Masquerading Malicious DNS Traffic
Bayesian Inference, Rainier, Spark

David Rodriguez
March 28, 2019
The Outline

- Masquerading DNS Traffic
- Time Series Modeling
- Rainier + Spark
- Anomaly Detection
The Outline

Masquerading DNS Traffic

Time Series Modeling

Rainier + Spark

Anomaly Detection
You've been chosen for a top-secret mission: Threat Busters.

Gather threat intel to defend your network.

Accept the mission
Part 1  DNS Resolution

Many More DNS Records

Mail Server

Web Server IP Address

Cisco Umbrella

180 Billion Per Day
Part 1 Protection 101

- Phishing
- Compromised Account
- Malvertising
- Ransomware
- Worms
- Virus
Part 1 Definition

Masquerading Traffic = Masquerading Users + Compromised Websites
Part 1 Masquerading Users
Part 1 Compromised Websites

- Typical Visitors
- Phished
- Browser Redirect
- Malicious Webpage
- Compromised Server
- Backdoor Vulnerability
Part 1  Masquerading DNS Traffic
Part 1 Emotet Campaign

Phishing Email

User Click Links or Opens Attachments to Email

Links or Macros Make DNS Requests

Malware Downloaded

Emotet Runs Code in Process and Registers Computer with C2 Server

Masquerading Traffic
Part 1  Emotet Campaign
The Outline

- Masquerading DNS Traffic
- Time Series Modeling
- Rainier + Spark
- Anomaly Detection
Part 2 Time-Series Analysis

\[ \psi \quad \text{Probability of Demand} \]

\[ \mu \quad \text{Expected Demand when non-zero} \]
Part 2 Croston’s Method

Note: \[ \mu_{new} = \mu \cdot w + (1 - w) \cdot x \quad \psi_{new} = \psi \cdot w + (1 - w) \cdot \chi_{x > 0} \]
Part 2 Bayesian Approach

\[ \begin{align*}
X & \quad \text{Probability Distribution} \\
Y & \\
\psi & \quad \text{Probability of Demand} \\
\mu & \quad \text{Expected Demand when non-zero}
\end{align*} \]
Part 2 Bayesian Approach
Part 2 Mixture Models

\[ f(x) \cdot \psi + (1 - \psi) \cdot g(x) \]
Part 2 Discrete Models
Part 2 Continuous Models

\[ f(x) \cdot \psi + (1 - \psi) \cdot g(x) \]
The Outline

Masquerading DNS Traffic

Time Series Modeling

Rainier + Spark

Anomaly Detection
Part 3 MCMC Methods

Observations

Proposed Distribution

Sampling From Distribution

Rejection of Samples

Observations

Proposed Distribution
Part 3 MCMC Methods

PyMC3

Probabilistic Programming in Python

Quickstart

Installation

Via conda-forge:

conda install -c conda-forge pvmc3

Rainier

Rainier provides an idiomatic, high-performance functional Scala API for bayesian inference via Markov Chain Monte Carlo.

Rainier allows you to describe a complex prior distribution by composing primitive distributions using familiar combiners like map, flatMap, and zip, condition that prior on your observed data, and, after an inference step, sample from the resulting posterior distribution.

Underlying this is a static scalable compute graph with auto-differentiation and very fast CPU-based execution.

It is implemented in pure Scala, with minimal external dependencies and no JNI files, and as such is convenient to deploy, including to Spark or Hadoop clusters.

Rainier currently provides two samplers: affine-invariant MCMC, an ensemble method popularized by the Emcee package in Python, and Hamiltonian Monte Carlo: a gradient-based method used in Stan and PyMC3.
Depending on your background, you might think of Rainier as aspiring to be either: “Stan, but on the JVM” or “Tensorflow, but for small data”.

~ README
Part 3 Rainier Methods

```scala
val data = List(2,0,0,3,3,0,3,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0)
val zipModel = for {
  psi <- Beta(1,1).param
  lambda <- Normal(5, 5).param
  zip <- Poisson(lambda).zeroInflated(psi).fit(data)
} yield (psi, lambda)
plot2D(zipModel.sample())
```
Part 3 PyMC Methods

```python
import numpy as np
import pymc3 as pm
import matplotlib.pyplot as plt
import seaborn as sns

data = [7.65024100665554e-05, 3.923801883327712e-05, 3.0, 3.0, 3.0]

model = pm.Model()

with model:
    p = pm.Beta('p', 1, 1)
    mu = pm.Normal('mu', 10, 10)
    sd = pm.HalfNormal('sd', 10)
    normal = pm.Normal('normal', mu, sd)
    u = pm.Uniform('u', 0, 1e-5)

    mixture = pm.Mixture('mixture', [1-p, p], comp_dists=[normal.distribution, u.distribution], observed=data)

with model:
    trace = pm.sample(1000, tune=1000)
    pm.traceplot(trace, varnames=['p', 'mu', 'sd'])
plt.figure()
plt.show()
```
Part 3 Rainier + Spark
Part 3 Rainier + Spark

Spark Job
150 Million
Paid-Level Domains

Hourly Aggregations

Daily Aggregations

Spark Job

Filtering Heuristics

Rainier Simulations
import com.stripe.rainier.core.{Normal, Poisson}
import com.stripe.rainier.sampler.{RNG, ScalaRNG}
import org.apache.spark.{SparkConf, SparkContext}

object Driver {
  implicit val rng: RNG = ScalaRNG(1527608515939L)
  val DROP_BURN_IN = 100

  def genMapper[A, B](f: A => B): A => B = {
    val locker = com.twitter.chill.MeatLocker(f)
    x => locker.get.apply(x)
  }

  def average[l: List[Double]]: Double =
    l.size.toDouble / l.sum

  def dropBurnIn(dropBurn: Int)(v: List[Double]): List[Double] =
    v.drop(dropBurn)

  def fitPoisson(y: List[Int]): List[Double] = {
    val rate = for {
      ...
The Outline

Masquerading DNS Traffic

Time Series Modeling

Rainier + Spark

Anomaly Detection
Part 4 Window Based

Window 1

Window 2

Rainier

Simulated Parameter Values

Distribution Parameter Values

Difference
Part 4 Window Simulations
Part 4 Outlier Window
Part 4 Local Outlier to Global
Closing Recap

- Masquerading DNS Traffic
- Time Series Modeling
- Rainier + Spark
- Anomaly Detection
Closing Glossed Over Details

Outliers

Goodness of Fit
A Review of Croston's method for intermittent demand forecasting

Rainier
https://github.com/stripe/rainier

PyMC3
https://docs.pymc.io/

Emotet
https://www.us-cert.gov/ncas/alerts/TA18-201A

Bokeh Plots
https://bokeh.pydata.org/en/latest/

Twitter Chill
https://github.com/twitter/chill
Closing Contact

Website
davidrdgz.github.io

Github
@davidrdgz

Twitter
@davidrdgz

Email
davrodr3 at cisco.com