.55</td>
<td>$7,287.94</td>
<td>$0.00</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
models - Materialized data
models - Signals
Pure FRP is useless

- Weird language - (ELM anyone?)
- Fixed signal flow
- Strictly no side-effects
Dirty FRP is awesome

• Weird language
  • Python

• Fixed signal flow
  • Attacks come and go, but patterns fixed

• Strictly no side-effects
  • Dynamic "subscriptions", but idempotent
Prior art - Trellis

Event-Driven Programming The Easy Way, with `peak.events.trellis`

(NOTE: As of 0.7a1, many new features have been added to the Trellis API, and some old ones have been deprecated. If you are upgrading from an older version, please see the porting guide for details.)

Whether it's an application server or a desktop application, any sufficiently complex system is event-driven -- and that usually means callbacks.

Unfortunately, explicit callback management is to event-driven programming what explicit memory management is to most other kinds of programming: a tedious hassle and a significant source of unnecessary bugs.
Software Transactional Memory (STM) And Observers

The Trellis is built on a simple Python STM (Software Transactional Memory) and "Observer Pattern" implementation. This document specifies how that implementation works, and tests it.

You should read this document if you plan to implement your own Trellis cell types or other custom data structures, or if you just want to know how things work "under the hood".

Table of Contents

- STM History
  - Commit/Abort Notices
  - Undo, Rollback and Save Points
  - Commit Callbacks
  - Logged Setattr
- The Observer Framework
  - Subjects, Listeners, and Links
    - Subjects
    - Listeners
  - Controllers
  - The Singleton Controller
  - Creating Custom Cell Types (TBD)
Gatelogic!

Gatelogic - somewhat reactive programming framework

Gatelogic is a functional reactive programming / software transactional memory inspired framework.

In classic FRP programming model, the signal graph is defined on the start and doesn't change during the lifetime of a program. For our usage we needed something more dynamic. We also wanted to do express the computable code in an easy to understand Python, as opposed to a classic composition of pure functions.

The idea is to write pretty simple, side-effect free functions that describe some business logic, while allowing them to dynamically "subscribe" arbitrary signal sources.

For example, a basic "action" code could look like:

```python
def action(row):
    if confirm_hub.get('all_mitigations_enabled').value != 'True':
        return None
```

This will actually "subscribe" to `confirm_hub/all_mitigations_enabled` input signal. Note, that this subscription is declared in the code, as opposed to predefined as input. Change of that input signal value will trigger recomputation of all dependent action functions.

https://github.com/cloudflare/gatelogic
GateLogic

- Input - ReadableHub
  - update(full_data)

- Processing - ComputableHub
  - maintain(key, function)
  - unmaintain

- Subscriptions - QueryHub
  - update(full_data)
Input data - a dict

```json
{
    '00001': {ip:'1.2.3.4', port: 80, domain: 'bar.com'},
    '00002': {ip:'1.2.3.5', port: 80, domain: 'foo.com'},
    ...
}
```
update()

ReadableHub

```javascript
update
{
  'attack1': 'example.com',
  ...
}
```
def on_hook(_, kind, k, row):
    if kind == 'add':
        mitigations.maintain(k, action, row)
    if kind == 'delete':
        mitigations.unmaintain(k)

subscribe(readable, on_hook)

def action(row):
    return None
ComputableHub → OutputHub → ?
Materialized
def action(row, plan_hub, subdomain_hub, toggle_hub):
    domain = row.value
    if not domain:
        return None

    if toggle_hub.get('all_mitigations_disabled').value != 'True':
        return None

    qps = 100
    if plan_hub.get(domain).value in ('business', 'b'):
        qps = 500

    sd = (subdomain_hub.get(domain).value or '').split(' ')

    mitigation = "
        domain + "
        ' --qps=%s ' % qps + "
        ' '.join( '--except=%s' % s for s in sd)
    return mitigation
It works!

• Solid foundation!
• Composable!
• Scalable
• Maintainable
• But:
  • no event loop
  • lacks higher-order abstractions
204 loc

<table>
<thead>
<tr>
<th>Language</th>
<th>files</th>
<th>blank</th>
<th>comment</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>3</td>
<td>61</td>
<td>24</td>
<td>204</td>
</tr>
<tr>
<td>SUM:</td>
<td>3</td>
<td>61</td>
<td>24</td>
<td>204</td>
</tr>
</tbody>
</table>

```bash
marek@ubuntu-saucy:~/cloudflare/gatelogic/gatelogic$ cloc *py
  3 text files.
  3 unique files.
  0 files ignored.

http://cloc.sourceforge.net v 1.60  T=0.01 s (240.8 files/s, 23197.8 lines/s)
```
Thanks!

• FRP is grea
  • http://www.flapjax-lang.org/
  • https://www.youtube.com/watch?v=mEvo6TVAf64
  • https://www.youtube.com/watch?v=Agu6jipKfYw

https://github.com/cloudflare/gatelogic

marek@cloudflare.com @majek04