
CHIRON
By @rodsoto @josephzadeh

$whoami...

Rod Soto

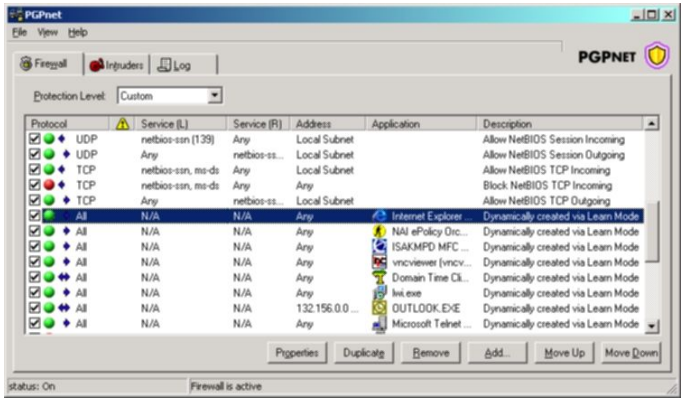
Director of Security Research at JASK.AI. Co-founder of Hack The Valley %27. Creator/Developer of Kommand & KonTroll CTF. Co-founder of Hackmiami. Previously at Prolexic, Caspida, AKAMAI, Splunk.

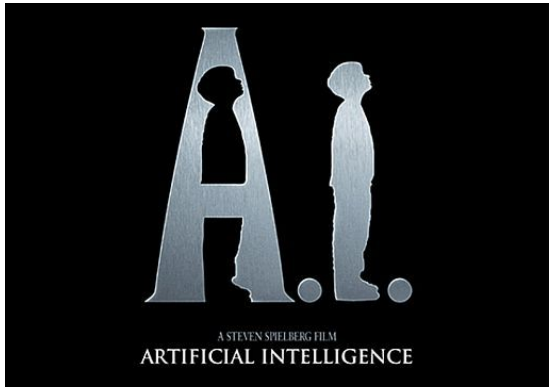
Joseph Zadeh

Director of Data Science at JASK.AI. Co-founder of Hack The Valley %27. Previously at Pivotal, Kaiser Permanente, Caspida, Splunk.

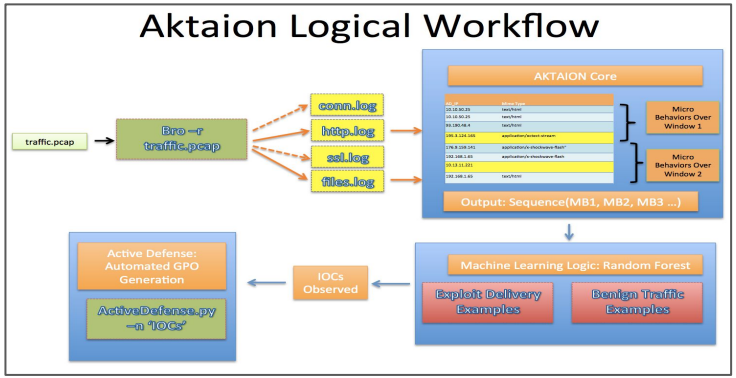
Introduction

ML + FW = Next Generation of Home Defense



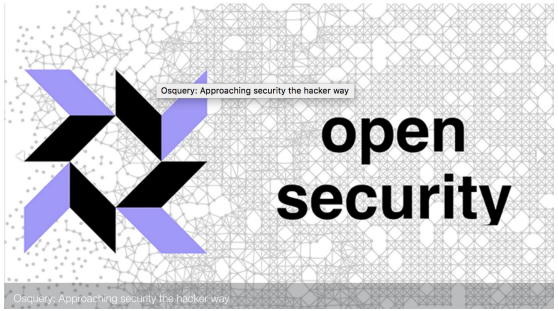


Motivation: Security Innovations are hard to scale down

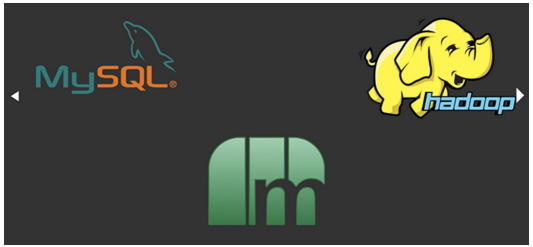


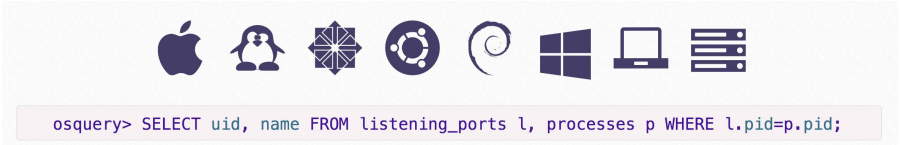
OpenSOC

Big Data Security Analytics Framework



Apache Spot





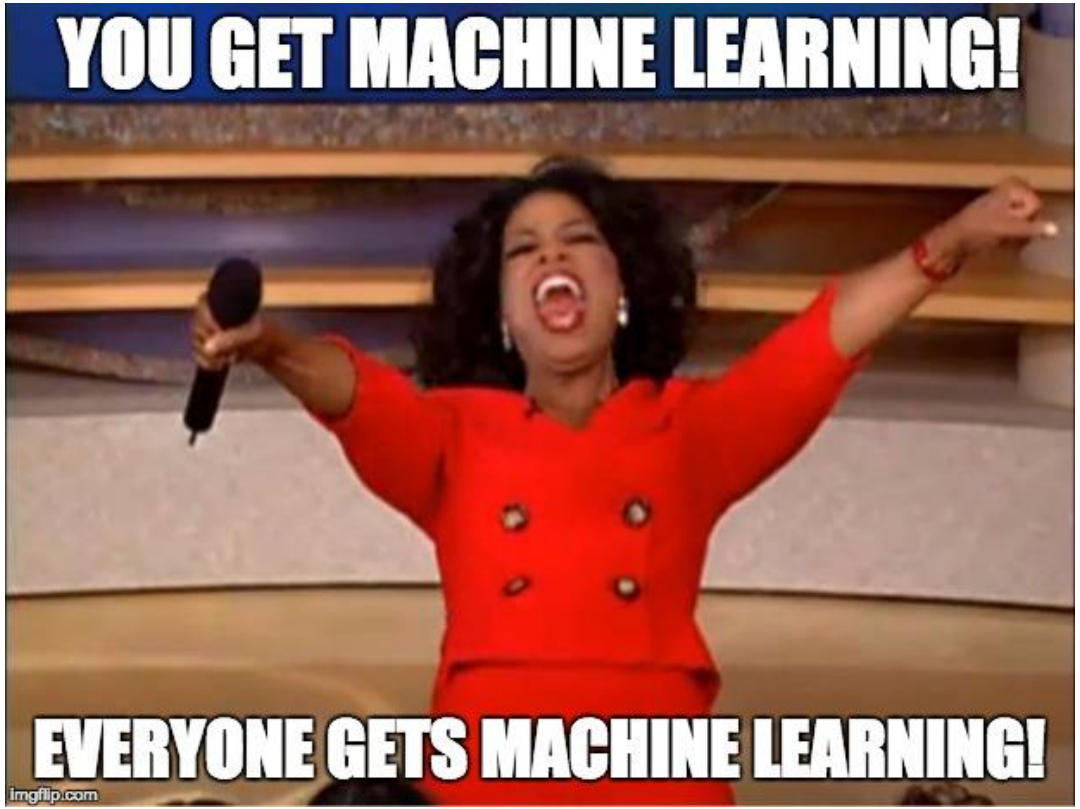
Challenges in home networking defense

- Internet of Things (toasters, refrigerators, thermostats, cameras, wearables, AI, routers, printers, etc)
- Unpatched vulnerable devices become Zombies of very large botnets feeding the crime ecosystem (CPEs are prime target)
- Common users have no idea or knowledge of what to do with all these “home” infrastructure becoming prey of snake oil pharisees
- Huge privacy risks, (I.E Unknown external IP accessing webcam or child monitor, somebody unlocking doors remotely, large data transfer from Home NAS to unknown IP address)

Challenges in home network defense II

- Wifi poachers, p2p piracy, someone using your home network as “VPN”
- Targeted attacks against your organization via your home network
- Targeted attacks against personal property (identity, finance, insurance)
- Port 25/110/143/993/465 running?. Is your home network being used for SPAM
- 9001/9030/9050/9150 ? Why is somebody using TOR at home?

Bringing Machine Learning to Home Defense



No longer a silicon valley, tech class thing

- Our effort is to bring a basic analytics tool to the masses. We believe this technology can help everyone, not just large enterprises.
- We tried to make it simple and easy to use.
- Machine learning can help everyone at home and in the SMB discover certain classes of threat actors
- Seeing is believing, these basic statistics will provide an overall yet sufficient picture to understand what is going on at your home network

Let’s be practical about what software automation can uncover. ML or any single point solution is not good for defending against advanced actors.

Advantages of ML

- Using ML allows us to put together simple analytics, on the main elements of home networking: First class citizens in our data model (Assets, Services, Protocols, Users, Traffic).
- ML allows us to go beyond of static signature based technologies. This aims to augment what we are good at so far and apply evidence fusion to point solution output like Bro/Snort alerts or iptable type log changes (osquery).
- IDS best practices + analytic algorithms help creates a scenario where detection of threats based on dynamic and multi contextual indicators is possible.

CHIRON - ML network home security use cases

- Malicious file downloads
- Fingerprints users, services, and protocols
- High risk users, assets, domains
- Data usage
- North - South traffic
- Relationships between assets
- Social Media usage
- Odd application/traffic/services

Bringing Machine Learning to home defense

- You don’t need a hadoop cluster at home in order to use machine learning
- There are several open source tools that we can use to have a decent proof of concept of network analytics plus machine learning
- No “SNAKE OIL” OPEN SOURCE github.com/jzadeh/chiron
- No need for a network tap, we are using passive detection tool pof in combination with NMAP. You can use a network tap and pipe the output if you want.
- It can be run from security onion
- FIRE AND FORGET NO NEED TO CODE

A case to bring ML to home defense

- Home networks are your castle
- Pivot of targeting/phishing employees at their homes (remote workers)
- IoT -- big leakers of private info, unpatch, vulnerable, significant risk
- Do you live in highly dense building? Anybody poaching your Internet service?
- Where all those devices connect to?
- Where all my users connecting to?
- Is there any suspicious NORTH-SOUTH traffic?
- Dynamic asset discovery

A case to bring ML to home defense

Three basic tasks:

- Perform basic discovery and analytics of home network assets (IoTs, Workstations, Laptops, Servers, routers)
- Fingerprints users, services, and protocols
- Applies analytics to users and devices (Average session length, Traffic, Visited sites)

The Tool

IDS Source Code: http://www.github.com/jzadeh/chiron-elk

VM ISO: https://drive.google.com/uc?id=0B8Goyrax_wtaNzAxM2ptdUZ6anc&export=download

Requirements

- Can be deployed on a VM - 60GB HD 1GB RAM
- Chiron is good for a home network 5 - 10 users
- 30 day log retention standard
- ELK for visualization and simple interactive analysis

Core Principles

- **Lambda defense**: Use of big data tech for distributed processing and analytics
- **Micro behavior**: Basic units within a exploit chain or threat
- **Deployment Model**: Target a specific home use demographic for ease of use detection of change point behaviors linked to threats.

The Lambda defense at home...

- Large organizations are currently attempting to scale intrusion detection by using big data distributed processing technologies (mainly Hadoop).
- CHIRON does not depend on a Hadoop infrastructure even though it does use some libraries part of the ecosystem. CHIRON is not meant to scale to 1000's of users however it is based on the same analytic principles and some of its ML frameworks
- By identifying, analyzing and providing key analytics for home network security, CHIRON gives home defense insights that is simply not available at this point without technical expertise and hardware.

Micro behaviors what is that?

- Basic units present within a context of a threat or exploit kill chain. In order to define Micro behaviors we need to label data and give it a value within a context of threat or exploit kill chain.
- List of Micro behaviors present in a ransomware attack: -
  - Payload delivery.
  - Focused on traffic to malicious sites and the related indicators when malicious code is served.
  - Including things such as URI entropy, redirects, domain generated by algorithms (DGAs), types and sequences of MIME content presented to victim during payload delivery.

Micro behaviors what is that? II

- Call back (Phone home) patterns, including user agent, URI strings, HTTP “GET” or “POST” requests, DNS queries, URI strings, frequency of call-backs, periodicity of connections.
- Covert Channel indicators, such as non HTTP traffic (HTTPS), and non DNS traffic present during such communications.
- Entropy in URI/URL (number of dots, special characters, URL crazy) - Whois (registrar with bad rep / geolocation / bulletproof hosting

What are Micro behaviors? III

- URL Shorteners (Tiny URL, bit.ly, etc)
- Spelling errors, passwords fields
- Presence of iframes, javascript, pagerank?, source of images - Presence of .exe .pdf .bat .ps1 .ps .bin .bat .jar .bin .zip

By analyzing these Micro behaviors within a context of exploitation and entity behavior we can develop methods for detection. Then train the models, then test detection.

Microbehavior : A definition to express behaviors associated to TTP's in Code

```scala
package com.aktaion.ml.behaviors

import scala.util.matching.Regex

class ExploitationUriBehaviors extends MicroBehaviorSet {
  val uriMaxPathDepth = MicroBehaviorData("MaxPathDepth", "Maximum path length of all URI's observed")
  val uriMinPathDepth = MicroBehaviorData("MinPathDepth", "Minimum path length of all URI's observed")
  val uriMaxLength = MicroBehaviorData("MaxUriLength", "Maximum length of all URI's observed")
  val uriMinLength = MicroBehaviorData("MinUriLength", "Minimum length of all URI's observed")
  val uriDistinct = MicroBehaviorData("UniqNumberOfUri", "Unique count of URI's in window")
  val uriMaxEntropy = MicroBehaviorData("UriMaxEntropy", "Maximum entropy of all URI's observed")
  val uriMinEntropy = MicroBehaviorData("UriMinEntropy", "Minimum entropy of all URI's observed")
  val base64Match = MicroBehaviorData("UriBase64", "URI's obseved contain large number of base 64 strings")
  val percentEncondingMatch = MicroBehaviorData("UriBase64", "URI's obseved contain large number of percent encoded strings")

  val behaviorVector = List(...)

  val encodingBase64 = new Regex( """^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]{3}=)?$""")
  val encodingBase64modified = new Regex( """^([A-Za-z0-9+/]{4})*([A-Za-z0-9+/]{4}|[A-Za-z0-9+/]{3}=|[A-Za-z0-9+/]{2}==)$""")
  val encodingPercentSimple = new Regex( """%[A-Fa-f0-9]{2}""")
}

class ExploitationTimingBehaviors extends MicroBehaviorSet {
  val maxTimeIntervalA = MicroBehaviorData("MaxTimeIntervalA", "Difference in timestamp between event 1 and event 2")
  val maxTimeIntervalB = MicroBehaviorData("MaxTimeIntervalB", "Difference in timestamp between event 2 and event 3")
  val maxTimeIntervalC = MicroBehaviorData("MaxTimeIntervalC", "Difference in timestamp between event 3 and event 4")
  val maxTimeIntervalD = MicroBehaviorData("MaxTimeIntervalD", "Difference in timestamp between event 4 and event 5")

  val minTimeIntervalA = MicroBehaviorData("MinTimeIntervalA", "Difference in timestamp between event 1 and event 2")
  val minTimeIntervalB = MicroBehaviorData("MinTimeIntervalB", "Difference in timestamp between event 2 and event 3")
  val minTimeIntervalC = MicroBehaviorData("MinTimeIntervalC", "Difference in timestamp between event 3 and event 4")
  val minTimeIntervalD = MicroBehaviorData("MinTimeIntervalD", "Difference in timestamp between event 4 and event 5")

  val intervalLength = MicroBehaviorData("TimeLengthInWindow", "Difference last and first time in the window")

  val ratioOfDeltasA = MicroBehaviorData("DeltaRatioA", "Count of time delta < 1 second over window size")
  val ratioOfDeltasB = MicroBehaviorData("DeltaRatioB", "Count of time delta < 5 second over window size")
  val ratioOfDeltasC = MicroBehaviorData("DeltaRatioC", "Count of time delta < 10 second over window size")
  val ratioOfDeltasD = MicroBehaviorData("DeltaRatioD", "Count of time delta < 20 second over window size")
  val ratioOfDeltasE = MicroBehaviorData("DeltaRatioE", "Count of time delta >= 100 second over window size")

  val behaviorVector = List(...)
}
```

CHIRON ARCHITECTURE

Scaling Op-Sec to the Home Environment

- There are tools for defense but there is a learning curve to deployment models see for instance “DEFCON 17: Asymmetric Defense: How to Fight Off the NSA Red Team with Five People or Less” https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-efstratios_gavas-nsa_red_team.pdf

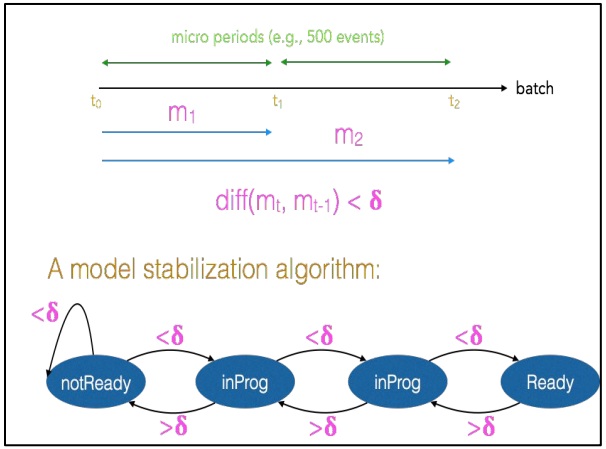
Some of the tools mentioned + bootstrapping FreeBSD

Monowall, IPCop, Untangle, pfSense

Goal is to make these type of techniques easier to adopt for non-power users

How Does CHIRON Learn?

1. Key insight is to take action on new behaviors observed on a small network segment/s
2. New Behavior means a change point in underlying data:
   a. New service interaction (IP1 ↠ IP2 via port /app never observed before)
   b. New DMZ interaction
   c. Exploit chain observed with new MIME Type
3. Exploit detection learning (standalone model based off Aktaion)



Adversary Modeling For The Home

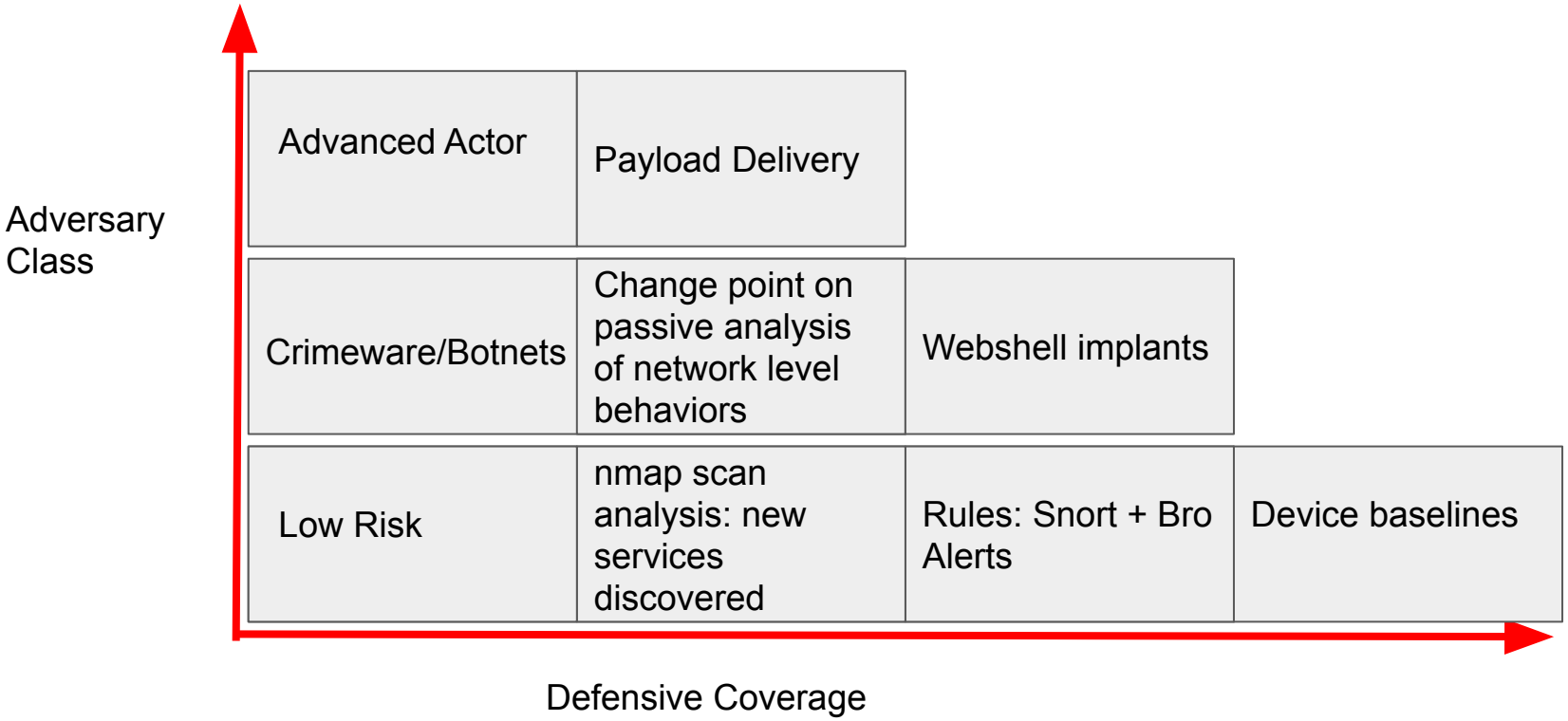
Key classes to model:

1. Crimeware / Ransomware
2. Opportunistic Neighbors/Local attacks
3. IoT/BYOD exploitation (android attacks)
4. Merai/Botnets

Key Use Cases to Targeted in V1:

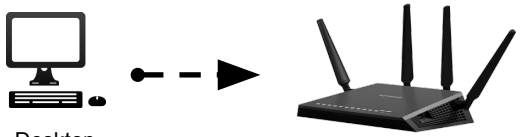
1. Delivery of payloads (weaponized code/RATs and implants on hosts)
2. IoT / Perimeter Device attacks

Adversary Modeling For The Home/SMB

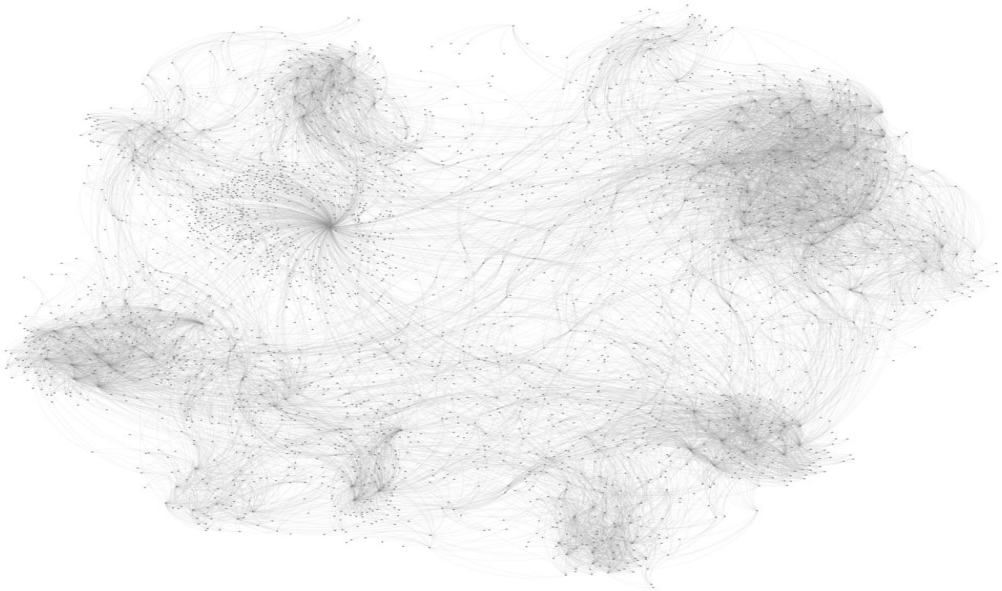


Graph Behaviors For Home Defense

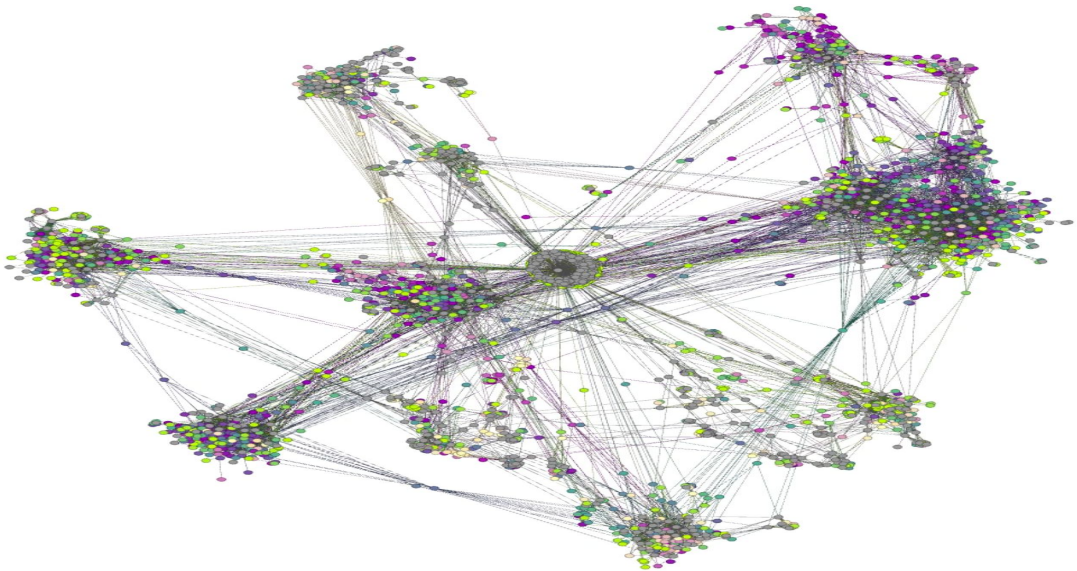
Key intuition: Anomalies on rare services and rare behaviors not in the 30 day log history



Seeing the Analytic In Action

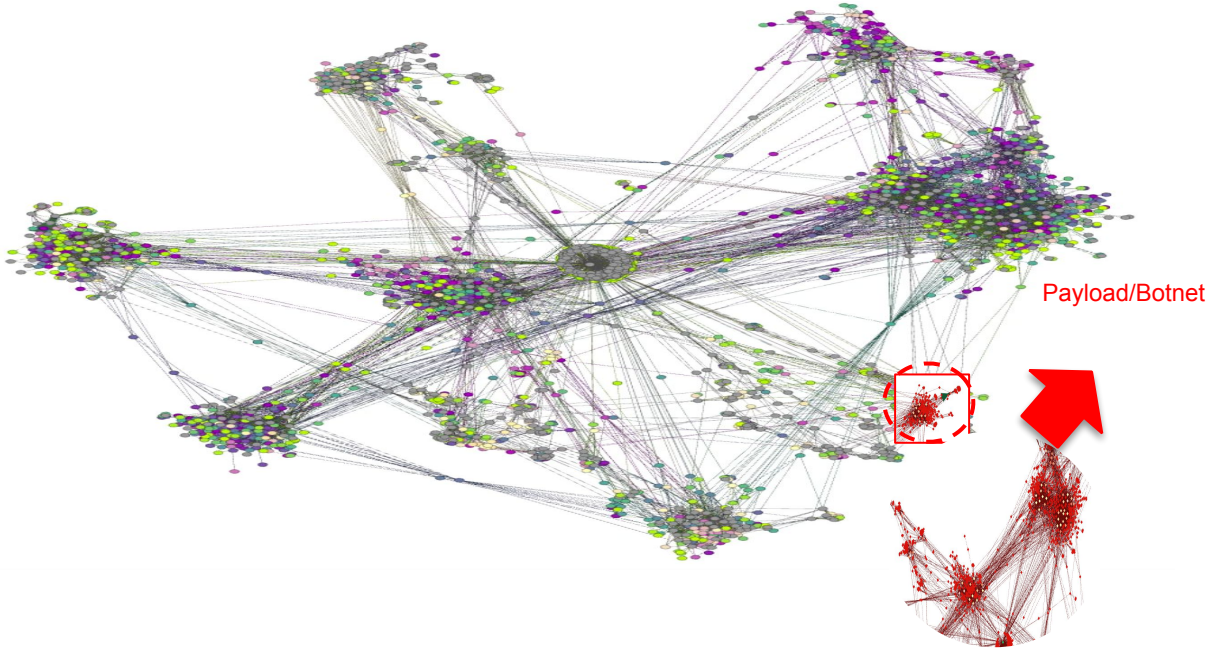


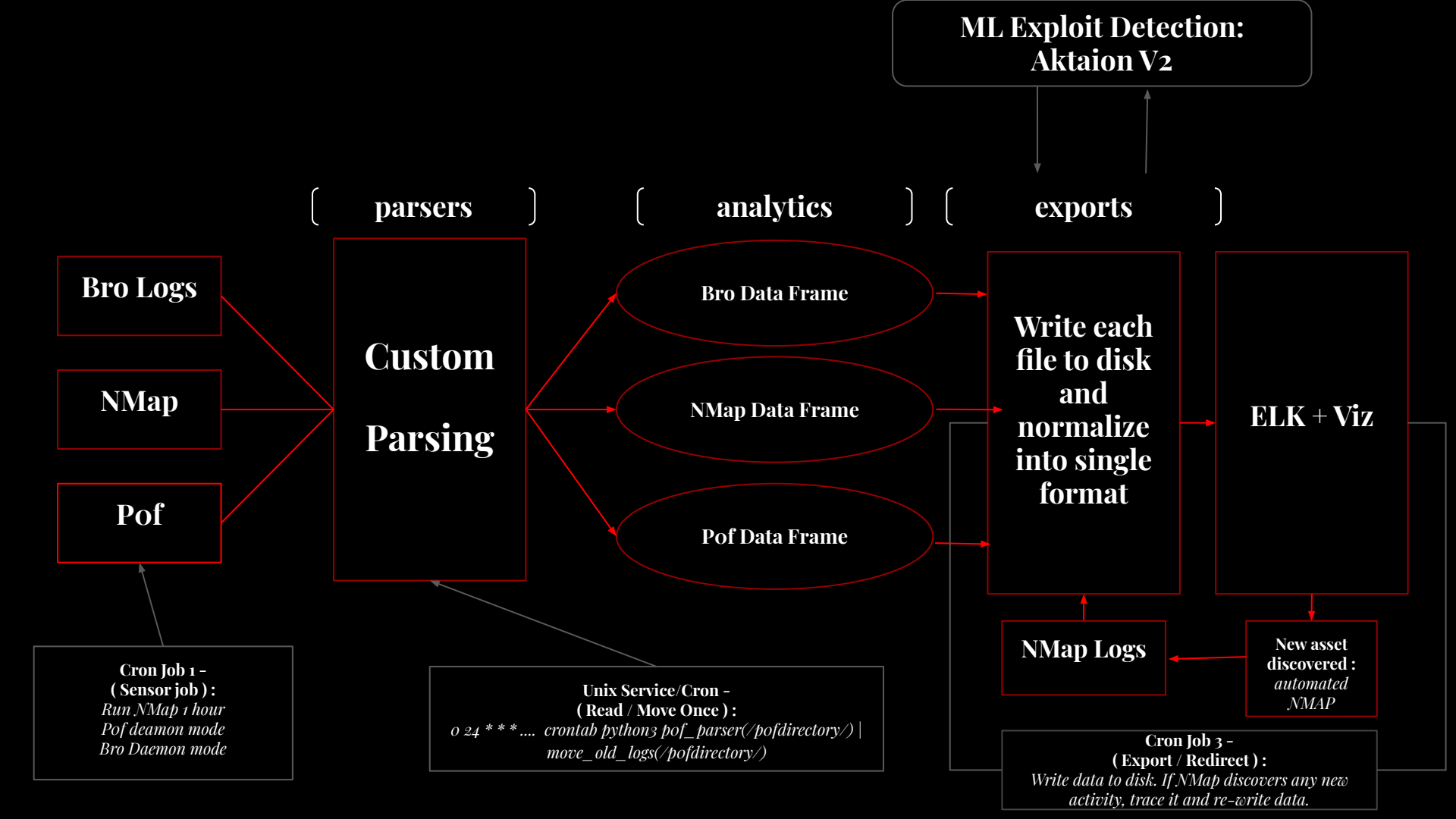
Seeing the Analytic In Action

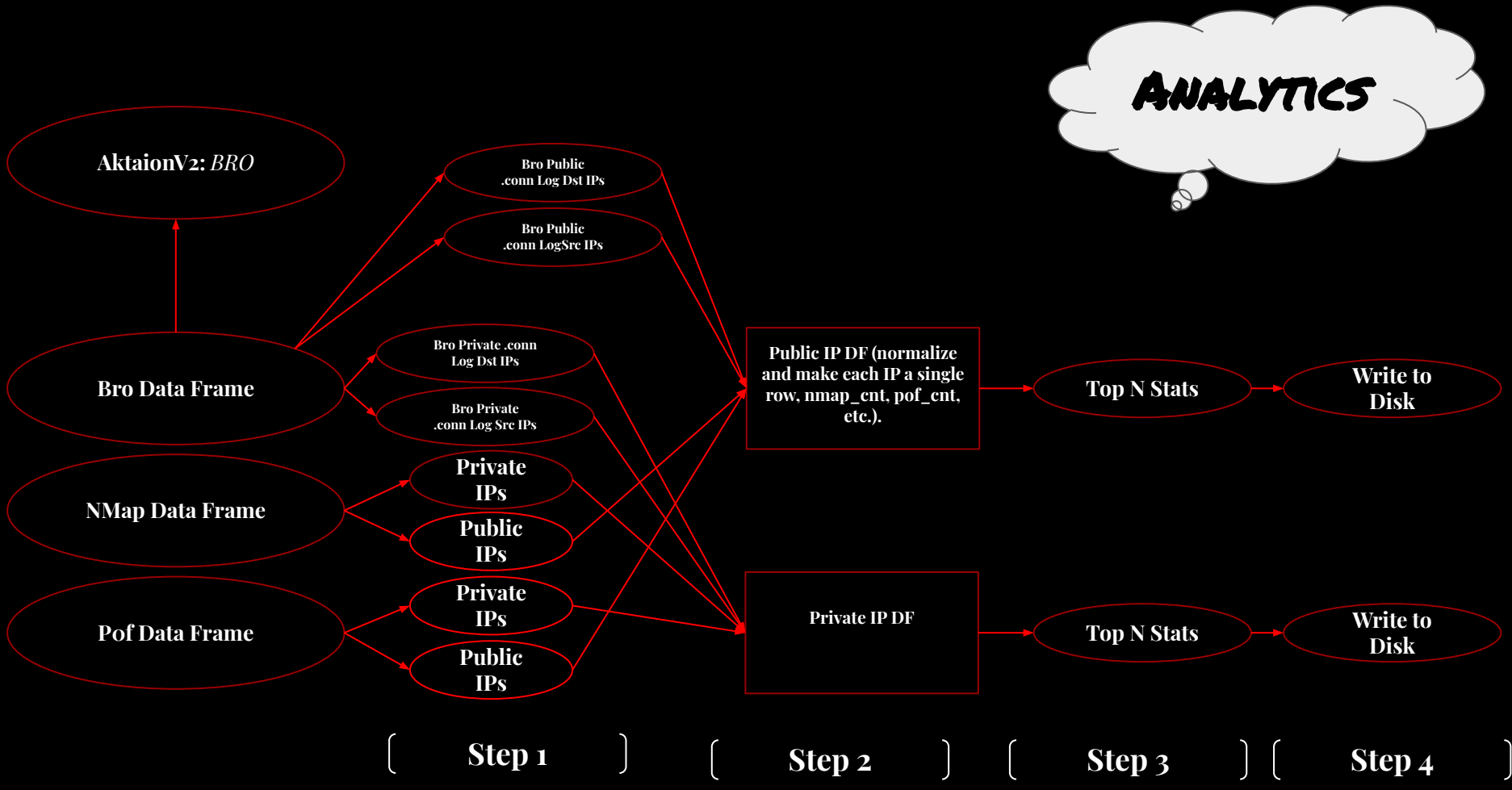


Seeing the Analytic In Action

Changepoint detection at the service/application layer




ML Exploit Detection:
Aktaion V2
parsers
analytics
exports
Bro Logs
NMap
Pof
Custom
Parsing
Bro Data Frame
NMap Data Frame
Pof Data Frame
Write each file to disk and normalize into single format
ELK + Viz
NMap Logs
New asset discovered : automated NMAP
Cron Job 1 -
(Sensor job) :
Run NMap 1 hour
Pof deamon mode
Bro Daemon mode
Unix Service/Cron -
(Read / Move Once) :
0 24 * * * .... crontab python3 pof_parser(/pofdirectory/) | move_old_logs(/pofdirectory/)
Cron Job 3 -
(Export / Redirect) :
Write data to disk. If NMap discovers any new activity, trace it and re-write data.


Analytics
AktaionV2: BRO
Bro Data Frame
NMap Data Frame
Pof Data Frame
Bro Public .conn Log Dst IPs
Bro Public .conn LogSrc IPs
Bro Private .conn Log Dst IPs
Bro Private .conn Log Src IPs
Private IPs
Public IPs
Private IPs
Public IPs
Public IP DF (normalize and make each IP a single row, nmap_cnt, pof_cnt, etc.).
Private IP DF
Top N Stats
Top N Stats
Write to Disk
Write to Disk
Step 1
Step 2
Step 3
Step 4

The Tool

IDS Source Code: http://www.github.com/jzadeh/chiron-elk

VM ISO: https://drive.google.com/uc?id=0B8G0yrax_wtaNzAxM2ptdUZ6anc&export=download

Requirements

- Can be deployed on a VM - 60GB HD 1GB RAM
- Chiron is good for a home network 5 - 10 users
- 30 day log retention standard

V2 of Architecture

Leverage OS query to do active defense interaction for targeting one specific use case: Given a network flow on the home Router what process on what host generated that flow.

Snort integration

Shadow IDS (active response disabled)

- Receive but no Transmit for a hardwired network tap

Community Involvement!

Exploit Detection ML: Aktaion

Phishing Attacks Had A Big Year In 2016, But Exploit Kits Will Make A Comeback In 2017

Spandas Lui
Feb 7, 2017, 12:30pm · Filed to: Australian Stories

Share

Image: iStock

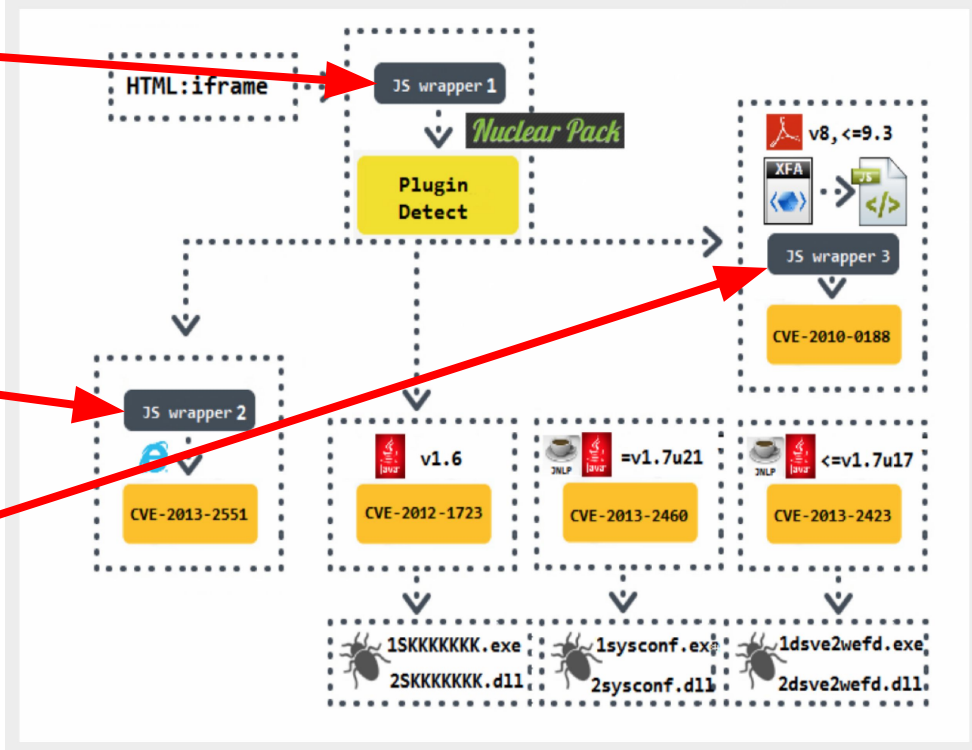
Last year saw a steep increase in malware distributed through email attachments in phishing attacks but 2017 will be the year of malware exploit kits, according to a report by security vendor Malwarebytes. We take a closer look at malware trends from 2016 and

Successful Exploit Chain Post Phish (Angler EK)

User clicks on email link

Exploit is looking for vulnerability

Exploit is successful ->payload delivered

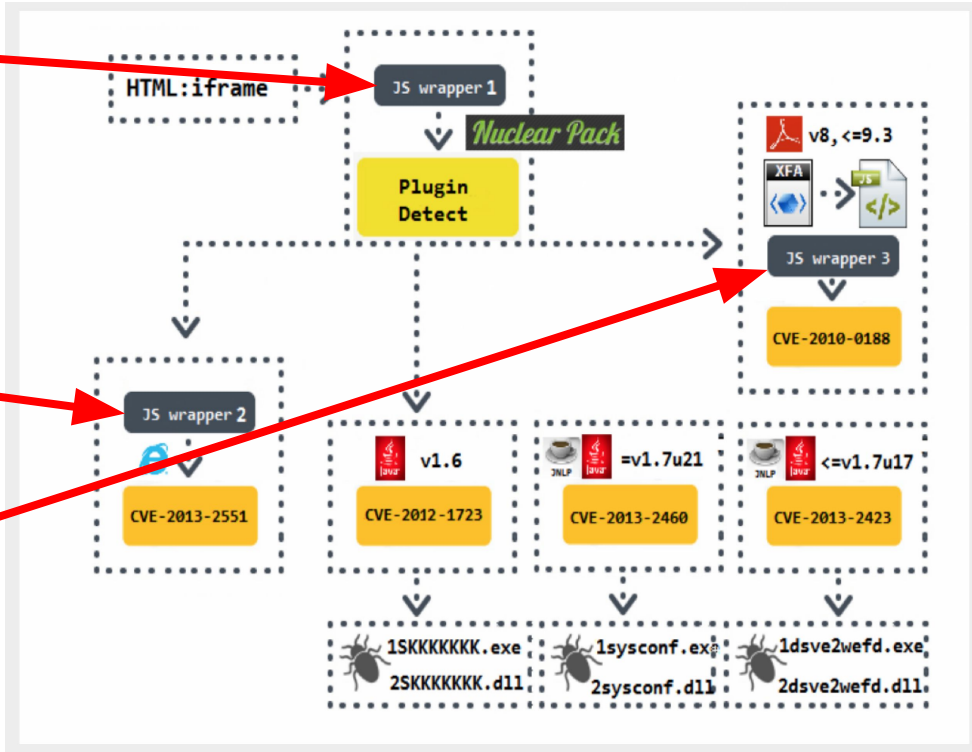


This is what it looks like in web proxy logs

1. **Initial Redirect From Poisoned Domain:** [29/Apr/2015:16:52:23 -0700] "Nico Rosberg" 192.168.122.177 69.162.78.253 1500 200 TCP_HIT "**GET http://forbes.com/gels-contrariness-domain-punchable/1.html/548828415920276748** HTTP/1.1" "Internet Services" "low risk" "**text/html**" 604 142 "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)" "http://forbes.com/gels-contrariness-domain-punchable/1.html" "-" "0" "" "-"

2. **Flash Exploit:** [29/Apr/2015:16:52:26 -0700] "Nico Rosberg" 192.168.122.177 69.162.78.253 1500 200 TCP_HIT "**GET http://portcullisesposturen.europartsplus.org/lMvOBBZKDLqAJYIDe02t5hMMNyzBLN_q4kafJkVNqJVTnTmd** HTTP/1.1" "Internet Services" "low risk" "**application/x-shockwave-flash**" 518 821 "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)" "http:///forbes.com/gels-contrariness-domain-punchable/1.html/548828415920276748" "-" "0" "" "-"

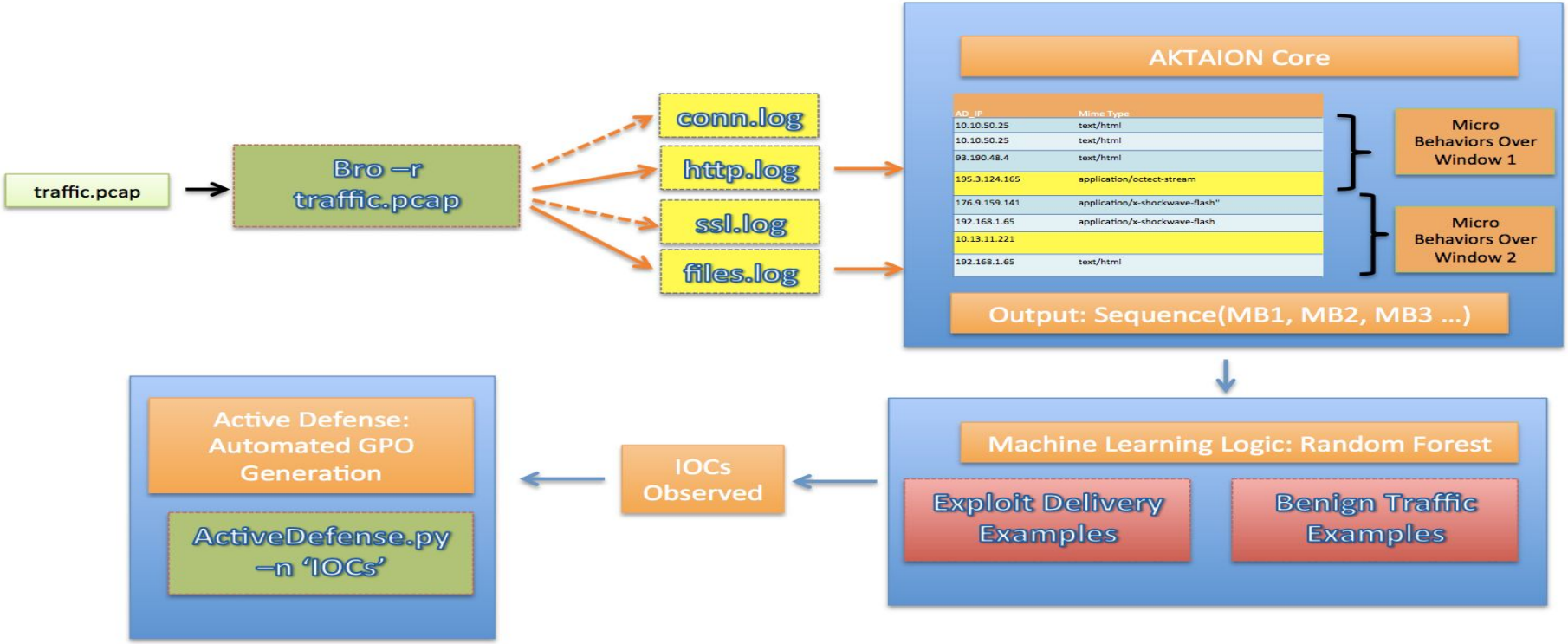
3. **Payload:** [29/Apr/2015:16:52:27 -0700] "Nico Rosberg" 192.168.122.177 69.162.78.253 1500 200 TCP_HIT "**GET http://portcullisesposturen.europartsplus.org/UX7n1YkbNn8FUV6QVtEZLj_p_gLvRKlWEWmz3r7Ug8suRiY_** HTTP/1.1" "Internet Services" "low risk" "**application/octet-stream**" 136 915 "" "" "-" "0" "" "-"



Workflow

1. Take PCAPs of known (labeled) exploits and known (labeled) benign behavior and convert them to bro format
2. Convert each Bro log to a sequence of micro behaviors (machine learning input)
3. Compare the sequence of micro behaviors to a set of known benign/malicious samples using a Random Forest Classifier (http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html)
4. Derive a list of indicators from any log predicted as malicious
5. Pass the list of IOCs (JSON) to a GPO generation script (https://github.com/jzadeh/Aktaion/tree/master/python)

Aktaion Logical Workflow



Learning = Compression?

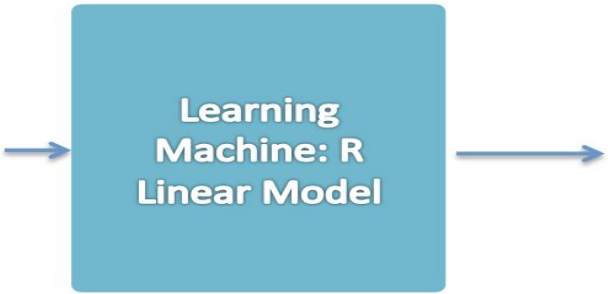
- Train a model to predict mpg as a function of car weight, number of cylinders and displacement

```
> mtcars
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
```

Learning = Compression?

- Train a model to predict mpg as a function of car weight, number of cylinders and displacement

```
> mtcars
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
```




```
> mfit = lm(mpg ~ wt + disp + cyl, data=mtcars)
```

Learning = Compression?

- The overall input data is reduced in a “compressed form” to use in future predictions

```
> mtcars
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
```

Learning Machine: R Linear Model

```
> (41.107678, -3.635677, 0.007473,-1.784944)
```

```
> mfit = lm(mpg ~ wt + disp + cyl, data=mtcars)
```

Security Analytics: The Lambda Defense

Operator Time is Valuable!

- Googles Experience with ML in Cybersecurity: https://web.stanford.edu/class/cs259d/lectures/Session11.pdf

Google

Why is statistical anomaly detection for security hard?

	Learning	Cost of error FP / FN	Goal	Attacker
Anomaly detection	Unsupervised	Medium / High	Classify & ***explain***	Adaptive
Spam detection	Supervised	High / Low	Classify	Adaptive
Product recommendation	Supervised	Low / Low	Classify	N/A

For more on this topic, consider "On Using Machine Learning For Network Intrusion Detection", Sommer and Paxson, Oakland 2010.

Google

An alternative: rule-based detection

Manually created rules

→ Characterize attacks or deviations, not normalcy

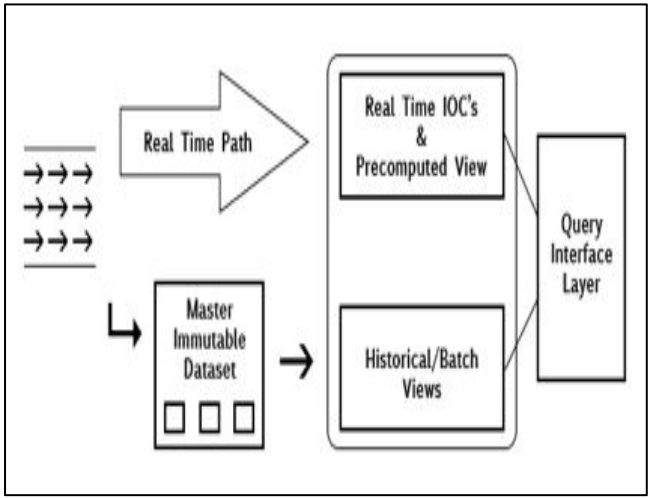
- Locality: each rule covers a small set of logs and features
- Explainability: direct connection between rules and alerts
- Specificity: make better use of analysts' expertise

Lambda Architecture

- The reference to the pattern comes from “Lambda Architecture” Coined by Nathan Mars
- GOTO 2012 • Runaway Complexity in Big Data Systems...and a Plan to Stop it • Nathan Marz: https://www.youtube.com/watch?v=ucHjyb6jv08

Architecture is described by three simple equations:

batch view = **function(all data)**
realtime view = **function(realtime view, new data)**
query = **function(batch view, realtime view)**



The Lambda Defense: Decomposing Behaviors for Intrusion Detection

DEFINITION 1. ***The Lambda Defense Pattern***

1. *$|S| = |\mathbb{R}|$: Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \dots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

3. *Functional Decomposition: For each TTP_i we model the behavior as $TTP_i := B_i + NB_i$. Where B_i is what we call the sequential component of the behavior and NB_i is called the non-sequential component. Formally speaking each TTP_i is actually a stochastic process (sequence of random variable) $TTP_i = \{TTP_i(t) : t \geq 0\}$.*

4. *Behavioral Model: For each sub component B_i, NB_i we assign one or more models $M_1(B_i), \dots, M_k(B_i), M_{k+1}(NB_i), M_{k+2}(NB_i), \dots$ and we evolve these models over time in the next step. We keep the definition of model intentionally vague here but in practice each model is usually machine learning based (supervised/semi-supervised/unsupervised) or heuristically engineered around a use case that is not driven by a pattern recognition problem. It is important to note in that we are looking for the right tool for the job and machine learning is sometimes only used as a pre-filtering step or not at all. In some cases we combine graph processing or techniques from signal analysis where it helps with overall problem of a specific model implementation.*

5. *Map/Reduce Based Model Life Cycle: We can update simultaneous models in a scalable way with a simple map and reduce operation given by a signature of the form [41]: ModelRDD.aggregate[M](seqOp: (M, Data) => M(Data), combOp: (M_{t_0},M_{t_1}) => M_{t_2})*

Step 1: Break the problem into use cases

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

Step 2: Find what use cases have highest security impact

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

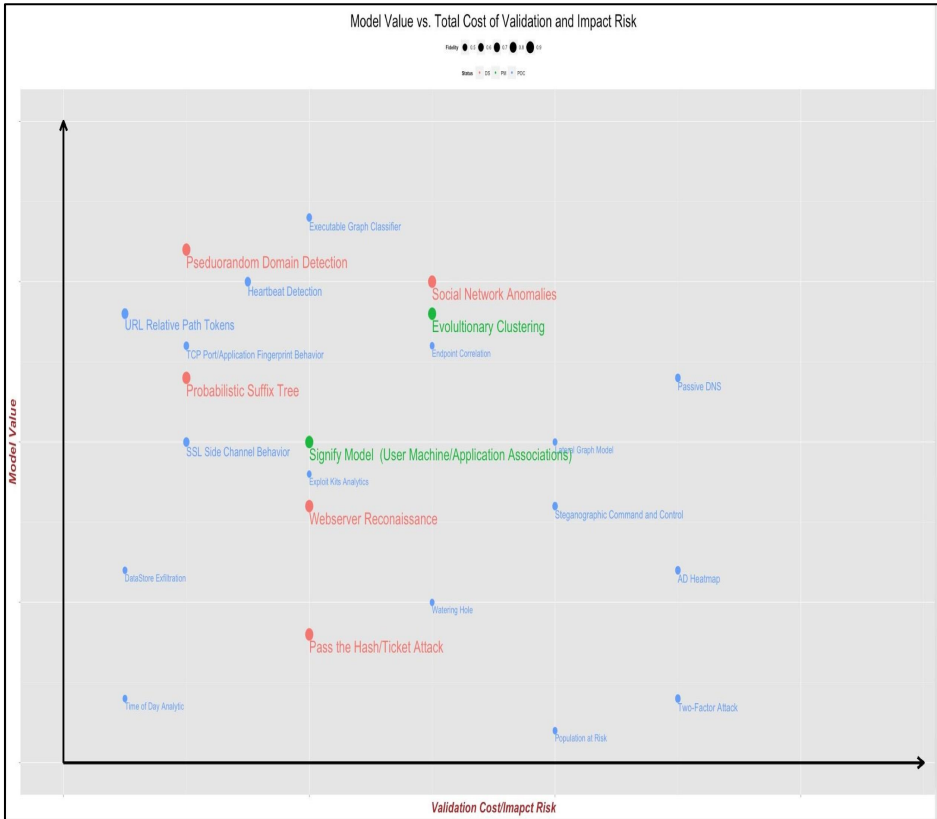
2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \ldots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

Step 2: Find what use cases have highest security impact

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \ldots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

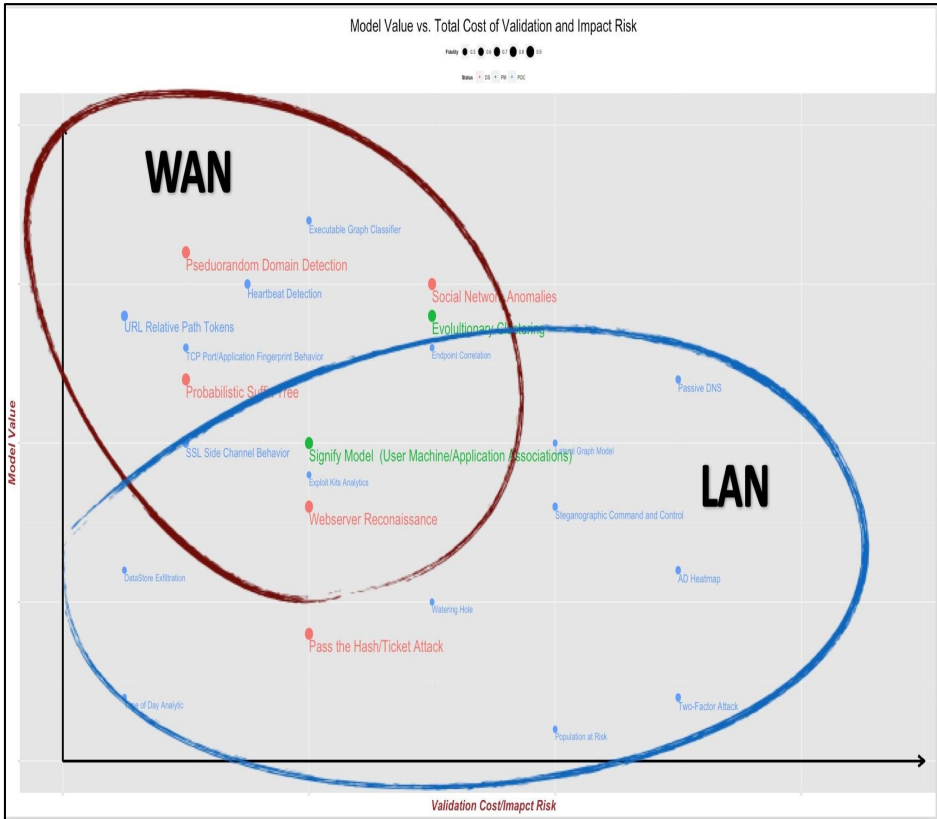


Step 2: Find what use cases have highest security impact

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \ldots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

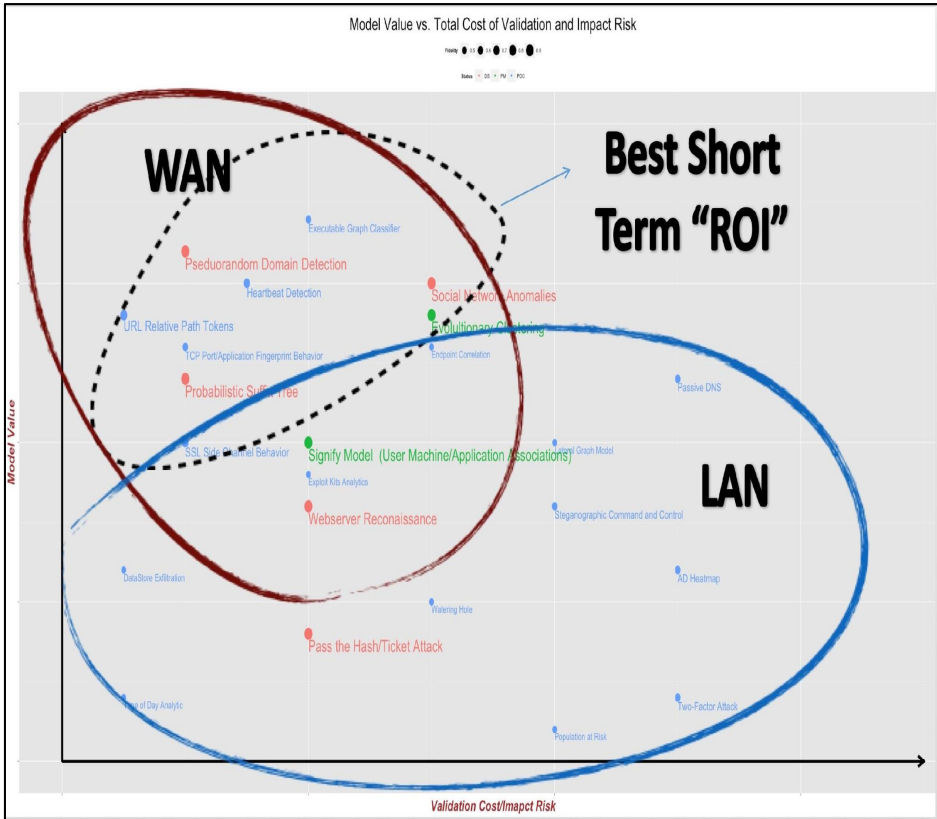


Step 2: Find what use cases have highest security impact

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \ldots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*



Step 3: Decompose the problem into two types of computation

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \dots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

3. *Functional Decomposition: For each TTP_i we model the behavior as $TTP_i := B_i + NB_i$. Where B_i is what we call the sequential component of the behavior and NB_i is called the non-sequential component. Formally speaking each TTP_i is actually a stochastic process (sequence of random variable) $TTP_i = \{TTP_i(t) : t \geq 0\}$.*

Step 3: Decompose the problem into two types of computation

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \ldots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

3. *Functional Decomposition: For each TTP_i we model the behavior as $TTP_i := B_i + NB_i$. Where B_i is what we call the sequential component of the behavior and NB_i is called the non-sequential component. Formally speaking each TTP_i is actually a stochastic process (sequence of random variable) $TTP_i = \{TTP_i(t) : t \geq 0\}$.*

Arbitrary User Behavior = Sequential Component + "Un-Ordered" Component

Step 3: Decompose the problem into two types of computation

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \ldots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

3. *Functional Decomposition: For each TTP_i we model the behavior as $TTP_i := B_i + NB_i$. Where B_i is what we call the sequential component of the behavior and NB_i is called the non-sequential component. Formally speaking each TTP_i is actually a stochastic process (sequence of random variable) $TTP_i = \{TTP_i(t) : t \geq 0\}$.*

Arbitrary User Behavior = Sequential Component + "Un-Ordered" Component

Examples

- Sequential Behaviors
  1. Exploit Chains
  2. Timing Analysis (Periodicity)
  3. Active Directory Sequence
  4. Authentication Graph
- Non Sequential Behaviors
  1. Fingerprinting
  2. Grouping Behaviors
  3. Application Counts
  4. Rare file extension counts for Webshell detection

Step 3: Decompose the problem into two types of computation

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \ldots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

3. *Functional Decomposition: For each TTP_i we model the behavior as $TTP_i := B_i + NB_i$. Where B_i is what we call the sequential component of the behavior and NB_i is called the non-sequential component. Formally speaking each TTP_i is actually a stochastic process (sequence of random variable) $TTP_i = \{TTP_i(t) : t \geq 0\}$.*

Arbitrary User Behavior = Sequential Component + "Un-Ordered" Component

Mapping Behaviors to Computational Paths

- Easy to Parallelize
  1. Count()
  2. Average()
  3. Time series()
  4. Local state computations
     - Per user/IP/account/…
- Hard to Parallelize (NC Complete Complexity)
  1. Rank()
  2. Median
  3. Anything that keeps track of global state
  4. Machine Learning Computations
     - Matrix Inversion or
     - Stochastic Gradient Descent

Step 3: Optimizing the use of technology for problem complexity

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \ldots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

3. *Functional Decomposition: For each TTP_i we model the behavior as $TTP_i := B_i + NB_i$. Where B_i is what we call the sequential component of the behavior and NB_i is called the non-sequential component. Formally speaking each TTP_i is actually a stochastic process (sequence of random variable) $TTP_i = \{TTP_i(t) : t \geq 0\}$.*

In complexity theory, the notion of **P-complete** decision problems is useful in the analysis of both:

1. which problems are difficult to parallelize effectively, and;
2. which problems are difficult to solve in limited space.

Arbitrary User Behavior = Sequential Component + "Un-Ordered" Component

Mapping Behaviors to Computational Paths

- Easy to Parallelize
  1. Count()
  2. Average()
  3. Time series()
  4. Local state computations
     - Per user/IP/account/…
- Hard to Parallelize (NC Complete Complexity)
  1. Rank()
  2. Median
  3. Anything that keeps track of global state
  4. Machine Learning Computations
     - Matrix Inversion or
     - Stochastic Gradient Descent

Step 4: Build an ML Model for each important sub-behavior

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \dots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

3. *Functional Decomposition: For each TTP_i we model the behavior as $TTP_i := B_i + NB_i$. Where B_i is what we call the sequential component of the behavior and NB_i is called the non-sequential component. Formally speaking each TTP_i is actually a stochastic process (sequence of random variable) $TTP_i = \{TTP_i(t) : t \geq 0\}$.*

4. *Behavioral Model: For each sub component B_i, NB_i we assign one or more models $M_1(B_i), \dots, M_k(B_i), M_{k+1}(NB_i), M_{k+2}(NB_i), \dots$ and we evolve these models over time in the next step. We keep the definition of model intentionally vague here but in practice each model is usually machine learning based (supervised/semi-supervised/unsupervised) or heuristically engineered around a use case that is not driven by a pattern recognition problem. It is important to note in that we are looking for the right tool for the job and machine learning is sometimes only used as a pre-filtering step or not at all. In some cases we combine graph processing or techniques from signal analysis where it helps with overall problem of a specific model implementation.*

Step 4: Build an ML Model for each important sub-behavior

DEFINITION 1. ***The Lambda Defense Pattern***

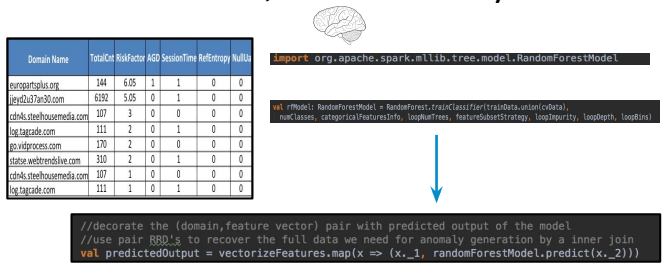
1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \dots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

3. *Functional Decomposition: For each TTP_i we model the behavior as $TTP_i := B_i + NB_i$. Where B_i is what we call the sequential component of the behavior and NB_i is called the non-sequential component. Formally speaking each TTP_i is actually a stochastic process (sequence of random variable) $TTP_i = \{TTP_i(t) : t \geq 0\}$.*

4. *Behavioral Model: For each sub component B_i, NB_i we assign one or more models $M_1(B_i), \dots, M_k(B_i), M_{k+1}(NB_i), M_{k+2}(NB_i), \dots$ and we evolve these models over time in the next step. We keep the definition of model intentionally vague here but in practice each model is usually machine learning based (supervised/semi-supervised/unsupervised) or heuristically engineered around a use case that is not driven by a pattern recognition problem. It is important to note in that we are looking for the right tool for the job and machine learning is sometimes only used as a pre-filtering step or not at all. In some cases we combine graph processing or techniques from signal analysis where it helps with overall problem of a specific model implementation.*

Each Model can be batch, real-time or hybrid mode



Step 5: Operationalize the Model Life Cycle

DEFINITION 1. ***The Lambda Defense Pattern***

1. $|S| = |\mathbb{R}|$: *Start with the threat surface S for a fixed adversarial environment. (The examples in this paper S will usually be the threat surface associated to a large enterprise network). The space of possible behaviors is a continuum at this point.*

2. *Enumerate the threat surface: How do we reduce the space of possible behaviors to actionable units that we can process in an upstream functional workflow? To answer this question we project down to modeling only a finite set $S \approx \{TTP_1, \dots TTP_n\}$. For a given threat surface the art of the initial design is to identify the core attack patterns needed to be modeled and to anticipate which of those parts are most likely to "drift" [19]. Better yet assign metrics to this finite approximation of the threat surface (see figure 1).*

3. *Functional Decomposition: For each TTP_i we model the behavior as $TTP_i := B_i + NB_i$. Where B_i is what we call the sequential component of the behavior and NB_i is called the non-sequential component. Formally speaking each TTP_i is actually a stochastic process (sequence of random variable) $TTP_i = \{TTP_i(t) : t \geq 0\}$.*

4. *Behavioral Model: For each sub component B_i, NB_i we assign one or more models $M_1(B_i), \dots, M_k(B_i), M_{k+1}(NB_i), M_{k+2}(NB_i), \dots$ and we evolve these models over time in the next step. We keep the definition of model intentionally vague here but in practice each model is usually machine learning based (supervised/semi-supervised/unsupervised) or heuristically engineered around a use case that is not driven by a pattern recognition problem. It is important to note in that we are looking for the right tool for the job and machine learning is sometimes only used as a pre-filtering step or not at all. In some cases we combine graph processing or techniques from signal analysis where it helps with overall problem of a specific model implementation.*

5. *Map/Reduce Based Model Life Cycle: We can update simultaneous models in a scalable way with a simple map and reduce operation given by a signature of the form [41]: ModelRDD.aggregate[M](seqOp: (M, Data) => M(Data), combOp: (M_{t_0},M_{t_1}) => M_{t_2})*

Q+A

The Tool

IDS Source Code: http://www.github.com/jzadeh/chiron-elk

VM ISO: https://drive.google.com/uc?id=0B8G0yrax_wtaNzAxM2ptdUZ6anc&export=download

Requirements

- Can be deployed on a VM - 60GB HD 1GB RAM
- Chiron is good for a home network 5 - 10 users
- 30 day log retention standard