



Anomaly detection, graph databases and NetFlows – from SECOR project perspective

Maciej Miłostan
45thTF-CSIRT meeting Poznań, 22nd May 2015

Work supported by the National Center for Research and Development of Poland (NCBiR) grant
no. PBS1/A3/14/2012 SECOR

Outline

- SECOR – beyond events correlation
- Architecture
- Anomalies
- Detection components
- NetFlow/IPFIX and graphs
- Graph DBs
- Complex Event Processor
- Future plans

SECOR – Facts sheet

- **Project name:** Security data correlation module for recognition of unauthorized activities and aiding decision processs
- **Code word:** SECOR
- **Source of funding:**
 - The National Center for Research and Development (NCBiR)
 - Applied Research Programme path A (support in scientific field)
- **Partners:**
 - Military Communication Institute (MCI)
 - PSNC
 - ITTI
- **Project duration:**
 - 30 months (01.12.2012 r. – 31.05.2015 r.)

Motivation

- Quite obvious 😊
- Rising a level of e-protection of e-infrastructures for e-science (including grid environments and NREN facilities)
- **Going beyond signature based systems to address emerging threats**
- Awareness building and early warning capabilities
- **Discovering complex events in large heterogenous data streams**
- Need for multi-level protection

Goal

- Implementation of SECOR prototype including:
 - **Novel detection methods** covering various levels of IT environment and based on:
 - Advanced statistics
 - Machine learning
 - Petri-Nets and ontologies
- Correlation of detection results coming from wide range of methods / **correlation of aggregated and raw symptoms** of attack or threat
- Visualization
- Promoting open standards for exchange of events



System architecture

Sensors

CEP + OSGi

Correlation & aggregation

Reasoning

Ontology

BA1

Behavioral analysis,
PetriNets

BA2

Machine learning

BA3

Statistical methods

Database

Graphical User
Interface

Mitigation

Anomaly and anomaly detection

- Definition of an anomaly
 - Dictionary: something that is unusual or unexpected
- Anomaly detection
 - Behavior anomaly detection
 - Identification of strange events (outliers) or trends that are not consistent with previous observations or theoretical model
 - Discovery of possible threats among anomalies (not all anomalies are threats and vice versa)
- We need events and/or model to detect anomaly

Event sources

- Main categories of event sources
 - System level
 - Service level
 - Network level
- Modules of SECOR analysis blocks work on different levels, but altogether cover all three levels

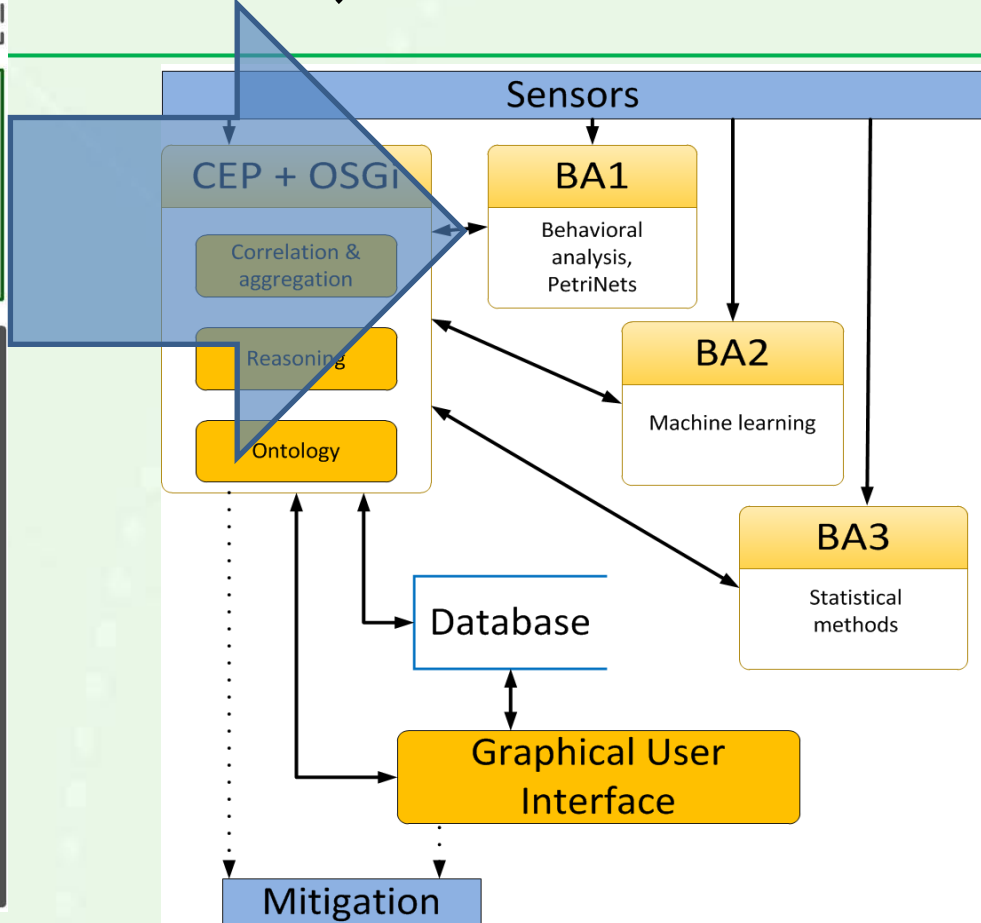
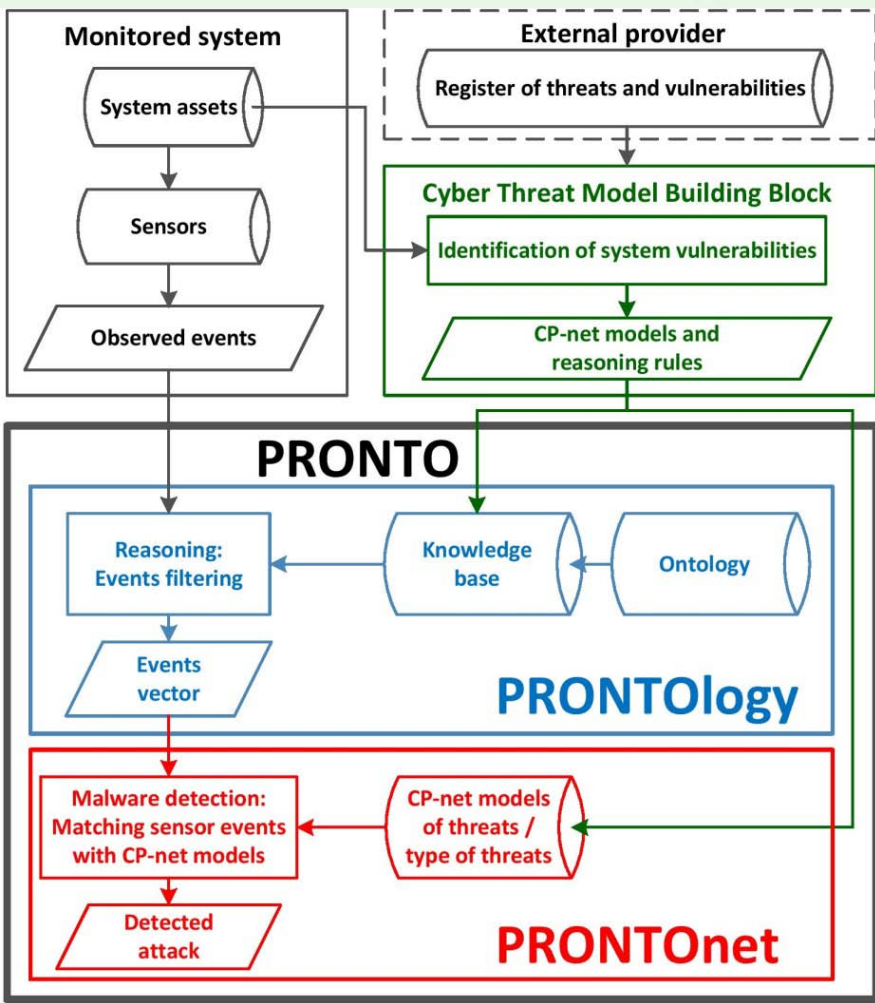


System level

Events on system level

- Windows
 - Microsoft API Hooks / Detours e.g. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms632589%28v=vs.85%29.aspx>
<http://research.microsoft.com/en-us/projects/detours/>
 - System events
- Linux
 - Library calls (ltrace)
 - System calls (syscalls)
 - Kernel messages (e.g. dmesg)
- We have separate modules for Linux and Windows

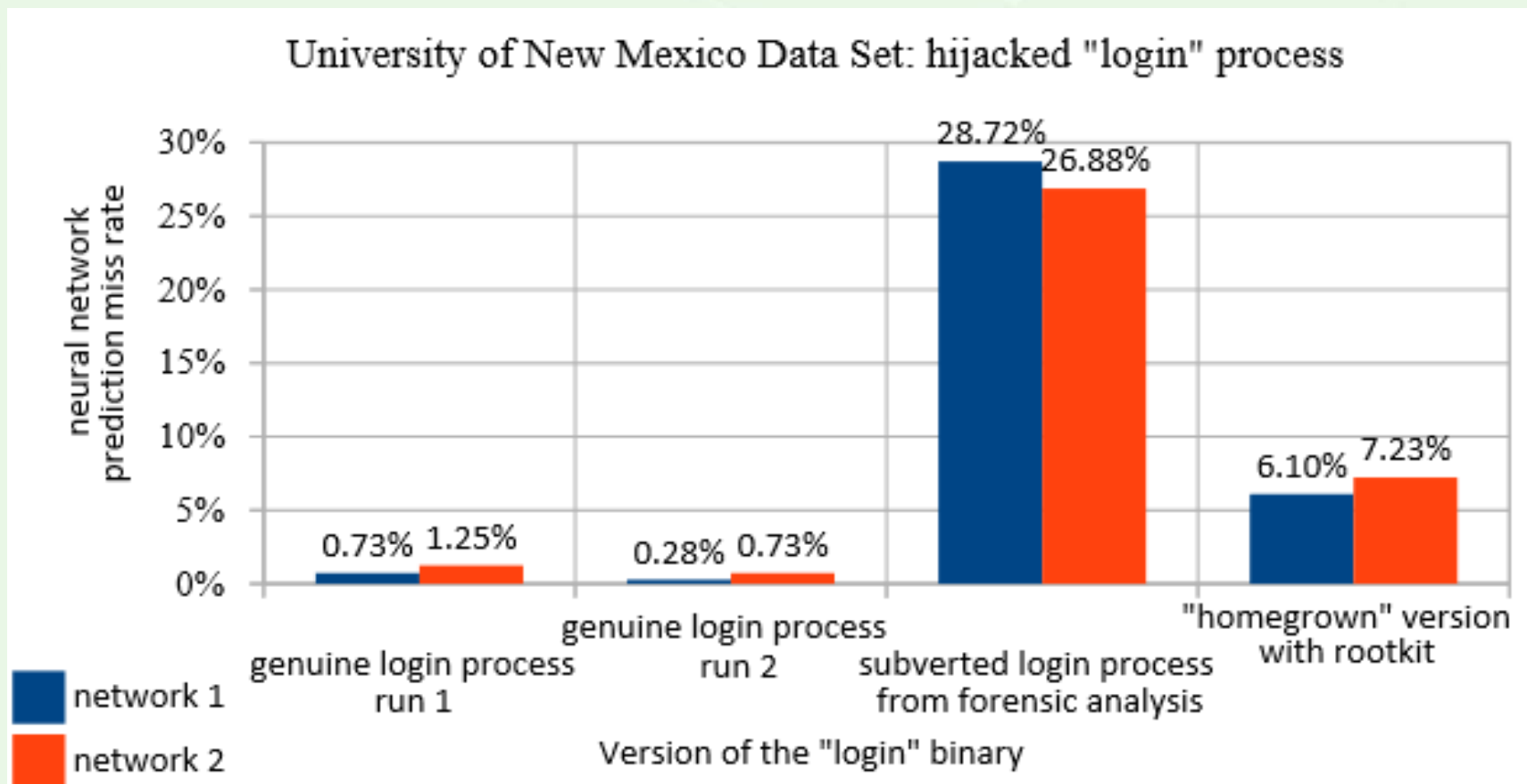
Behavioral analysis – PRONTO (MCI, Bartosz Jasiul)



EVENTS ARE GATHERED USING API HOOKS IN WINDOWS 7

Neural Networks

(PSNC, Tomasz Nowak, linux library calls)



Methods based on analysis of system/library/API calls on OS level

- Pros
 - Detection of unknown attacks based on knowledge about behavior of known malicious code that possibly could be reused in new malware
 - Detection of exploits activities on the basis of change in system and library calls patterns
 - Low number of false-positive alarms (errors)
- Cons
 - Complex implementation
 - Time-consuming detection
 - Necessity of continuous maintenance of models (new models are needed for new *malware* or new/updated executable)



Service level

Events on service level

- Let's focus on web-based applications:
 - Application logs
 - Web server logs
 - Access logs
 - Database logs
- Within SECOR, ITTI implemented a set of machine learning algorithms for SQL-injection attacks.



Network level

Events on network level

- Ethernet layer (e.g. port statistics, MAC addresses, Ethernet flow etc.)
- IP layer
 - NetFlows
 - IPFIX
- Within SECOR, we use NetFlows and/or IPFIX (depending on device or software probe capabilities)

NetFlows and IPFIX

Few remarks

- Potential problems with NetFlows
 - Most probes generate unidirectional NetFlow record, thus to get bi-directional traffic we have to aggregate them
 - Problem to distinguish source and target (problem with getting right flow direction) – this is in most cases true for TCP and even in more cases for UDP
- Identification of correct direction of information flow is crucial for successful application of network (communication) graphs
- IPFIX support bi-directional flows, so better use it if you can (in case of software probes for example use yaf instead of fprobe)
- Beware of dump delays

NetFlows

- Pros
 - Widely deployed – many devices support it
 - Distributed vSwitches in VMWare vSphere Enterprise Plus support it as well
 - Cisco Nexus 1000V can be used in Microsoft Hyper-V

NetFlows in SECOR

- Statistical methods
 - Analysis of entropies
- Graph theory based methods
 - Analysis of communication graph properties



Statistical methods

(MCI, Przemysław Bereziński)

Statistical methods

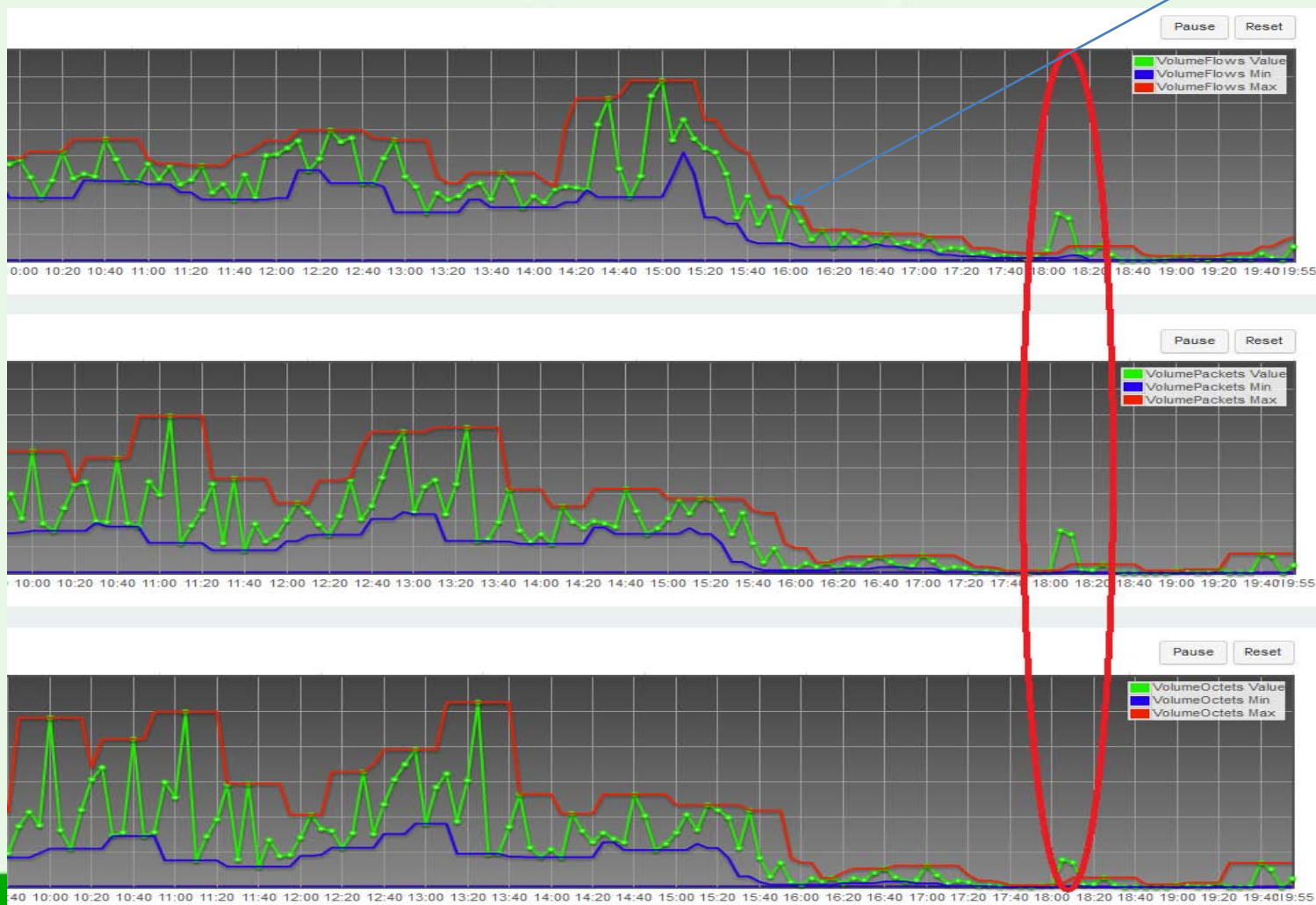
- Input
 - Full NetFlow dumps every 5 minute
 - Model received from observation of normal traffic
- Considered features
 - source /destination addresses and ports,
 - flows durations
 - transferred packets/bytes
 - in (out) -degree
- Mass functions for features
- Entropy measure
 - Tsallis
 - Tsallis with normalization
 - Renyi
 - Shannon

Example: Attack and detection

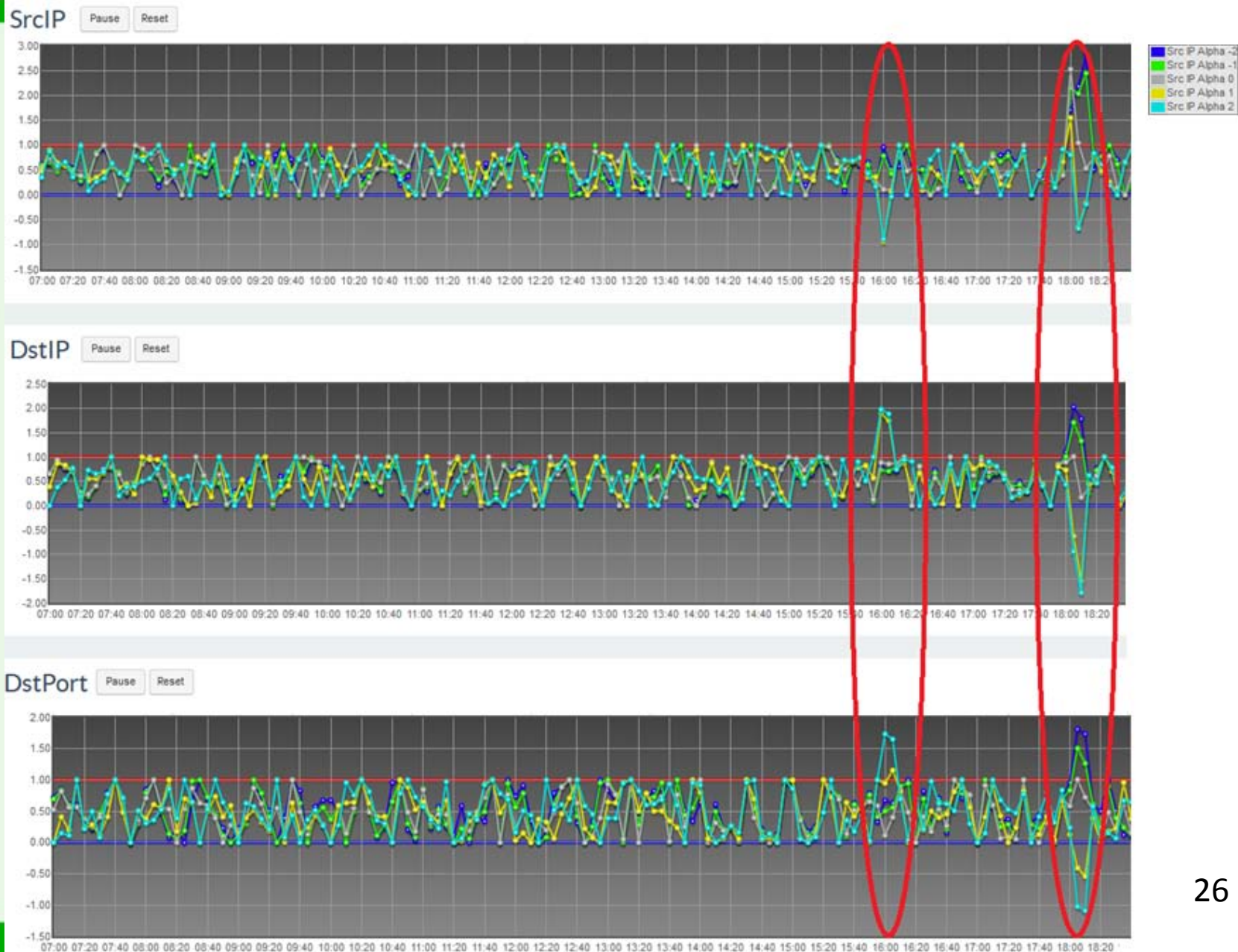
- Stage I
 - 15:55 – botnet malware infects single host
(can be detected on host level)
- Stage II
 - 16:00 – infected host begins network scan and exploits vulnerabilities (can be detected on network level)
 - 16:05 – more hosts get infected (detection on host system level)
- Stage III
 - 18:00 – malware communicates Command and Control (C&C)
(can be detected on network level)
 - 18:05 – malware take part in DNS reflected DoS attack
(can be detected on network level)

Analysis of traffic volume

New observation point is generated every 5 minutes



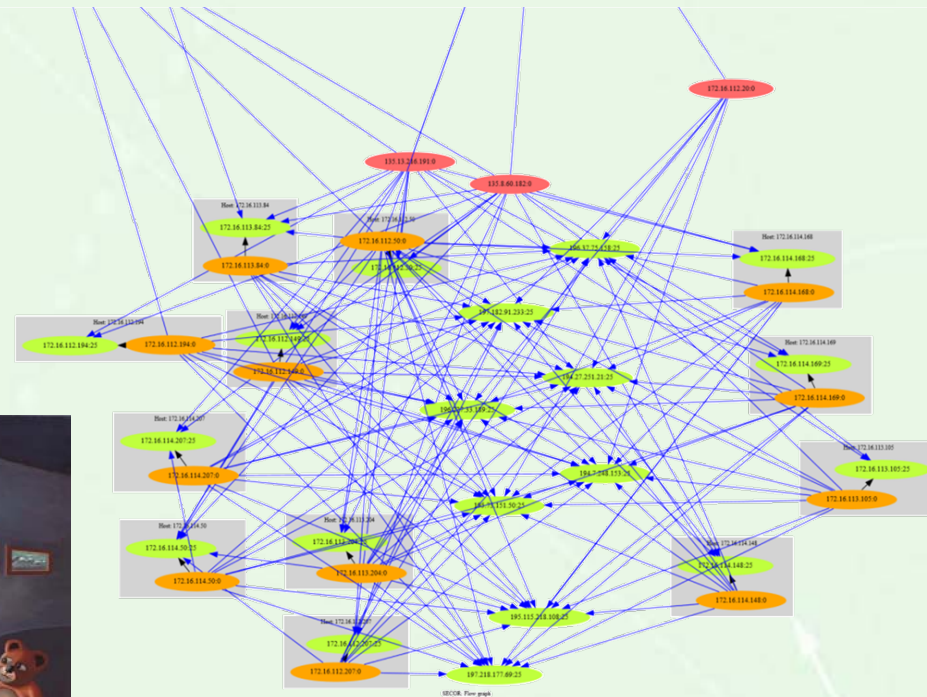
ENTROPY





Graph based methods (PSNC, Maciej Miłostan)

- Social network of communicating hosts
 - Who (what) communicates with whom (or what)



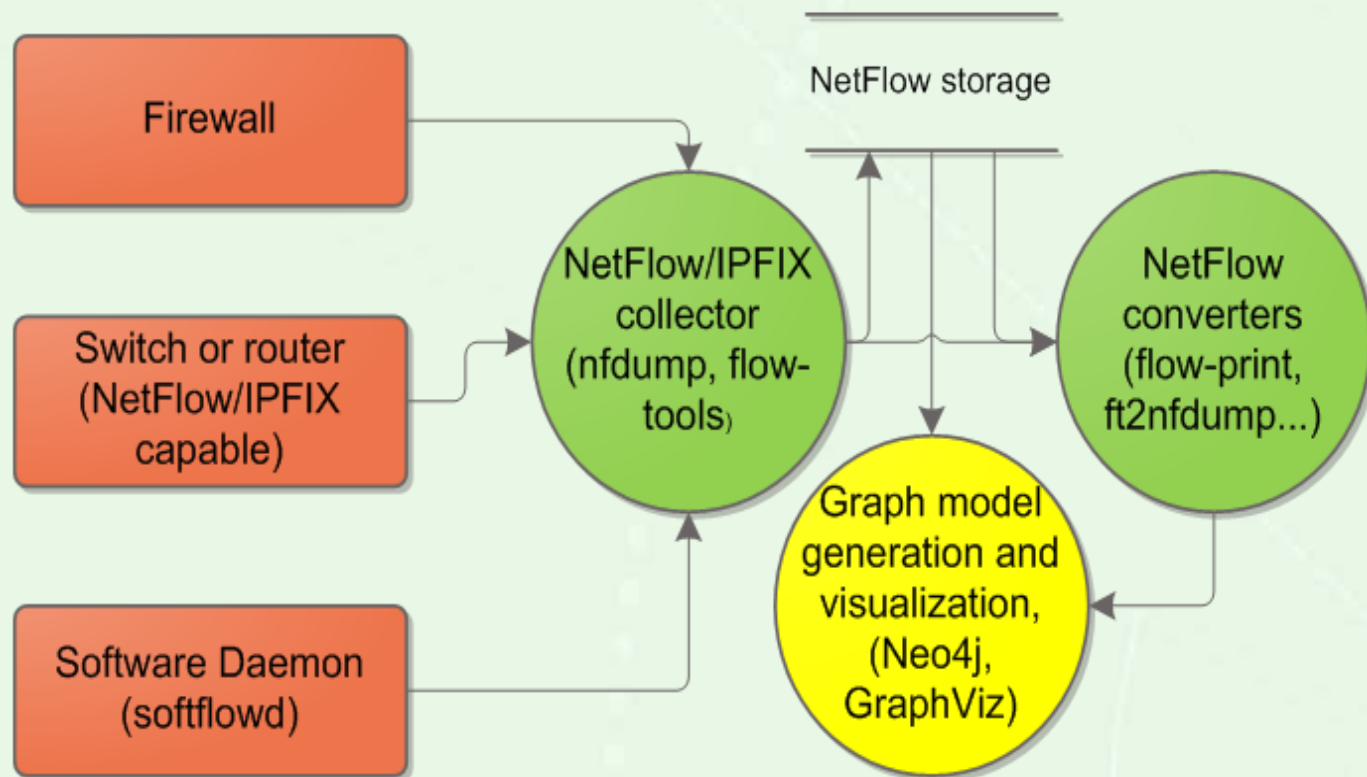
Dance by Matisse



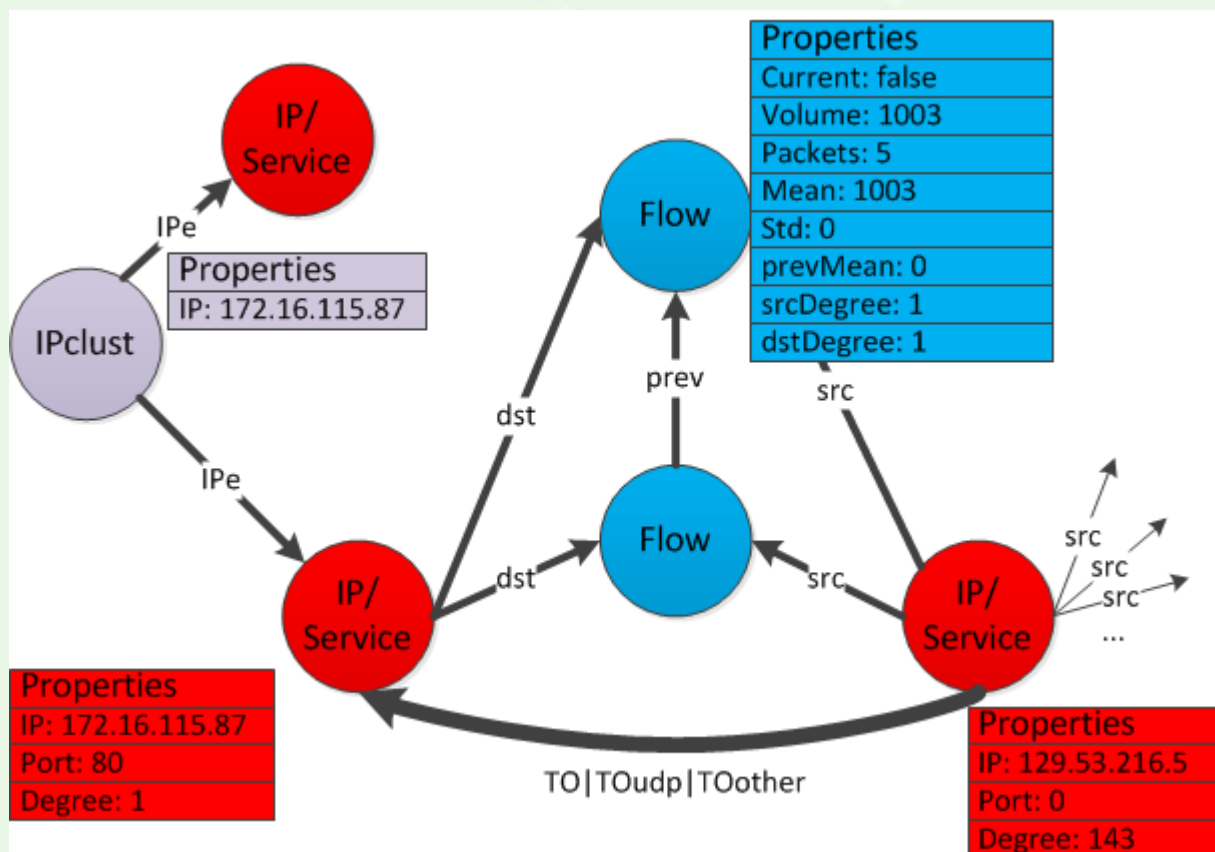
Melvin McGee – „Don't Touch the Painting”

Network flows (NetFlows) – collecting data for graph based model

- NetFlow/IPFIX processing general schema



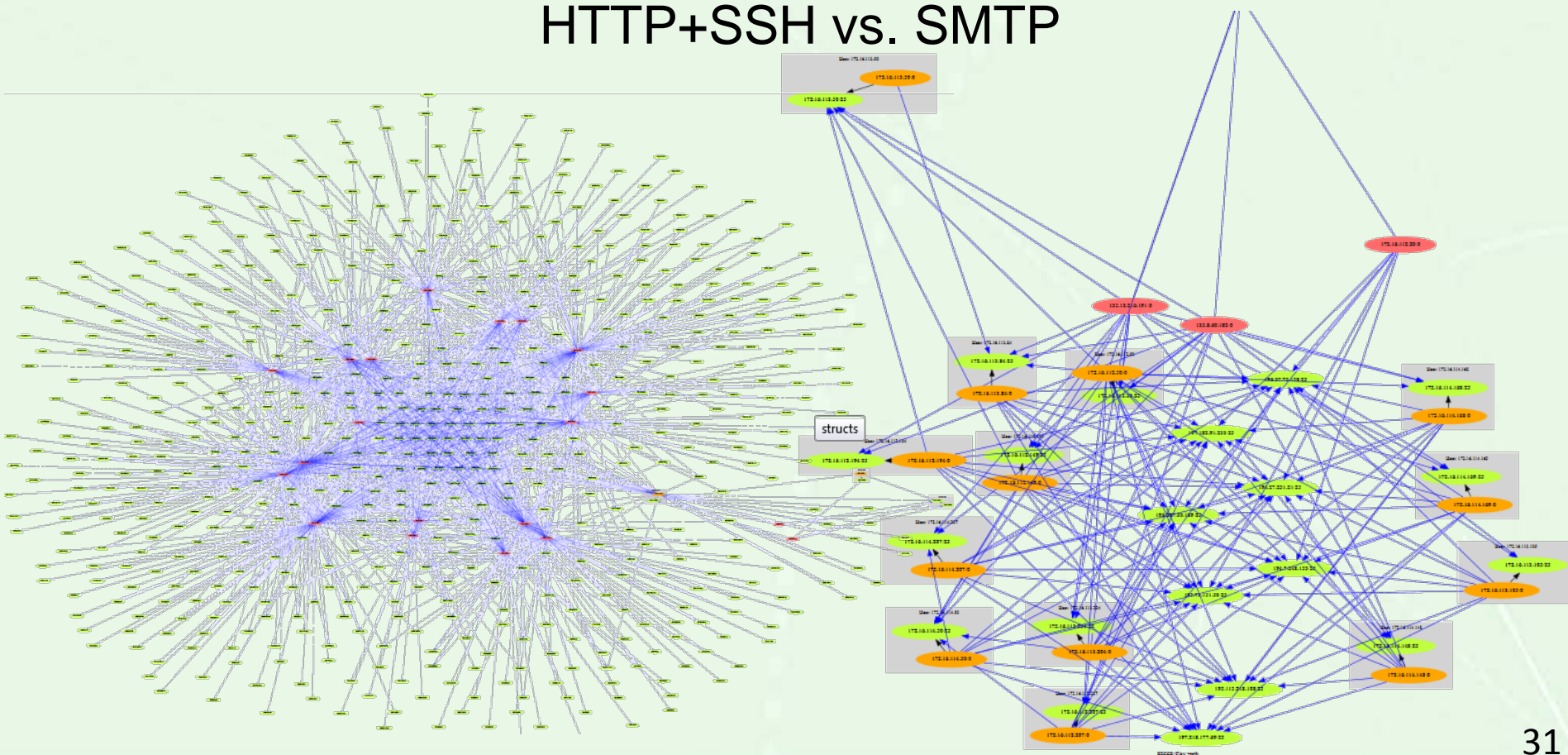
Network flows (NetFlows) – graph representations



Examples of simplified NetFlow graphs

DARPA sets

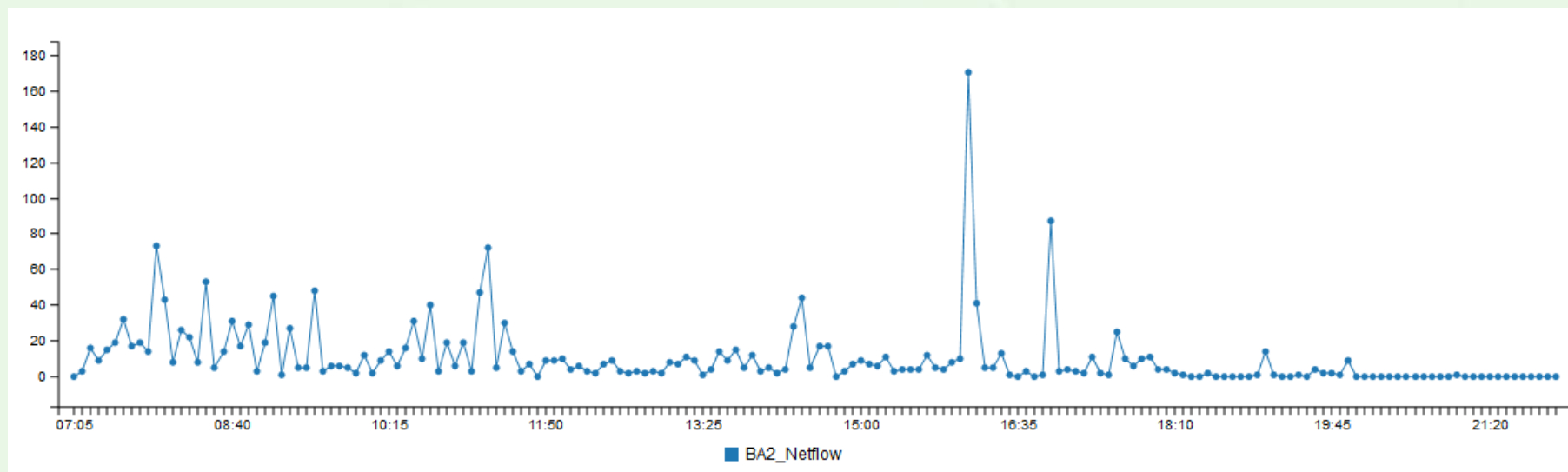
HTTP+SSH vs. SMTP



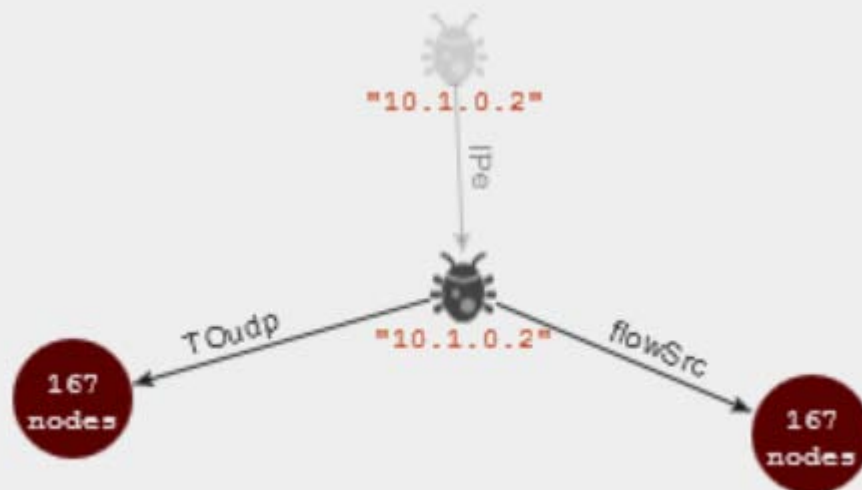
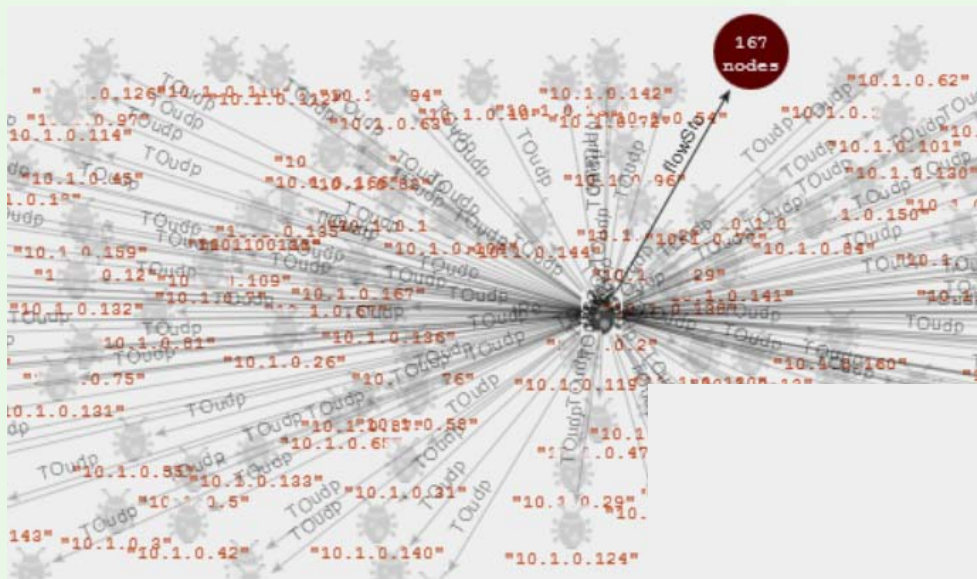
Graph edges dynamics

Example for attack scenario show earlier

- E.g. number of new edges corresponding to UDP connections created after the considered NetFlow dump

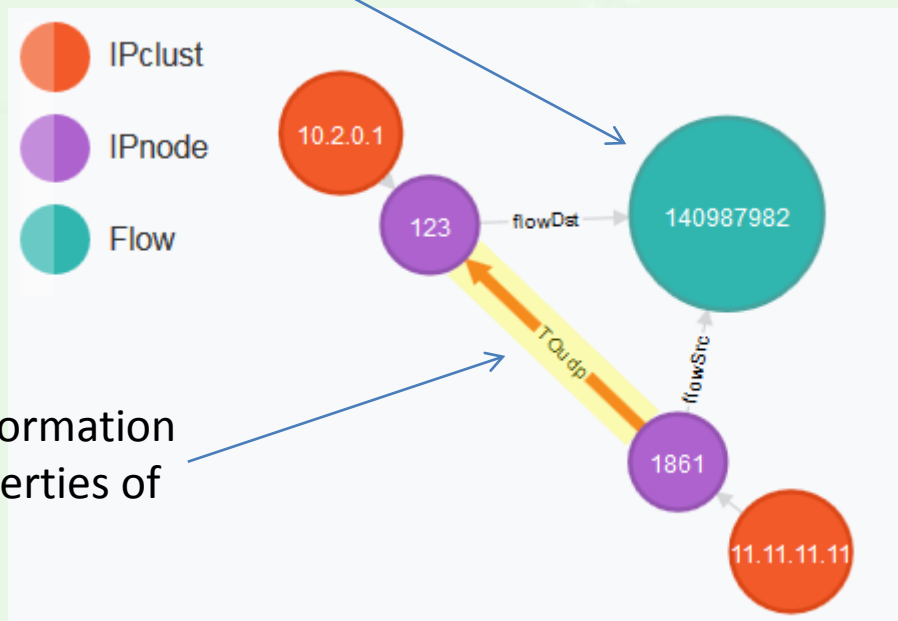


Topological changes



Increased volume

- Number of octets in flows

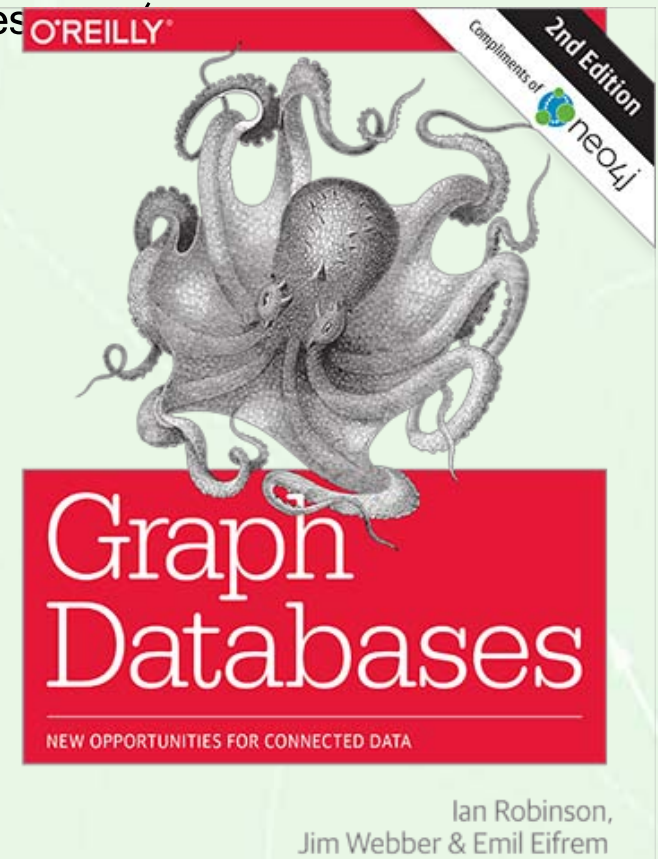
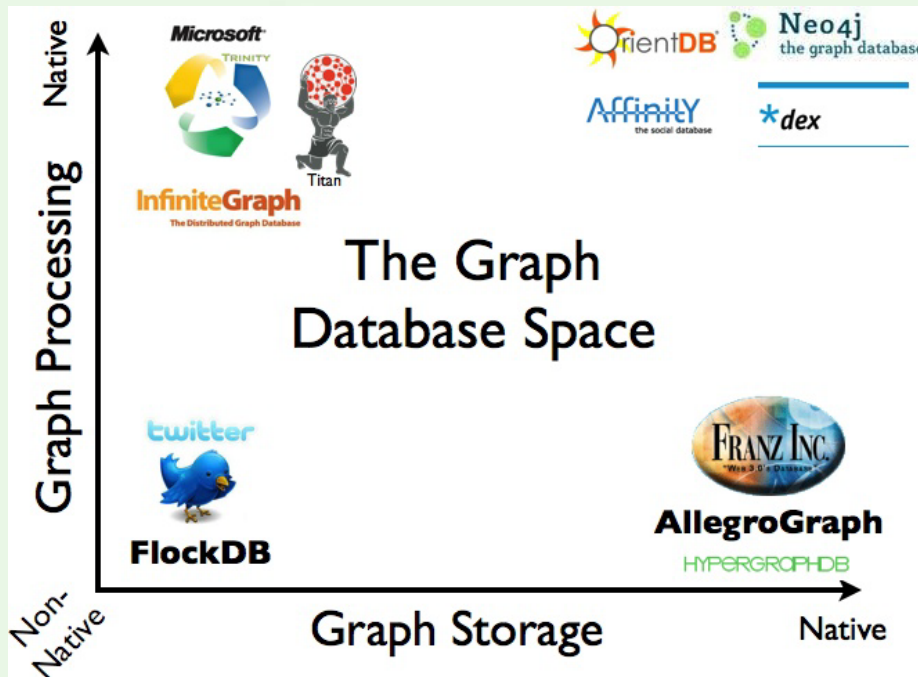


Aggregated volume information
are also stored as properties of
edge



Graph databases

- Book on Graph Databases: <http://graphdatabases.org>



Why Neo4j?

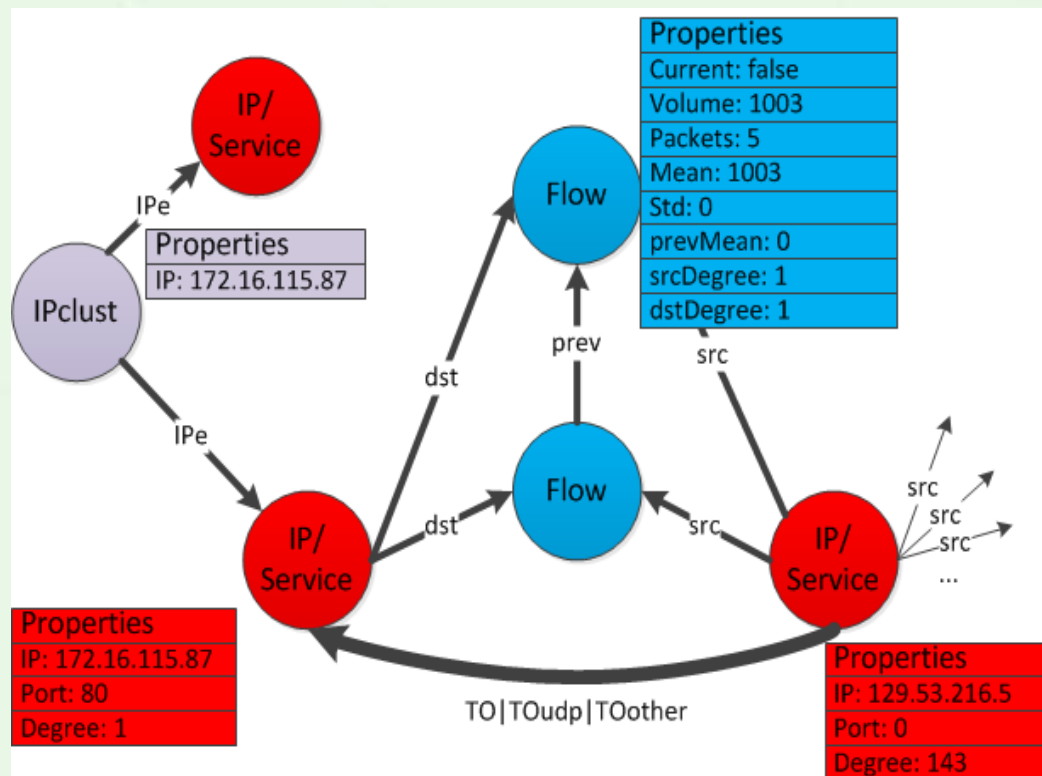
- One of the first Graph Databases (one of the market leaders)
- **Intuitive query language called Cypher**
- JDBC drivers
- Documented APIs
- Transactional properties
- Built-in management console and query console
- **Claimed support for relatively Big-data:**
 - data size is mainly limited by the address space of the primary keys for Nodes, Relationships, Properties and RelationshipTypes. Currently, the address space is as follows:
 - nodes 2^{35} (\sim 34 billion)
 - relationships 2^{35} (\sim 34 billion)
 - properties 2^{36} to 2^{38} depending on property types (maximum \sim 274 billion, always at least \sim 68 billion)
 - relationship types 2^{15} (\sim 32 000)

Cypher query example

Trivial example

- Identification of service listening on high ports and their clients
- Cypher query:

```
MATCH (ip:IPclust) --
    > (s:IPnode) --> (f:Flow
    {current:true}) <--
    (d:IPnode)
WHERE d.port > 1024
RETURN DISTINCT ip.ip, d.ip;
```



Result of the query

- Identification of service listening on high ports and their clients

- Cypher query:

```
MATCH (ip:IPclust)-->(s:IPnode)-->
    (f:Flow {current:true})<--(d:IPnode)
WHERE d.port >1024
RETURN DISTINCT ip.ip,d.ip;
```

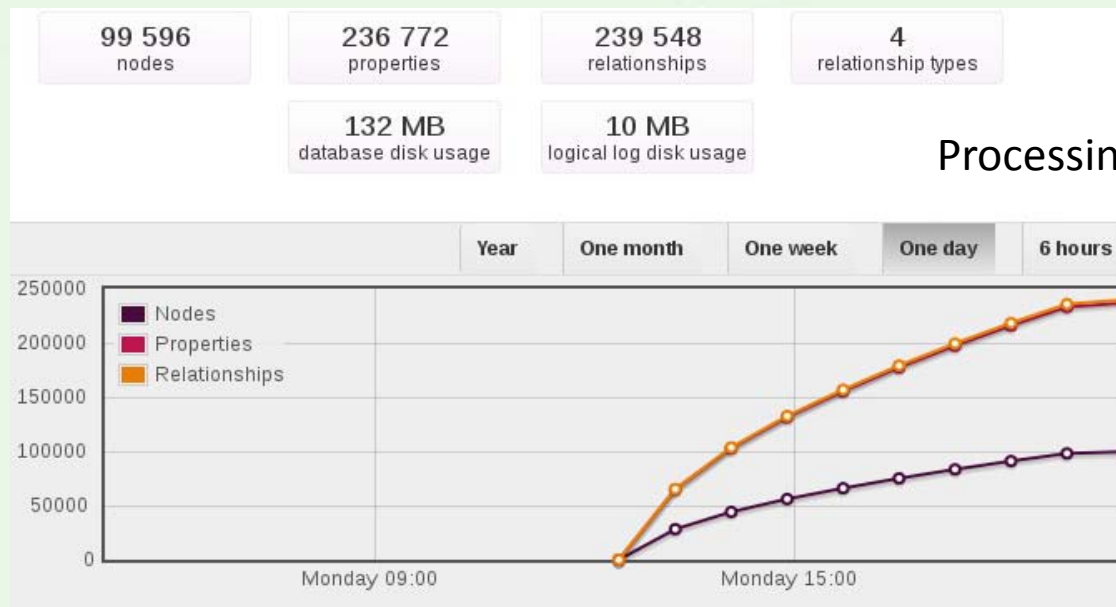
s.ip	d.ip
172.16.114.168	194.27.251.21
172.16.114.168	197.182.91.233
172.16.114.168	195.115.218.108
172.16.114.50	194.27.251.21
172.16.114.50	197.218.177.69
172.16.114.50	195.115.218.108
172.16.114.50	196.37.75.158
172.16.114.50	195.73.151.50
172.16.114.50	197.182.91.233
172.16.114.50	199.174.194.16

Adding new data using Cypher

```
CREATE CONSTRAINT ON (n:IPclust) ASSERT n.ip IS UNIQUE;
CREATE CONSTRAINT ON (n:IPnode) ASSERT n.ipport IS UNIQUE;
MERGE (n:IPnode {ip:"135.8.60.182",
    port:0,ipport:"135.8.60.182:0"});
MERGE (n:IPnode {ip:"172.16.112.207",
    port:20545,ipport:"172.16.112.207:20545"});
MERGE (n:IPclust {ip:"172.16.112.207"});
MATCH (n1:IPnode {ip:"135.8.60.182", port:0}),
    (n2:IPnode {ip:"172.16.112.207",
    port:20545,ipport:"172.16.112.207:20545"})
MERGE (n1)-[:src]->(x:Flow {current:true})<-[:dst]-(n2)
ON CREATE
    SET n1.degree=coalesce(n1.degree,0)+1
    SET n2.degree=coalesce(n2.degree,0)+1;
```


Encountered problems

- Merging nodes and relations on-line is inefficient due to locking and extensive scans



- Solution – use batch inserter instead of merge e.g. batch-import software

Solution

- Components
 - nfdump
 - Perl script processing aggregated NetFlow record from nfdump into batch-inserter format
 - Additional flat files storing historical edges to be able to spot changes in a communication pattern
 - Batch-import from: <https://github.com/jexp/batch-import>
 - Database reloader
 - The whole reload of database with 500k of NetFlow records take less than 30s, including restart of Neo4j engine
 - JDBC based query engine

More examples for attack scenario

TOP 5 -RPC communication

>>Executing query:

```
MATCH (n:IPnode) -- (c:IPnode) WHERE
    c.port = 135 or c.port=445 or
    c.port=593 RETURN n as IPnode, n.ip as
    Source, count(*) as
    NumberOfTargets, "RPC Scan Source" as
    Tag, "0.5" as Confidence ORDER BY
    NumberOfTargets DESC LIMIT 5
```

>>Results:

IPnode:

```
{ "ip": "10.1.0.2", "port": 0.0, "ip_port": "10.
1.0.2:0", "type": "I" },
```

Source: 10.1.0.2,

NumberOfTargets: 167,

Tag: RPC Scan Source,

Confidence: 0.5

IPnode:

```
{ "ip": "10.1.0.58", "port": 0.0, "ip_port": "1
0.1.0.58:0", "type": "IS" },
```

Source: 10.1.0.58,

NumberOfTargets: 3,

Tag: RPC Scan Source,

Confidence: 0.5

NTP monitoring for high volume traffic

>>Executing query:

MATCH (n:IPnode) -[r:TOudp] -> (c:IPnode) Where c.port=123 and r.sumOctets >100000 return n.ip as Target, r.sumOctets as Octets, c.ip as Source, "NTP DDOS" as Tag, "0.99" as Confidence

>>Results:

Target: 11.11.11.11,

Octets: 140987982,

Source: 10.2.0.1,

Tag: NTP DDOS,

- Confidence: 0.99

Complex query – change of mean values

Darpa set

```
MATCH (src:IPnode)-[:src]->(c:Flow {current:true})-[:prev*1..10]->(p:Flow)-[:dst]-(dst:IPnode)
WHERE c.nf=p.nf+10 and p.nf>5 and c.mean>p.mean
RETURN src.ipport,dst.ipport,c.nf,p.nf,c.mean,p.mean,p.std,
       max(c.mean-(p.mean+2*sqrt(p.std/(p.nf-1)))) as delta_mean
ORDER BY delta_mean DESC
```

Standard deviation

src.ipport	dst.ipport	c.nf	p.nf	c.mean	p.mean	stdev	p.std	delta_mean
172.16.113.105:0	197.218.177.69:20	19	9	729	185	97,588	76187	348,8282807
172.16.114.169:0	199.123.32.60:80	18	8	876	592	116,6	95171	50,80480279
172.16.113.105:0	204.146.18.33:80	24	14	517	500	1,4936	29	14,17157288
172.16.115.5:0	207.90.155.39:80	30	20	540	531	0,3974	3	9
172.16.114.168:0.0	207.25.71.142:80	17	7	466	446	5,9301	211	8,167840434
172.16.117.103:0	205.180.59.51:80	17	7	471	440	12,517	940	6,020008006
172.16.117.103:0	206.132.25.41:80	17	7	484	470	4,1433	103	5,753788749
....								

Additional possibilities

- Path traversal between a set of hosts
- Analysis of clusters properties
- Graph-clustering = process of finding communities (clusters/disjoint sub-graphs) in a graphs
- Clusters – groups of vertices; within each cluster the vertices are highly connected whereas there are only few edges between clusters
- Example methods:
 - Markov Clustering Algorithm (MCL) – it is straightforward to apply it in our model
 - Walktrap
 - Divide and merge strategies



Complex event processing (PSNC)

STIX, WSO2, OSGI

- All modules are sending events to CEP engine in STIX format
- WSO2 is used as CEP
- Correlation rules are defined
- Modules are controlled using OSGI framework

User application (1)

WSO₂

Complex Event Processor

Home

Dashboard

CEP Dashboard

Manage

Event Processor

Execution Plans

Event Streams

Services

List

Modules

List

Add

Shutdown/Restart

anode.menu

Registry

Anode - Statistical ANOmaly DETector

▶

●

Status: working

Available configs

Detection ▼

Set

Detected anomalies

↺

2015.03.26 16:00 BlockScan Suspected addresses: src("10.1.0.5"),dst("10.1.0.2","10.1.0.3","10.1.0.4","10.1.0.6",...) Confidentiality: H

2015.03.26 16:05 ServiceExploitation Suspected addresses: src("10.1.0.5"),dst("10.1.0.3","10.1.0.6","10.1.0.14","10.1.0.56",...) Confidentiality: M

2015.03.26 18:00 C&C communication Suspected addresses: src("10.1.0.3","10.1.0.6","10.1.0.56","10.1.0.89",...),dst("212.11.89.3") Confidentiality: M

2015.03.26 18:05 DDoS Suspected addresses: src("185.12.10.1"),dst("185.12.10.1") Confidentiality: H

2015.03.26 18:10 DDoS Suspected addresses: src("185.12.10.1"),dst("189.14.56.6") Confidentiality: H

User application (2)

WSO₂

Complex Event Processor

Home

Dashboard

CEP Dashboard

Manage

Event Processor

Execution Plans

Event Streams

Services

List

Modules

List

Add

Shutdown/Restart

anode.menu

Registry

Anode - Statistical ANOmaly DETector

▶

●

Status: working

Available configs

Detection ▼

Set

Detected anomalies

↻

2015.03.26 16:00 BlockScan Suspected addresses: src("10.1.0.5"),dst("10.1.0.2","10.1.0.3","10.1.0.4","10.1.0.6",...) Confidentiality: H

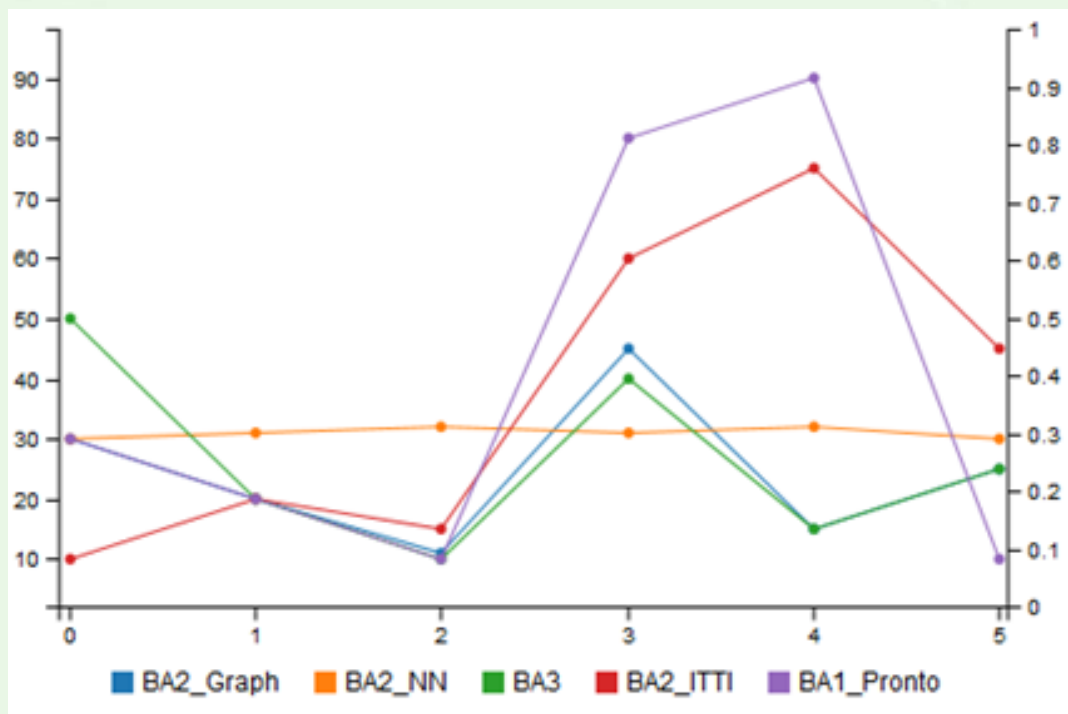
2015.03.26 16:05 ServiceExploitation Suspected addresses: src("10.1.0.5"),dst("10.1.0.3","10.1.0.6","10.1.0.14","10.1.0.56",...) Confidentiality: M

2015.03.26 18:00 C&C communication Suspected addresses: src("10.1.0.3","10.1.0.6","10.1.0.56","10.1.0.89",...)dst("212.11.89.3") Confidentiality: M

2015.03.26 18:05 DDoS Suspected addresses: src("185.12.10.1"),dst("185.12.10.1") Confidentiality: H

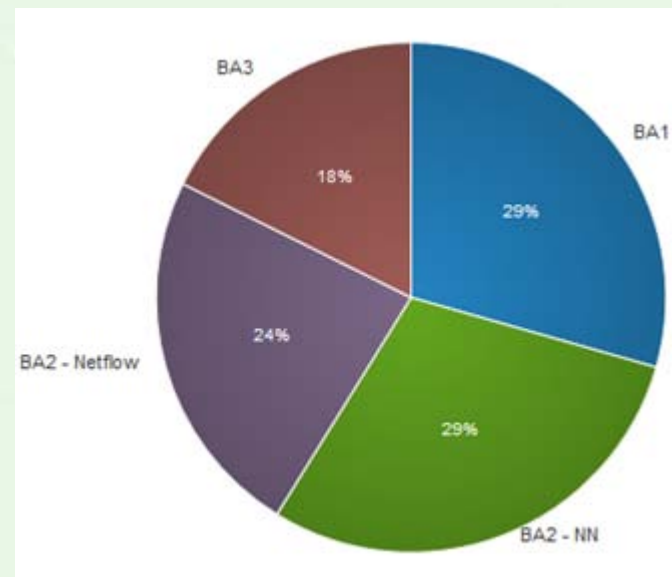
2015.03.26 18:10 DDoS Suspected addresses: src("185.12.10.1"),dst("189.14.56.6") Confidentiality: H

User application (3)



SECOR - confidence level by BA

A full pie chart show diferent confidence level by Block of Analysis.



Future plans

- Further research focused on detection algorithms
- Addition of more advanced correlation methods on CEP level
- Deployment in production environment
- Expansion to different environments (e.g. SCADA/ICS, sensor networks, IoT)
- Usage of obtained experience and results in new project



Thank you!

Dr Maciej Miłostan
miłos@man.poznan.pl
Security Team of PSNC
<http://security.psnc.pl>
security@man.poznan.pl



Poznań Supercomputing and Networking Center

affiliated to the Institute of Bioorganic Chemistry of the Polish Academy of Sciences,

ul. Noskowskiego 12/14, 61-704 Poznań, POLAND,
Office: phone center: (+48 61) 858-20-00, fax: (+48 61) 852-59-54,
e-mail: office@man.poznan.pl, <http://www.psnc.pl>