Linux APT Detection

Souradip Ghosh, Dylan Kennedy, Lenin Estrada, Sherwin Shen

Outline

- Quick Recap
- Modifications
  - Recap --- Midterm
  - Propagation policies
  - Tag attenuation and tag decay
  - Alarm generation and real-time detection
- Results
- Future Work

Outline

- **Quick Recap**
- Modifications
  - Recap --- Midterm
  - Propagation policies
  - Tag attenuation and tag decay
  - Alarm generation and real-time detection
- Results
- Future Work

Quick Recap: MORSE

- A **tag-based** approach to detecting APTs in real time
  - An addition to the conventional provenance graph approach
- Builds **data tags** and **subject tags** to denote the events that occur with a node
- Defines propagation policies to track “suspiciousness”
  - What happens when a process reads or writes a file, etc?
  - **Tag decay** and **Tag attenuation** --- Propagating in a clever way

Quick Recap: Problem Statement

Problem: How can we build an real-time Endpoint Detection and Response (EDR) system on Linux that is both efficient and accurate?

Quick Recap: Problem Statement

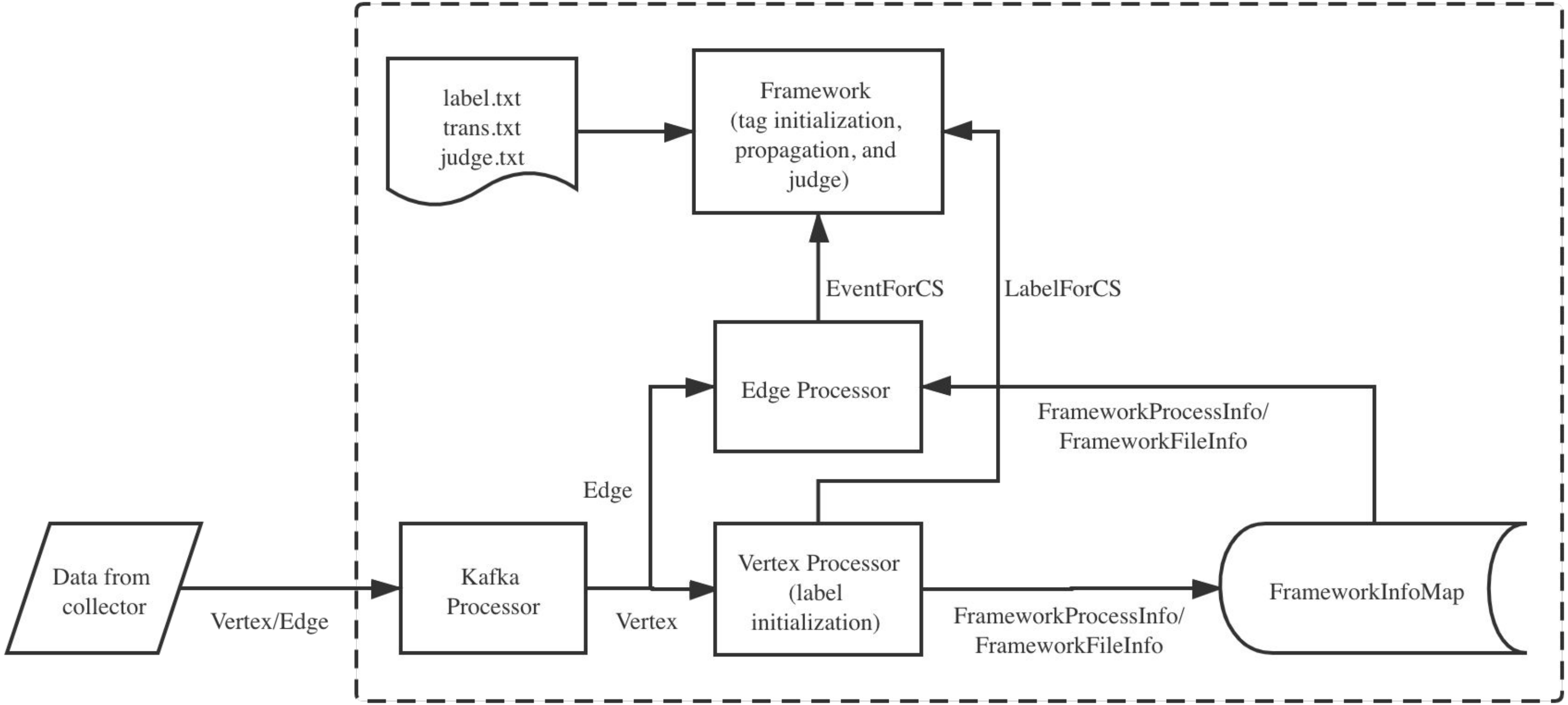
Problem: How can we build an real-time Endpoint Detection and Response (EDR) system on Linux that is both efficient and accurate?

We use the benefits of MORSE as inspiration --- efficient APT detection and reduction of false positives

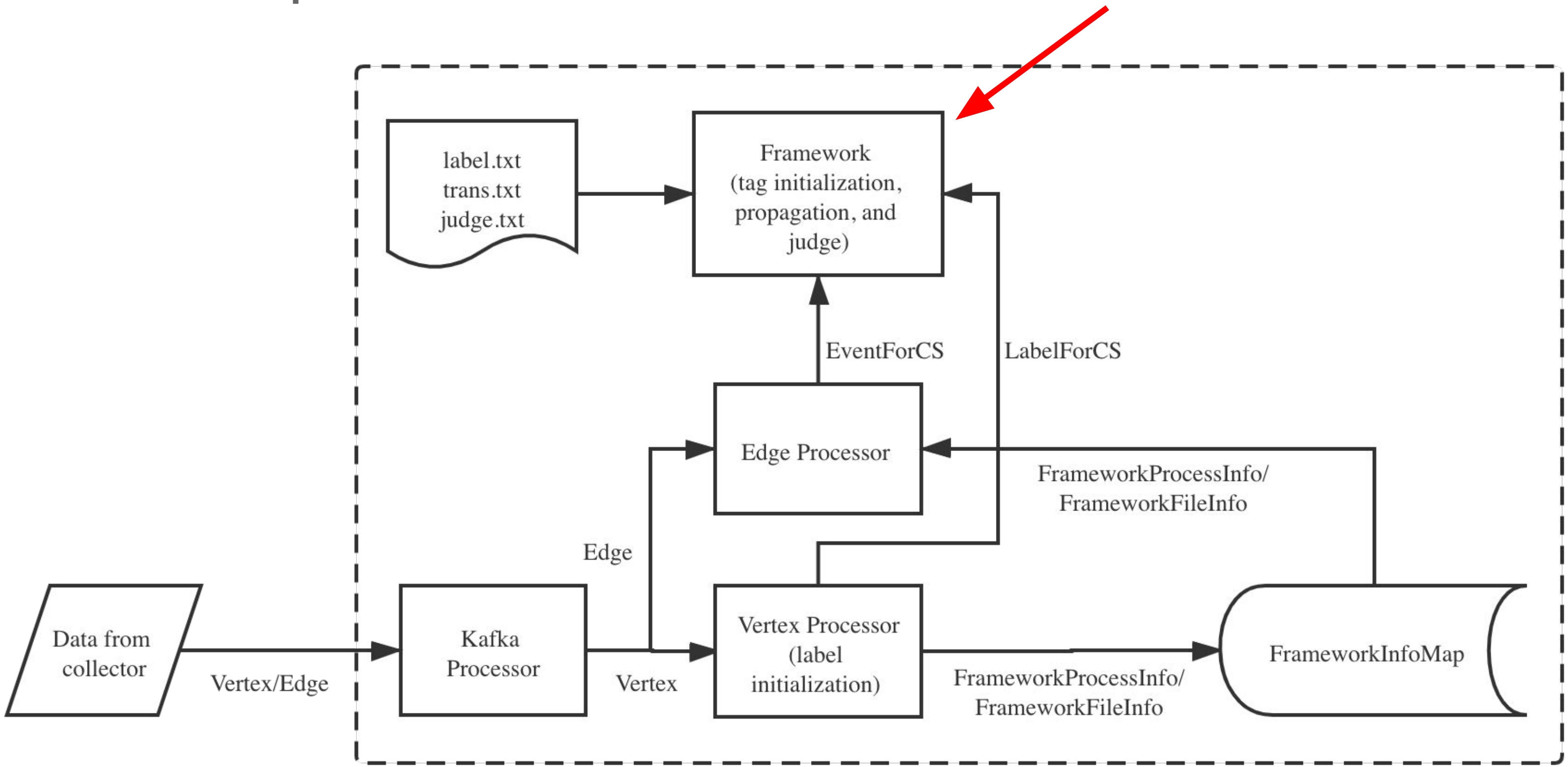
Quick Recap: Contributions

- Tag Initialization
  - Determined starting Data Tag values
- Tag Propagation
  - How these values transfer to children
- Tag Decay
  - Have suspicious processes slowly converge towards benign over time
- Judge Policies
  - Suppress false positive alerts from existing model based

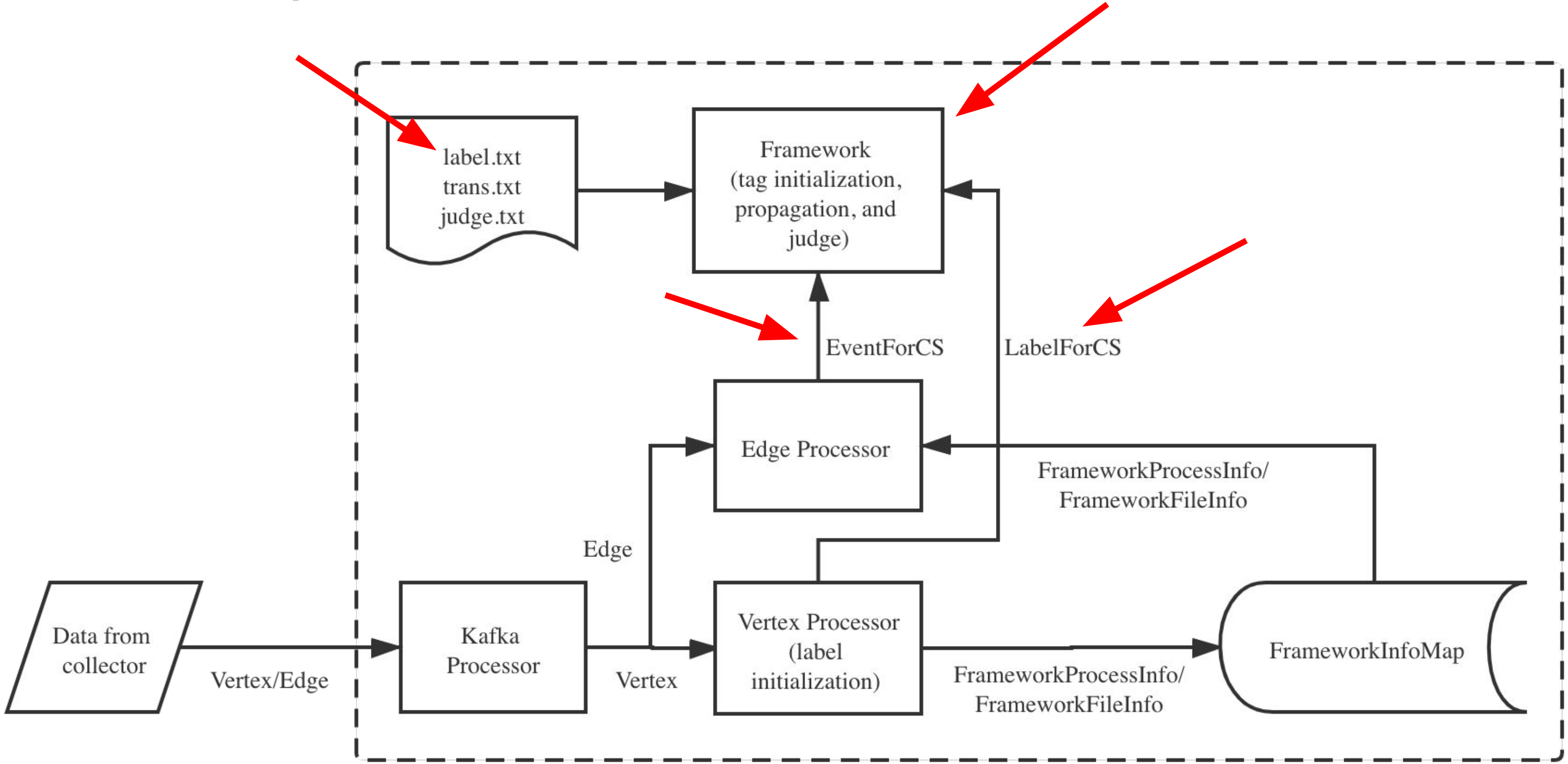
Quick Recap: Framework



Quick Recap: Framework



Quick Recap: Framework



Outline

- Quick Recap
- **Modifications**
  - Recap --- Midterm
  - Propagation policies
  - Tag attenuation and tag decay
  - Alarm generation and real-time detection
- Results
- Future Work

Outline

- Quick Recap
- **Modifications**
  - **Recap --- Midterm**
  - Propagation policies
  - Tag attenuation and tag decay
  - Alarm generation and real-time detection
- Results
- Future Work

Modification: Midterm Recap

- Developed tag decay for all nodes (including process and files)
- Wrote the most important propagation policies
- Picked arbitrary initialization and convergence values

```
public void decayBenignItag()
{
    float d = (float)(Math.pow(this.d_b, this.periods));
    this.itag = (this.init_itag * d) + ((1 - d) * this.T_qb);

    return;
}
```

```
if (event.getType() == EventType.FILE_CREATE) {
    fileNode.setCtag(processNode.getCtag());
    fileNode.setItag(processNode.getItag());
}
```

Outline

- Quick Recap
- **Modifications**
  - Recap --- Midterm
  - **Propagation policies**
  - Tag attenuation and tag decay
  - Alarm generation and real-time detection
- Results
- Future Work

Modification: Propagation Policies

Event	Tag to update	New tag value for different subject types		
		benign	suspect	suspect environment
create(s, x)	$x.dtag$	$s.dtag$		
read(s, x)	$s.dtag$	$min(s.dtag, x.dtag)$		
write(s, x)	$x.dtag$	$min(s.dtag + a_b, x.dtag)$	$min(s.dtag, x.dtag)$	$min(s.dtag + a_e, x.dtag)$
periodically:	$s.dtag$	$max(s.dtag,\ d_b{*}s.dtag{+}(1-d_b){*}T_{qb})$	no change	$max(s.dtag,\ d_e{*}s.dtag{+}(1-d_e){*}T_{qe})$

Event	Tag to update	New tag value for different subject types		
		benign	suspect	suspect environment
load(s, x)	$s.stag$	$min(s.stag, x.itag)$		
	$s.dtag$	$min(s.dtag, x.dtag)$		
exec(s, x)	$s.stag$	$x.itag$	$min(x.itag, susp\_env)$	$x.itag$
	$s.dtag$	$\langle 1.0, 1.0 \rangle$	$min(s.dtag, x.dtag)$	$min(s.dtag, x.dtag)$
inject(s, s')	$s'.stag$	$min(s'.stag, s.itag)$		
	$s'.dtag$	$min(s.dtag, s'.dtag)$		

"Tables I and II consider the main operations that propagate tags. Note that fork implicitly* copies the parent's tags to the child."

Modification: Propagation Policies --- Code Sample

```java
private void propTag(ProcessNode subjectNode, Node objectNode, EventForCS event)
{
    //...

    EventType eventType = event.getType();
    switch (eventType)
    {
        case FILE_CREATE:
            objectNode.setCtag(subjectNode.getCtag());
            objectNode.setItag(subjectNode.getItag());
            decayAndAttenuateDataTag(objectNode, event.getTimestamp());
            break;

        case FILE_OPEN:
        case FILE_READ:
            subjectNode.setCtag(Math.min(subjectNode.getCtag(),objectNode.getCtag()));
            subjectNode.setItag(Math.min(subjectNode.getItag(),objectNode.getItag()));
            decayAndAttenuateDataTag(subjectNode, event.getTimestamp());
            break;
```

Outline

- Quick Recap
- **Modifications**
  - Recap --- Midterm
  - Propagation policies
  - **Tag attenuation and tag decay**
  - Alarm generation and real-time detection
- Results
- Future Work

Modification: Tag Decay

- Decay *c* and *i* tags depending on environment

```
public void decayBenignItag()
{
    float d = (float)(Math.pow(DecAttValues.d_b, this.period));
    this.itag = (DecAttValues.init_itag * d) + ((1 - d) * DecAttValues.T_qb);

    return;
}

public void decayBenignCtag()
{
    float d = (float)(Math.pow(DecAttValues.d_b, this.period));
    this.ctag = (DecAttValues.init_ctag * d) + ((1 - d) * DecAttValues.T_qb);

    return;
}
```

Modification: Tag Decay

- Decay *c* and *i* tags depending on environment

```
public void decaySuspiciousItag()
{
    float d = (float)(Math.pow(DecAttValues.d_e, this.period));
    this.itag = (DecAttValues.init_itag * d) + ((1 - d) * DecAttValues.T_qe);

    return;
}

public void decaySuspiciousCtag()
{
    float d = (float)(Math.pow(DecAttValues.d_e, this.period));
    this.ctag = (DecAttValues.init_ctag * d) + ((1 - d) * DecAttValues.T_qe);

    return;
}
```

Modification: Tag Attenuation

- Additive approach
- Use min to extract the most confidential (and lowest integrity) from the data contained within

```
public Float getAttBCtag() { return ctag + DecAttValues.a_b; }
public Float getAttBItag() { return itag + DecAttValues.a_b; }
public Float getAttECtag() { return ctag + DecAttValues.a_e; }
public Float getAttEItag() { return itag + DecAttValues.a_e; }
```

```
public void attenuateBenignCtag() { this.ctag = Math.min(this.getAttBCtag(), 1f); }
public void attenuateBenignItag() { this.itag = Math.min(this.getAttBItag(), 1f); }
public void attenuateSuspCtag() { this.ctag = Math.min(this.getAttECtag(), 1f); }
public void attenuateSuspItag() { this.itag = Math.min(this.getAttEItag(), 1f); }
```

Modification: Mappings

- Tune *c* and *i* tag values from existing values based on label description

```
public static final HashMap<Float, subjectTag> subjectTags = new HashMap<Float, subjectTag>() {{
    put(0.25f, subjectTag.suspiciousCode);
    put(0.5f, subjectTag.suspiciousEnvironment);
    put(0.75f, subjectTag.benign);
    put(1f, subjectTag.trusted);
}};
```

```
put(LabelType.PT45,0.15f);
put(LabelType.PT46,0.15f);
put(LabelType.PT47,0.15f);
put(LabelType.PT48,0.15f);
```

```
PT45,1,1,PHF,The process wrote files into /etc/systemd/system
PT46,1,1,PHF,The process modified .bash_profile or .bashrc
PT47,1,1,PHF,The process read /etc/passwd
PT48,1,1,PHF,The process read several files which may contain password policy
```

Modification: Handling Suspicious Nodes

- A suspicious node will decay and attenuate **much slower** than benign nodes
- What if a node that had decayed to benign status experiences a suspicious event again?
  - Node's data/subject tags are reset to the values corresponding to event
  - Accomplished by parsing new EventForCS and LabelForCS objects
- A node will remain suspicious if it's data/subject tag values range below **0.5**

Outline

- Quick Recap
- **Modifications**
  - Recap --- Midterm
  - Propagation policies
  - Tag attenuation and tag decay
  - **Alarm generation and real-time detection**
- Results
- Future Work

Modification: Alarm Generation

- **Previously** --- an alarm will be generated when the labels aggregated on a node satisfy certain conditions
  - If a process or its ancestors had network connection and the process read some sensitive files, an alarm is generated
- **Now** --- we impose another prerequisite on alarm generation: the process is not benign
  - The *i* tag has to be below 0.5 to decrease number of the false positives

Modification: Real-time Detection

- EDR mimics real-time detection by setting a **minimum granularity** for time between events on the same node
  - Decay function exponentiates at a multiple of this granularity
  - Currently set to **100 nanoseconds** --- a promising granularity for the DARPA data set

```
// Set up for timestamps/counter
this.counter_interval = DecAttValues.init_interval;
this.period = 0;
this.last_timestamp = 0;
```

Outline

- Quick Recap
- Modifications
  - Recap --- Midterm
  - Propagation policies
  - Tag attenuation and tag decay
  - Alarm generation and real-time detection
- **Results**
- Future Work

Results: Methodology

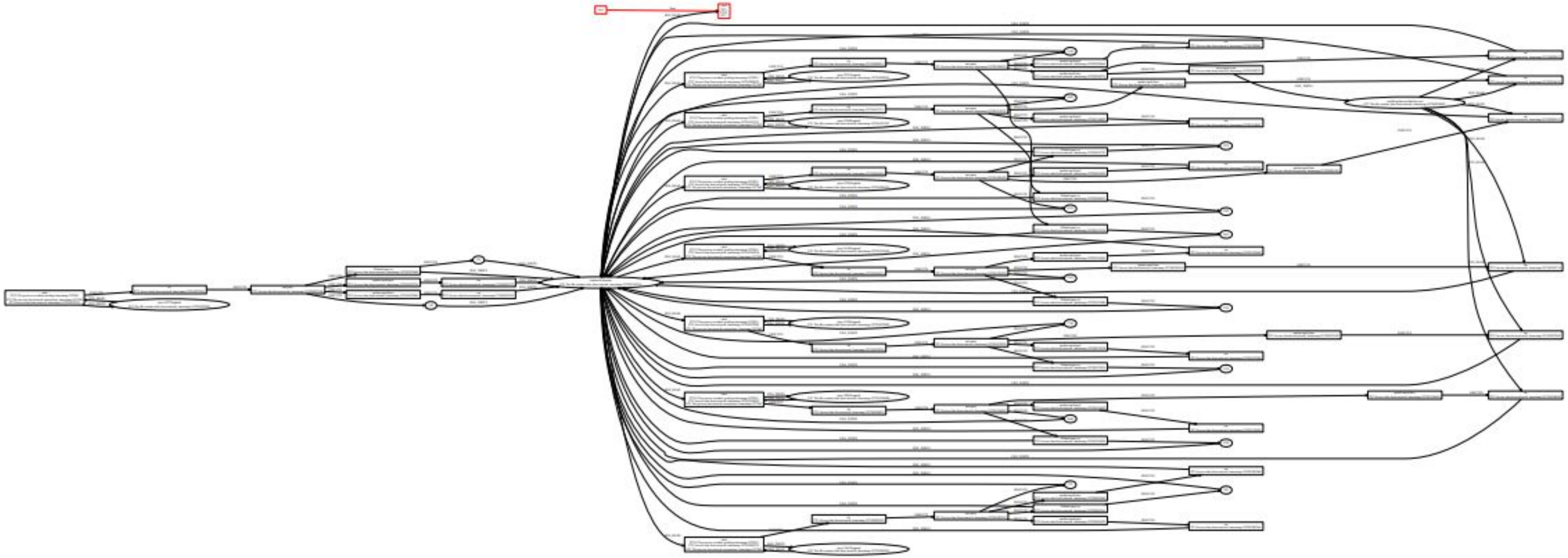
- Testing data sliced from the **DARPA data set**
  - 3.5 hours of trace data, covering a **privilege escalation** attack
- Our EDR parses and analyzes the data offline, generates alarms and provenance graphs once malicious behaviors are detected

Results: Our Progress

- Currently have two (slightly) different versions of the framework that include tag **decay**, tag **attenuation**, and **propagation**
- Both have the following:
    - Initialization values --- iTag = 0.51, cTag = 0.51, sTag = 0.75
    - Convergence values --- iTag = 0.75, cTag = 0.75
- Differences:
    - More specific about handling suspicious nodes
    - Capping convergence for *i* and *c* tags (to prevent undefined behavior)

Results: Alarm Generation

- Original framework:
  - 708 alarms
- Our framework:
  - Version 1: **267** alarms --- **62% decrease**
  - Version 2: **650** alarms --- **8% decrease**
- Accuracy --- all three frameworks generate alarms for the true positive/ground truth
  - `pid_4601_sshd_uuid_4525 PT3,PT1,PT33`



[Full Size]

Results: Space Complexity

- EDR efficiency relies on memory usage --- smaller node objects
  - Faster to parse and analyze
  - Memory usage is lower
  - Increases runtime performance

```
// data tag implementation --- consists of <c, i>. No wrapper data structure used to save memory.
private Float ctag;
private Float itag;

// propagation fields
private long counter_interval; // length of time that defines a period (i.e. 10 seconds, etc.)
private int period; // number of periods accumulated
private long last_timestamp; // last recorded timestamp for the node
```

Outline

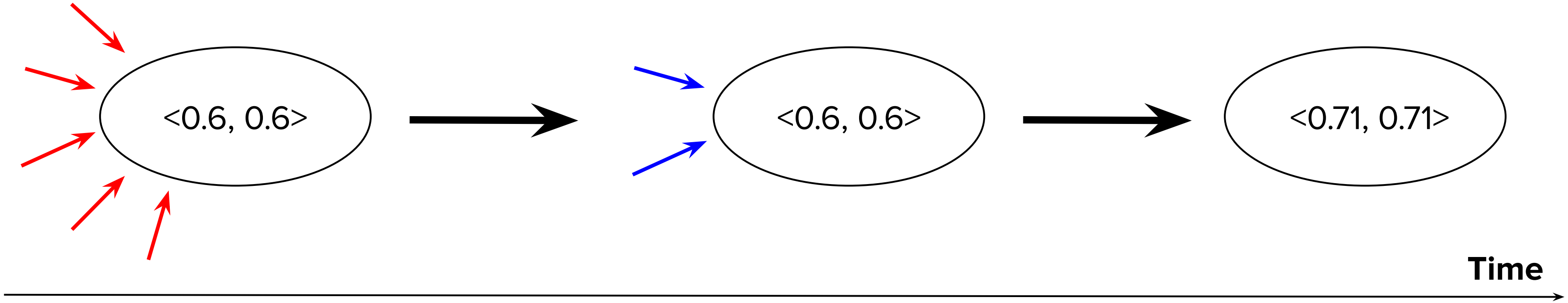
- Quick Recap
- Modifications
  - Recap --- Midterm
  - Propagation policies
  - Tag attenuation and tag decay
  - Alarm generation and real-time detection
- Results
- **Future Work**

Future Work: “Hotness”

- Motivation --- Why?
  - Decaying and/or attenuating too early leads to a loss of accuracy
  - Some APTs are very drawn out, some have bursts of malicious activity
- We want to control **when** we decay and attenuate

Future Work: “Hotness”

- “Hotness” metric is a solution to control decay/attenuation:
  - Each node will **not** be decayed and/or attenuated if the node is visited very frequently in a short amount of time
  - We want to keep the node to maintain accuracy of the events even though it may end up being benign



Future Work: Tuning Convergence/Init Values

- If initialization and convergence values for data tags, decay, and attenuation are inaccurate --- results can be inaccurate
  - Can lead to quick decay --- not enough and often false alarms
  - Can lead to enlarged provenance graphs
- Accuracy of our system is **directly dependent** on these values

Future Work: Machine Learning

- Machine learning methods are an idea to tune the values **accurately** and **automatically** for a particular environment
- **Pros**: Given parameters and the detection environment, ML models can tune initialization and convergence values correctly
- **Cons**: Real-time detection can slow down if a model takes time to determine the values first

Future Work: Runtime Efficiency

- APT detection tools need to be quick --- our EDR should have better analysis and propagation time
- Requires refactoring for runtime efficiency
  - Reducing the usage of floats
  - Using a more efficient algorithm for exponentiation --- decay
  - Handling propagation --- changing design patterns

Overall Contributions

- Tag Initialization
  - Determined starting Data Tag values based on 41 different labels
- Tag Propagation
  - How these values transfer to children based on 8 different events and 4 different Subject Tags
- Tag Decay
  - Have suspicious processes slowly converge towards benign over time
- Judge Policies
  - Suppress false positive alerts (up to 62%) from existing model

References

- Our codebase: https://github.com/nbshenxm/CS450_project (private, branch: “develop”)
- M. N. Hossain, S. Sheikhi, R. Sekar, “Combating Dependence Explosion in Forensic Analysis Using Alternative Tag Propagation Semantics,” in Proc. USENIX Secur., 2018, pp. 1723-1740.
- M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. D. Stoller, and V. Venkatakrishnan, “Sleuth: Real-time attack scenario reconstruction from cots audit data,” in Proc. USENIX Secur., 2017, pp. 487–504.
- S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, “Holmes: real-time apt detection through correlation of suspicious information flows,” arXiv preprint arXiv:1810.01594, 2018.