Towards Large-Scale Incident Response and Interactive Network Forensics

Matthias Vallentin
UC Berkeley / ICSI
vallentin@icir.org

Dissertation Proposal
UC Berkeley

December 14, 2011

April 21, 2009: Bad News for UC Berkeley

Archives | RSS

« UC Regents: Higher Tuition Mea... | Ron Davis for Oakland Police C... »

FRIDAY, MAY 8, 2009

Hacked! UC Berkeley Health Records

LAW ENFORCEMENT & CRIME / EDUCATION / HEALTH & MEDICINE   Kathleen Richards — Fri, May 8, 2009 at 11:25 AM

Notices went out today to approximately **160,000** **UC Berkeley alumnus and students** that the University's Health Services electronic databases, which contain personal information, have been hacked. According to the e-mail, the stolen information varies from person to person, but could include Social Security numbers, health insurance coverage, immunization history, and self-reported health history. The hacking occurred from October 9, 2008 to April 6, 2009, but wasn't determined until April 21. The university has alerted campus police detectives and the FBI, and has hired an outside auditor, Price Waterhouse Coopers, to help with the investigation. Let's hope the hackers aren't **demanding a multimillion-dollar ransom**; the regents may have to **raise tuition costs** even more.

Blind SQL Injection

Havij

```
..?deploy_id=799+and+ascii(substring((database()),1,1))<79     31
..?deploy_id=799+and+ascii(substring((database()),1,1))<103    11582
..?deploy_id=799+and+ascii(substring((database()),1,1))<91     31
..?deploy_id=799+and+ascii(substring((database()),1,1))<97     31
..?deploy_id=799+and+ascii(substring((database()),1,1))<100    11582
..?deploy_id=799+and+ascii(substring((database()),1,1))=99     11582
..?deploy_id=799+and+ascii(substring((database()),2,1))<79     31
..?deploy_id=799+and+ascii(substring((database()),2,1))<103    31
..?deploy_id=799+and+ascii(substring((database()),2,1))<115    11582
..?deploy_id=799+and+ascii(substring((database()),2,1))<109    11582
..?deploy_id=799+and+ascii(substring((database()),2,1))<106    11582
..?deploy_id=799+and+ascii(substring((database()),2,1))=105    11582
..?deploy_id=799+and+ascii(substring((database()),3,1))<79     31
..?deploy_id=799+and+ascii(substring((database()),3,1))<103    11582
..?deploy_id=799+and+ascii(substring((database()),3,1))<91     31
```

Database name: `ci...`

`Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727) Havij`

Example: Debugging an APT Incident

Advanced Persistent Threat (APT)

Severe security breaches manifest over large time periods

1. Initial compromise: stealthy and inconspicuous
2. Maintenance: periodic access checks
3. Sudden strike: quick and devastating

   or

3. Continuous leakage: piecemeal exfiltration under the radar

Analyst questions

- How did the attacker get in?
- How long did the attacker stay under the radar?
- What is the damage?
- Was an insider involved?
- How to detect similar attacks *in the future*?
- How do we describe the attack?

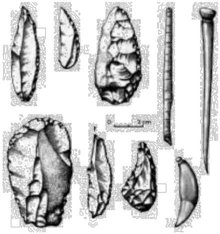
Incident Response Challenges and the Sobering Reality

Challenges

- **Volume**: machine-generated data exceeds our analysis capacities
- **Heterogeneity**: multitude of data and log formats
- **Procedure**: unsystematic investigations

Reality

- Reliance on incomplete context
- Manual ad-hoc analysis
- UNIX tools (awk, grep, uniq)
- Expert islands



How do we tackle this situation?

Thesis Statement

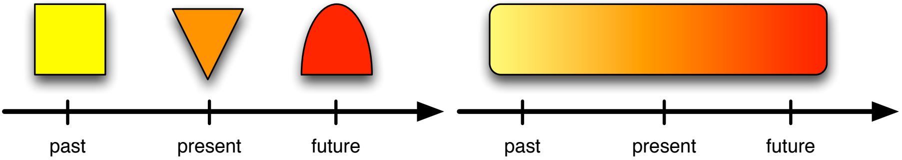
Hypothesis

Key operational networking tasks, such as incident response and forensic investigations, base their decisions on descriptions of activity that are fragmented across *space* and *time*:

- **Space**: heterogeneous data formats from disparate sources
- **Time**: discrepancy in expressing past and future activity

Statement

We can design and build a system to attain a *unified* view across space and time.



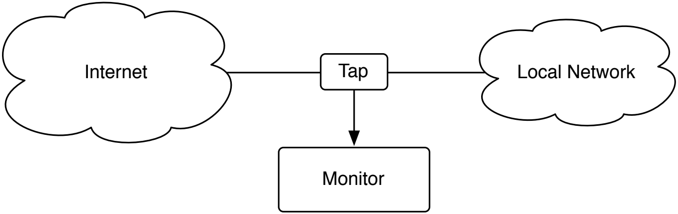
Outline

1. Prior Work: Building a NIDS Cluster
2. Use Cases
3. Workload Characterization
4. Requirements
5. Related Work
6. Architecture
7. Roadmap
8. Summary

Outline

1. Prior Work: Building a NIDS Cluster
2. Use Cases
3. Workload Characterization
4. Requirements
5. Related Work
6. Architecture
7. Roadmap
8. Summary

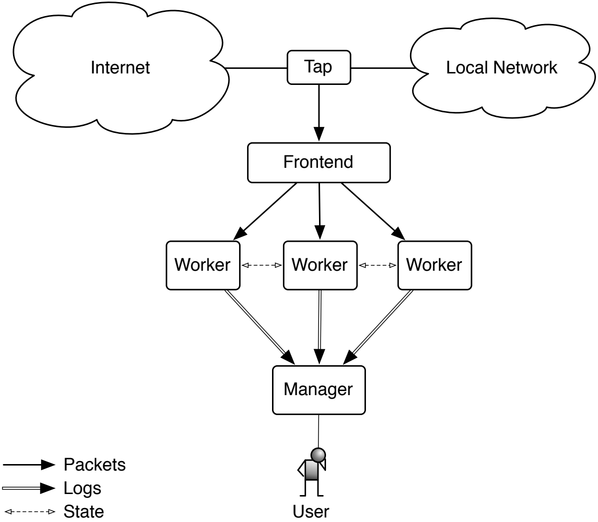
Basic Network Monitoring



- Passive tap splits traffic
  - Optical
  - Coppper
  - Switch span port
- Monitor receives full packet stream

→ Challenge: do not fall behind processing packets!

High-Performance Network Monitoring: The NIDS Cluster [VSL+07]



The NIDS Cluster

- Contributions
  - Design, prototype, and evaluation of cluster architecture
  - Bro scripting language enhancements
- Runs now **in production** at large sites with a 10 Gbps uplink:
  - UC Berkeley (26 workers), 50,000 hosts
  - LBNL (15 workers), 12,000 hosts
  - NCSA (10 $\times$ 4-core workers), 10,000 hosts
- Generates follow-up challenges
  - How to archive and process the *output* of the cluster?
  - How to *efficiently* support incident response and network forensics?

Outline

1. Prior Work: Building a NIDS Cluster
2. Use Cases
3. Workload Characterization
4. Requirements
5. Related Work
6. Architecture
7. Roadmap
8. Summary

Use Case #1: Classic Incident Response

- $\boxed{\textbf{Goal}\text{: quickly isolate scope and impact of security breach}}$
- Often begins with a piece of **intelligence**
  - "IP X serves malware over HTTP"
  - "This MD5 hash is malware"
  - "Connections to `128.11.5.0/27` at port 42000 are malicious"
- Analysis style: Ad-hoc, interactive, several refinements/adaptions
- Typical operations
  - **Filter**: project, select
  - **Aggregate**: `mean`, `sum`, `quantile`, `min/max`, `histogram`, `top-k`, `unique`

⇒ Bottom-up: concrete starting point, then widen scope

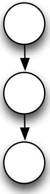
Use Case #2: Network Troubleshooting

- $\boxed{\textbf{Goal}\text{: find root cause of component failure}}$
- Often no specific hint, merely symptomatic feedback
  - "Email does not work :-/"
- Typical operations
  - **Zoom**: slice activity at different granularities
    - Time: seconds, minutes, days, ...
    - Space: layer 2/3/4/7, protocol, host, subnet, domain, URL, ...
  - Study **time series** data of activity aggregates
  - Find abnormal activity
    - "A sudden huge spike in DNS traffic"
    - Use past behavior to determine present impact [KMV+09] and predict future [HZC+11]
    - Judicious machine learning [SP10]

⇒ Top-down: start broadly, then narrow scope incrementally

Use Case #3: Combating Insider Abuse

- **Goal**: uncover policy violations of personnel
- Insider attack:
  - Chain of **authorized** actions, hard to detect individually
  - E.g., data exfiltration
    1. User logs in to internal machine
    2. Copies sensitive document to local machine
    3. Sends document to third party via email



- Analysis procedure: connect the dots
  - Identify first action: gather and compare activity profiles
    - "Vern accessed 10x more files on our servers today" [SS11]
    - "Ion usually does not log in to our backup machine at 3am"
  - Identify last action:
    - Filter fingerprints of sensitive documents at border
    - Reinspect past activity under new bias

⇒ Relate temporally distant events

Outline

1. Prior Work: Building a NIDS Cluster
2. Use Cases
3. Workload Characterization
4. Requirements
5. Related Work
6. Architecture
7. Roadmap
8. Summary

Descriptions of Activity: Bro Event Trace

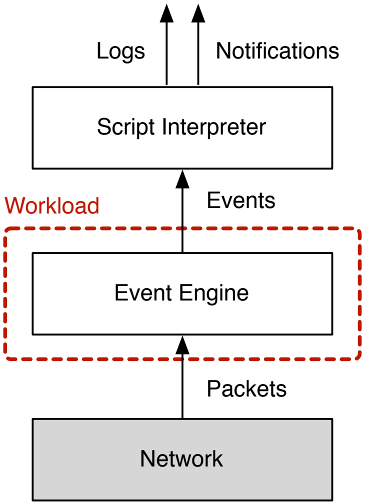
- Use Bro event trace

→ Descriptions of activity

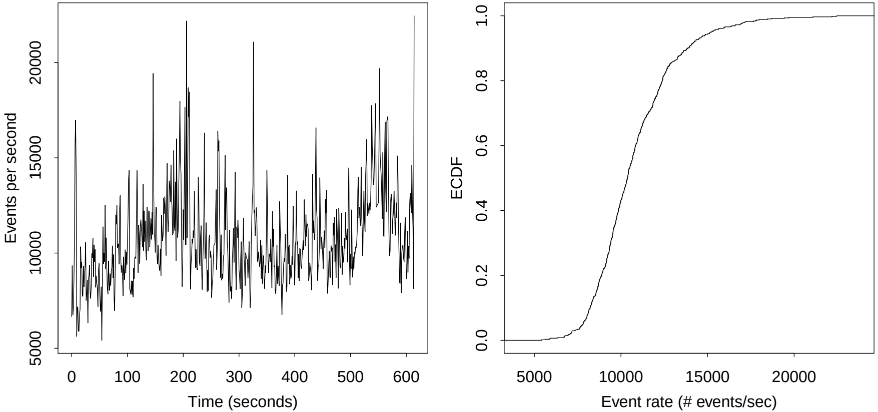
- Instrumentation: *meta events*
  - Timestamp
  - Name
  - Size
- Generate from real UCB traffic

Trace Details

- October 17, 2011, 2:35pm, 10 min
- 219 GB
- 284,638,230 packets
- 6,585,571 connections



Event Workload of one node (1/26)



Estimator	Events/sec	MB/sec
Median	10,760	13.4
Mean	10,370	14.2
Peak	22,460	35

$\rightarrow$ need to support peaks of 10^6 events/sec and 1 GB/sec

Outline

1. Prior Work: Building a NIDS Cluster
2. Use Cases
3. Workload Characterization
4. Requirements
5. Related Work
6. Architecture
7. Roadmap
8. Summary

Requirements

- **Interactivity**
  - Security-related incidents are time-critical
- **Scalability**
  - Distributed system to handle high ingestion rates
  - Aging: graceful roll-up of older data
- **Expressiveness**
  - Represent arbitrary activity
- **Result Fidelity**
  - Trade latency for result correctness
- **Analytics & Streaming**
  - A unified approach to querying historical and live data

Outline

1. Prior Work: Building a NIDS Cluster
2. Use Cases
3. Workload Characterization
4. Requirements
5. Related Work
6. Architecture
7. Roadmap
8. Summary

Traditional Views

- Data Base Management Systems (DBMS)
  - Store first, query later
  - \+ Generic
  - − Monolithic
- Data Stream Management Systems (DSMS)
  - Process and discard
  - \+ High throughput
  - − No persistence
- Online Transactional Processing (OLTP)
  - Small transactional inserts/updates/deletes
  - \+ Consistency
  - − Overhead
- Online Analytical Processing (OLAP)
  - Aggregation over many dimensions
  - \+ Speed
  - − Batch loads

Newer Movements

- NoSQL
  - + Scalability
  - − Flexibility
- MapReduce
  - + Expressive
  - − Batch processing
- In-memory Cluster Computing
  - + Speed
  - − Streaming data, initial load

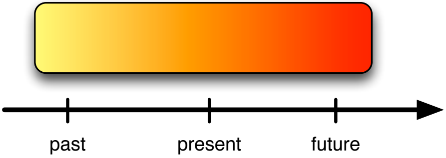
Outline

1. Prior Work: Building a NIDS Cluster
2. Use Cases
3. Workload Characterization
4. Requirements
5. Related Work
6. Architecture
7. Roadmap
8. Summary

VAST: Visibility Across Space and Time

VAST

- **Visibility**
  - Realize interactive data explorations
- Across **space**:
  - Unify heterogeneous data formats
- Across **time**:
  - Express past and future behavior uniformly



Bro's Data Model: Declaration and Instantiation

- **Rich-typed**: first-class networking types (addr, port, subnet, ...)
- **Semi-structured**: nested data with container types

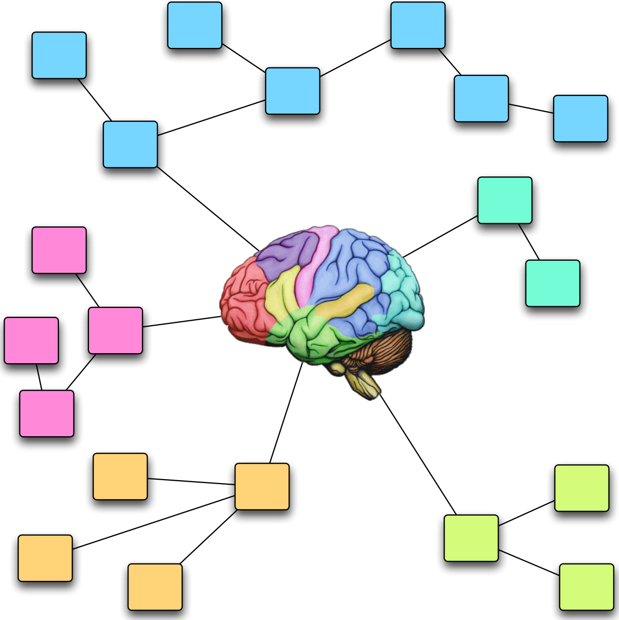
Event declaration (simplified)

```
type connection: record { orig: addr, resp: addr, ... }
event connection_established(c: connection)
event http_request(c: connection, method: string, URI: string)
event http_reply(c: connection, status: string, data: string)
```

Event instantiation

```
connection_established({127.0.0.1, 128.32.244.172, ... })
http_request({127.0.0.1, 128.32.244.172, ..}, "GET", "/index.html")
http_reply({127.0.0.1, 128.32.244.172, ..}, "200", "<!DOCTYPE ht..")
http_request({127.0.0.1, 128.32.244.172, ..}, "GET", "/favicon.ico")
http_reply({127.0.0.1, 128.32.244.172, ..}, "200", "\xBE\xEF\x..")
connection_established({127.0.0.1, 128.32.112.224, ... })
```

Network-Wide Unified Representation of Activity

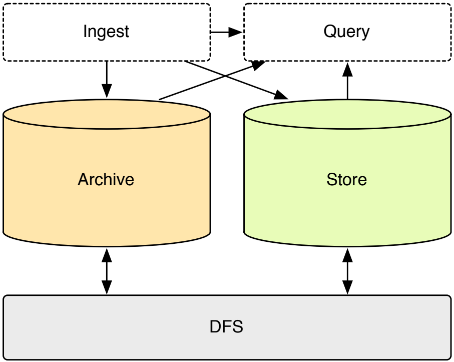


Expressing Behavior *Between* Events [VHM+11]

$$
\begin{array}{rcll}
\phi & ::= & \text{`('}\ S \,|\, \phi\ \text{`)'}\ \{\, bcon \,\} & \\
 & | & \texttt{not}\ \phi & (negation) \\
 & | & \phi\ \texttt{and}\ \phi & (logical\ and) \\
 & | & \phi\ \texttt{or}\ \phi & (logical\ or) \\
 & | & \phi\ \texttt{xor}\ \phi & (logical\ xor) \\
 & | & \phi \leadsto_{(opcon)} \phi & (leadsto) \\
 & | & \Box_{(opcon)}\ \phi & (always) \\
 & | & \phi\ \texttt{olap}_{(opcon)}\ \phi & (overlaps) \\
 & | & \phi\ \texttt{dur}_{(opcon)}\ \phi & (during) \\
 & | & \phi\ \texttt{sw}_{(opcon)}\ \phi & (startswith) \\
 & | & \phi\ \texttt{ew}_{(opcon)}\ \phi & (endswith) \\
 & | & \phi\ \texttt{eq}_{(opcon)}\ \phi & (equals) \\
bcon & ::= & \text{`['}\ \{tc \,|\, cc\}\ \text{`]'} & \\
tc & ::= & \{\texttt{at} \,|\, \texttt{duration} \,|\, \texttt{end}\}\ relop\ t\{:t\} & \\
cc & ::= & \{\texttt{icount} \,|\, \texttt{bcount} \,|\, \texttt{rate}\}\ relop\ c\{:c\} & \\
opcon & ::= & \text{`['}\ relop\ t\{:t\}\ \text{`]'} & \\
relop & ::= & \{ > \,|\, < \,|\, = \,|\, \geq \,|\, \leq \,|\, \neq \} & \\
t & ::= & [0-9]+\ \{s|ms\} & \\
c & ::= & [0-9]+ &
\end{array}
$$

VAST: Architecture Overview

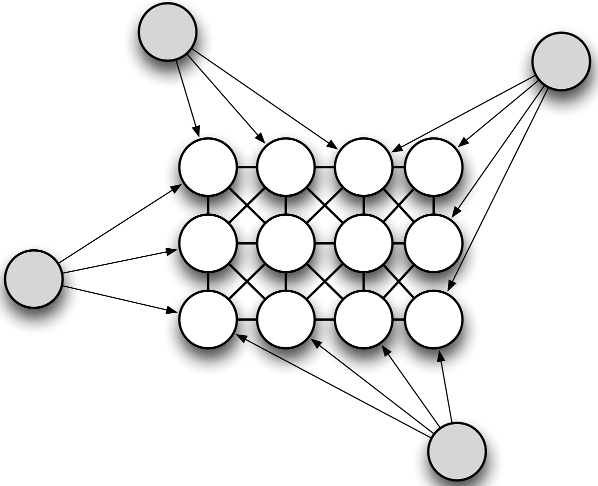
- Distributed architecture
  - Elasticity via MQ middle layer
  - Exchangeability of components
- **DFS**: fault-tolerance, replication
- **Archive**: key-value store
  - Contains serialized events
- **Store**
  - Partitioned in-memory column-store
  - Cache semantics (e.g., LRU)
  - Indexing via compressed bitmaps



Software Reuse

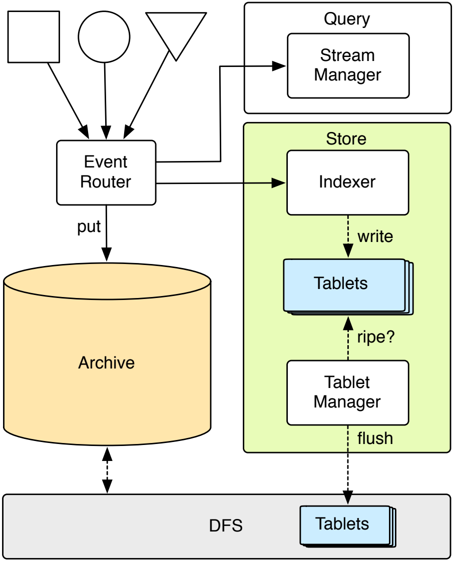
- Don't build from scratch unless necessary
- Reuse?
  - Streaming: SparkStream
  - Archive: Spark, memcached
  - Query engine: Shark
  - DFS: HDFS, KFS
- Build
  - Store
  - Glue for unified data model

Distributed Ingestion



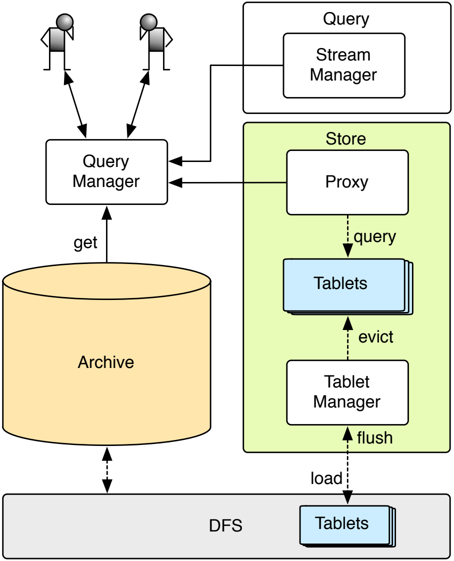
Ingest

1. Events arrive at **Event Router**
   1.1 Assign UUID `x`
   1.2 Put `(x, event)` in archive
   1.3 Forward event to **Indexer**
   1.4 Forward event to **Stream Manager**
2. **Indexer** writes event into tablet
   - Group related activity
3. **Tablet Manager** flushes "ripe" tablets based on
   - Reached capacity (bytes or events)
   - Last access
   - Age



Query

1. User or NIDS issues query
2. **Query Manager**
   - Distributes query to data nodes
   - Spins up new nodes
3. **Proxy** hits tablet index
   a Generates direct result (as tablet)
   b Returns set of UUIDs
      → Archive lookup
4. **Tablet Manager**
   - Flush and load tablets



Meeting the Requirements

- **Interactivity**
  - → In-memory cache of tablets
  - → (Bitmap) Indexing
- **Scalability**
  - → Messaging middle-layer (MQ)
  - → Distributed architecture
- **Expressiveness**
  - → Data: Bro's event model
  - → Query: Rich inter-event relationships
- **Result Fidelity**
  - → Sampling & Bootstrapping
- **Analytics & Streaming**
  - → Historical queries: tablet-based storage + archive
  - → Live queries: stream processing engine

Outline

1. Prior Work: Building a NIDS Cluster
2. Use Cases
3. Workload Characterization
4. Requirements
5. Related Work
6. Architecture
7. Roadmap
8. Summary

Roadmap

1. Identify Building Blocks
   - What existing systems to leverage?
   - What to build? What not?

   → Time Estimate: 1 month
2. Infrastructure
   - Core data structures to represent activity
   - Message-passing middle layer

   → Time Estimate: 1-2 months
3. Ingestion
   - How to handle high-volume event stream?
   - Provide circular buffer semantics: recent activity in-memory

   → Time Estimate: 2 months

Roadmap

4. Query
   - Express and implement data queries: historical & live
   - Express and implement behavior models
   - Bounding errors: trading accuracy for latency

   → Time Estimate: 3-4 months
5. Testing Usability
   - Bring in early adopters: ICSI, LBNL, NCSA
   - Deploy-measure-tweak cycle: integrate feedback, fix bugs

   → Time Estimate: 1 month
6. Real-World Evaluation
   - Use the system *in production* for real incidents
   - Learn how effectively it supports incident response & forensics

   → Time Estimate: 2 month
7. Tuning
   - Time: Build bitmap indexing on top of tablet store
   - Space: elevate old activity into higher-level abstractions (aging)
   - Address the lessons learned from the evaluation

   → Time Estimate: 3-4 months

Outline

1. Prior Work: Building a NIDS Cluster
2. Use Cases
3. Workload Characterization
4. Requirements
5. Related Work
6. Architecture
7. Roadmap
8. Summary

Recapitulation

1. Large-scale operational network analysis is ill-supported today
    - No homogeneous representation of activity
    - Dealing with past activity differs from expressing future events
2. We need an integrated platform to better support these tasks
3. Derived requirements based on workload characterization
4. Presented an architecture draft

Thank You

FIN

References I

F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber.
Bigtable: A Distributed Storage System for Structured Data.
ACM Transactions on Computer Systems (TOCS), 26(2):1–26, 2008.

A. Colantonio and R. Di Pietro.
Concise: Compressed 'n' Composable Integer Set.
Information Processing Letters, 110(16):644–650, 2010.

Francesco Fusco, Marc Ph. Stoecklin, and Michail Vlachos.
NET-FLi: On-the-fly Compression, Archiving and Indexing of Streaming Network Traffic.
Proceedings of the VLDB Endowment, 3:1382–1393, September 2010.

References II

Amir Houmansadr, Ali Zand, Casey Cipriano, Giovanni Vigna, and Christopher Kruegel.
Nexat: A History-Based Approach to Predict Attacker Actions.
In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, Orlando, Florida, December 2011.

Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Paramvir Bahl.
Detailed Diagnosis in Enterprise Networks.
In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 243–254, New York, NY, USA, 2009. ACM.

Andrew Lamb.
Building Blocks for Large Analytic Systems.
In *5th Extremely Large Databases Conference*, XLDB '11, Menlo Park, California, October 2011.

References III

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis.
Dremel: Interactive Analysis of Web-Scale Datasets.
Proceedings of the VLDB Endowment, 3(1-2):330–339, September 2010.

Robert Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan.
Interpreting the Data: Parallel Analysis with Sawzall.
Scientific Programming, 13(4):277–298, 2005.

Robin Sommer and Vern Paxson.
Outside the Closed World: On Using Machine Learning for Network Intrusion Detection.
In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 305–316, Washington, DC, USA, 2010. IEEE Computer Society.

References IV

Malek Ben Salem and Salvatore J. Stolfo.
Modeling User Search Behavior for Masquerade Detection.
In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, RAID '11, Menlo Park, CA, 2011.

Arun Viswanathan, Alefiya Hussain, Jelena Mirkovic, Stephen Schwab, and John Wroclawski.
A Semantic Framework for Data Analysis in Networked Systems.
In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI '11, Boston, MA, 2011. USENIX Association.

References V

Matthias Vallentin, Robin Sommer, Jason Lee, Craig Leres, Vern Paxson, and Brian Tierney.
The NIDS Cluster: Scalably Stateful Network Intrusion Detection on Commodity Hardware.
In *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*, RAID '07, pages 107–126, Gold Goast, Australia, September 2007. Springer.

Kesheng Wu, Ekow J. Otoo, Arie Shoshani, and Henrik Nordberg.
Notes on Design and Implementation of Compressed Bit Vectors.
Technical Report LBNL-3161, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, 94720, 2001.

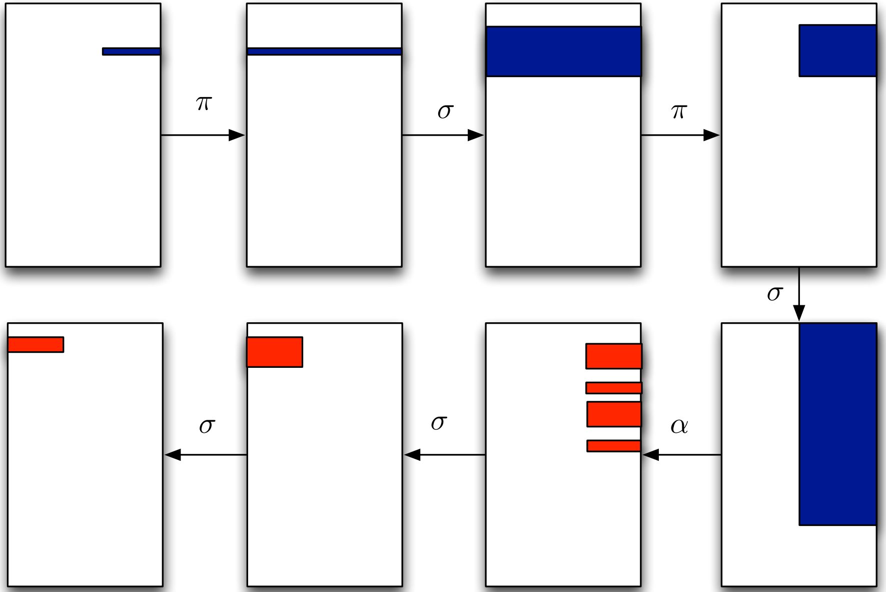
Kesheng Wu.
FastBit: an Efficient Indexing Technology for Accelerating Data-Intensive Science.
Journal of Physics: Conference Series, 16:556–560, 2005.

References VI

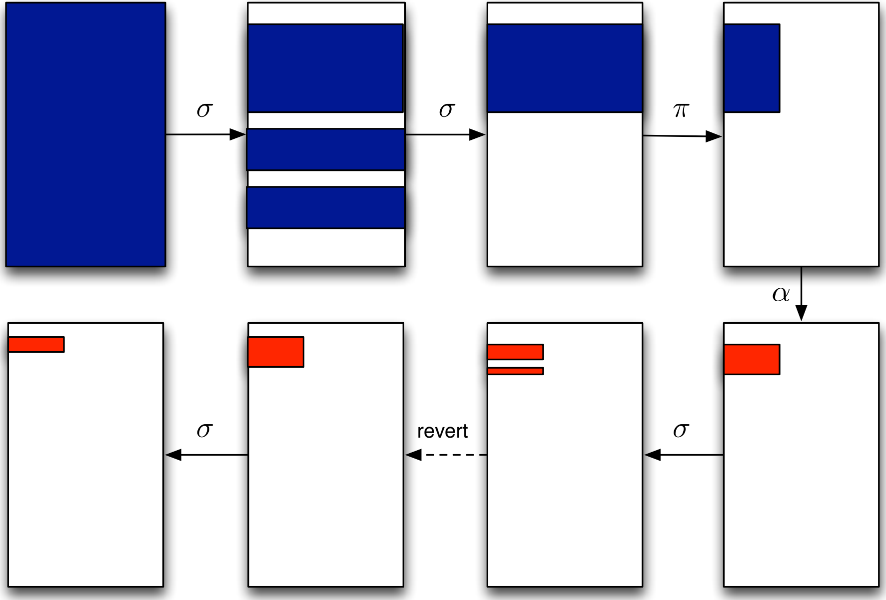
Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica.
Spark: Cluster computing with working sets.
In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud '10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

Backup Slides

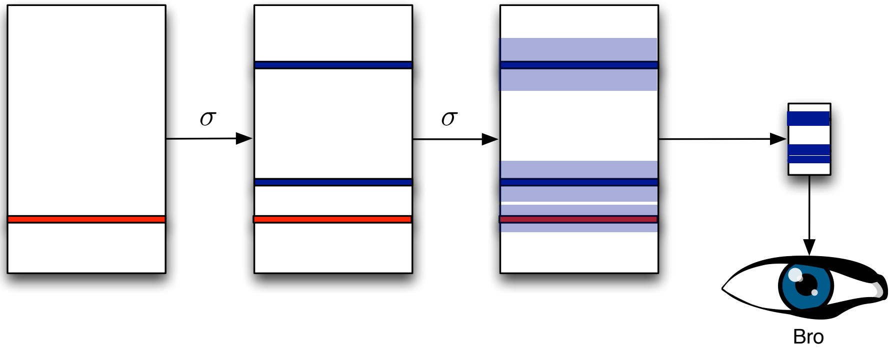
Illustrating Bottom-Up Data Navigation



Illustrating Top-Down Data Navigation



Illustrating Insider Abuse Data Navigation

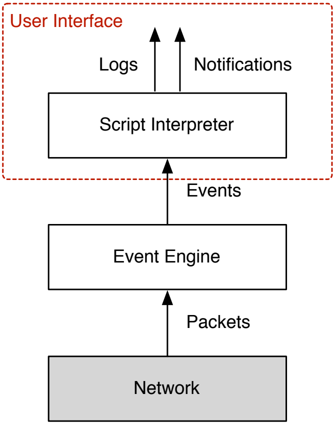


The Bro Network Security Monitor

- Fundamentally different from other IDS
- Network analysis platform
- Policy-neutral at the core
- Highly stateful

Key components

1. Event engine (core)
   - TCP stream reassembly
   - Protocol analysis
2. Script interpreter
   - "Domain-specific Python"
   - Generates extensive logs

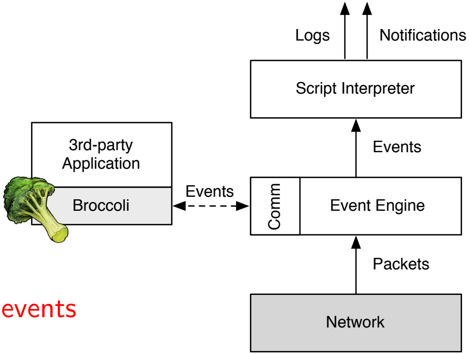


Generating and Receiving Bro Events

Broccoli

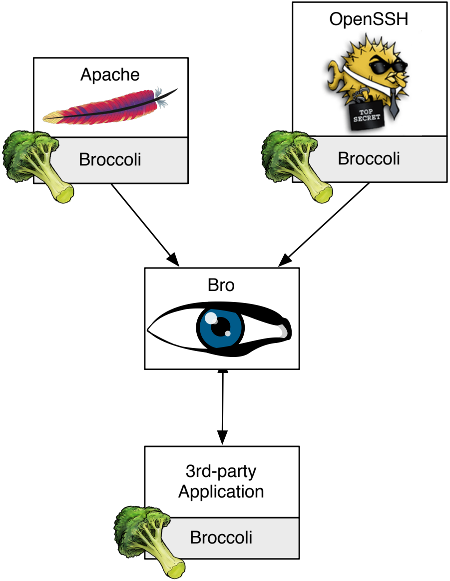
- C library
- Send/Receive Bro events
- Language bindings
  - Ruby
  - Python
  - Perl

→ Anyone can generate/receive events



(Broccoli = Bro client communications library)

Publish/Subscribe in the Bro Event Model



Expressing Behavior [VHM+11]

- Requirements
  - Analysis over multi-type, multi-variate, timestamped data
  - Analysis over higher-level abstractions
  - Composition of abstractions
  - A wide variety of relationships
- Relationships
  - Causality
  - Partial or total ordering
  - Dynamic changes over time
  - Concurrency
  - Polymorphism
  - Synchronous and asynchronous operations
  - Eventual operations
  - Value dependencies
  - Invariants
  - Basic relations: boolean operators, loops, etc.

Expressing Behavior *Between* Events [VHM+11]

```
[behavior]
initial_query = (AtoV_query ~> VtoR_query)
b_1 = initial_query~>RtoV_resp ~> (AtoV_resp xor
                                    AtoV_noresp)
b_2 = initial_query ~> AtoV_noresp ~> RtoV_resp
b_3 = initial_query ~> AtoV_resp ~> RtoV_resp

[model]
FAILURE(sip,dip,sport,dport,dnsid,dnsauth) = b_1 or b_2
SUCCESS(sip,dip,sport,dport,dnsid,dnsauth) = b_3
```

Inspirations

1. Dremel [MGL+10]
    - In-situ data access
    - Columnar storage
    - Nested data model
2. Bigtable [CDG+08]
    - Sharding: distributed tablets
3. Sawzall [PDGQ05]
    - Aggregators: `sample`, `sum`, `maximum`, `quantile`, `top-k`, `unique`
4. Spark [ZCF+10]
    - In-memory computation
    - Iterative processing
5. FastBit [Wu05]
    - Bitmap indexes

Design Philosophy Touch Stones [Lam11]

Storage

- Keep data sorted $\rightarrow$ reduce seeks, easy random entry
- Shard with access locality $\rightarrow$ minimize involved nodes
- Store data in columns $\rightarrow$ don't waste I/O
- Use append-only disk format $\rightarrow$ avoid expensive index updates

Compute

- Use disk appropriately $\rightarrow$ large sequential reads
- Trade CPU for I/O $\rightarrow$ type-specific, aggressive compression
- Use pipelined parallelism $\rightarrow$ hide latency
- Ship compute to data $\rightarrow$ aggregation serving tree

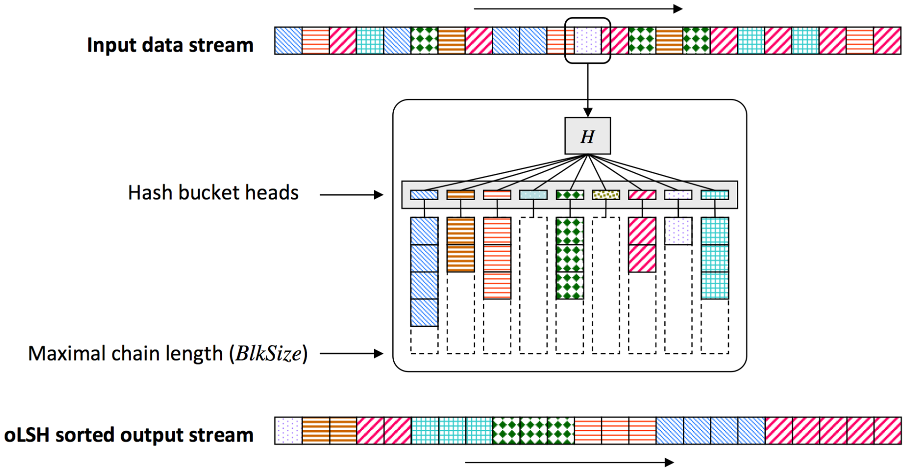
Query

- Make it user-friendly $\rightarrow$ declarative query interface
- Provide query hooks $\rightarrow$ support complex analysis

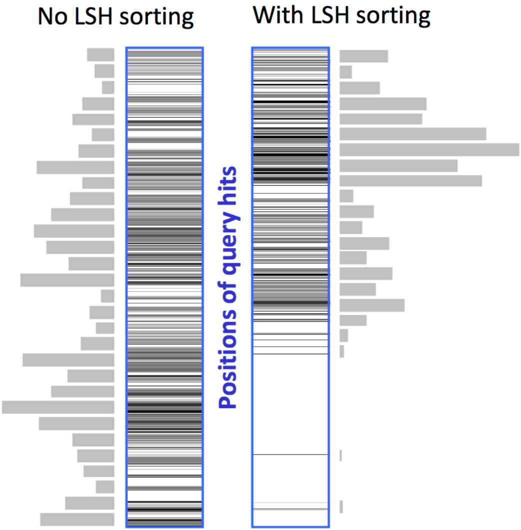
Taxonomy Grammar

```
taxonomy  ::= typdef* | event+
typedef   ::= name, type
event     ::= name, argument*, attribute*
argument  ::= name, type, attribute*
attribute ::= key, value?
type      ::= basic | domain | complex
basic     ::= bool | count | int | double | string
domain    ::= addr | port | subnet | time | interval
complex   ::= enum | vector | set | map | record
record    ::= argument+
```

Event Reordering in NET-FLi via Locality-Sensitive Hashing (oLSH) [FSV10]



Effect of LSH [FSV10]



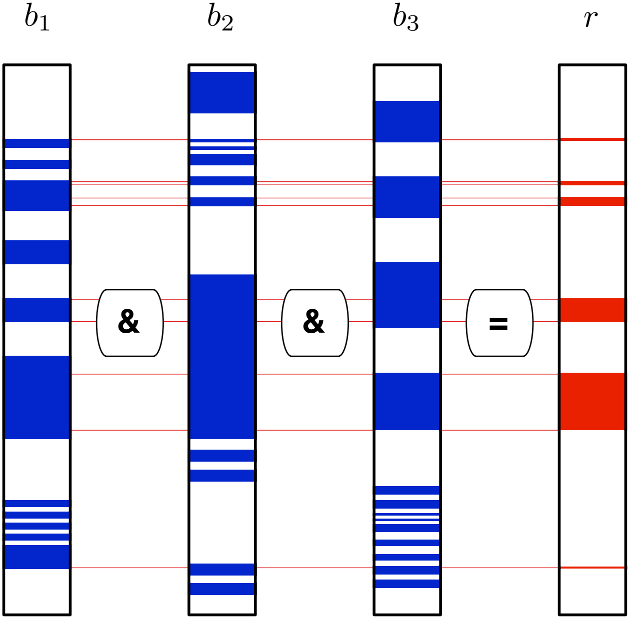
Query	SELECT	*
	WHERE	SrcIP = 10.20.30.40

Bitmap Indexes

- Column **cardinality**: # distinct values
- One bitmap b_i for each value i
- Sparse, but compressible
  - WAH [WOSN01]
  - COMPAX [FSV10]
  - CONSICE [CDP10]
- Can operate on compressed bitmaps
  - No need to decompress

Data		Bitmap Index b_0	b_1	b_2	b_3
2		0	0	**1**	0
1		0	**1**	0	0
2		0	0	**1**	0
0	→	**1**	0	0	0
0		**1**	0	0	0
1		0	**1**	0	0
3		0	0	0	**1**

Bitmap Index Operations



Bitmap Index Retrieval