ORACLE

Runtime Prevention of Deserialization Attacks

François Gauthier and Sora Bae
Oracle Labs Australia

Problem and Proposed Solution

- Untrusted deserialization exploits, where a serialised object graph (i.e. gadget chain) is used to achieve denial-of-service or arbitrary code execution were introduced in the 2017 OWASP Top 10 and merged in the broader "injection" category in the 2021 version.

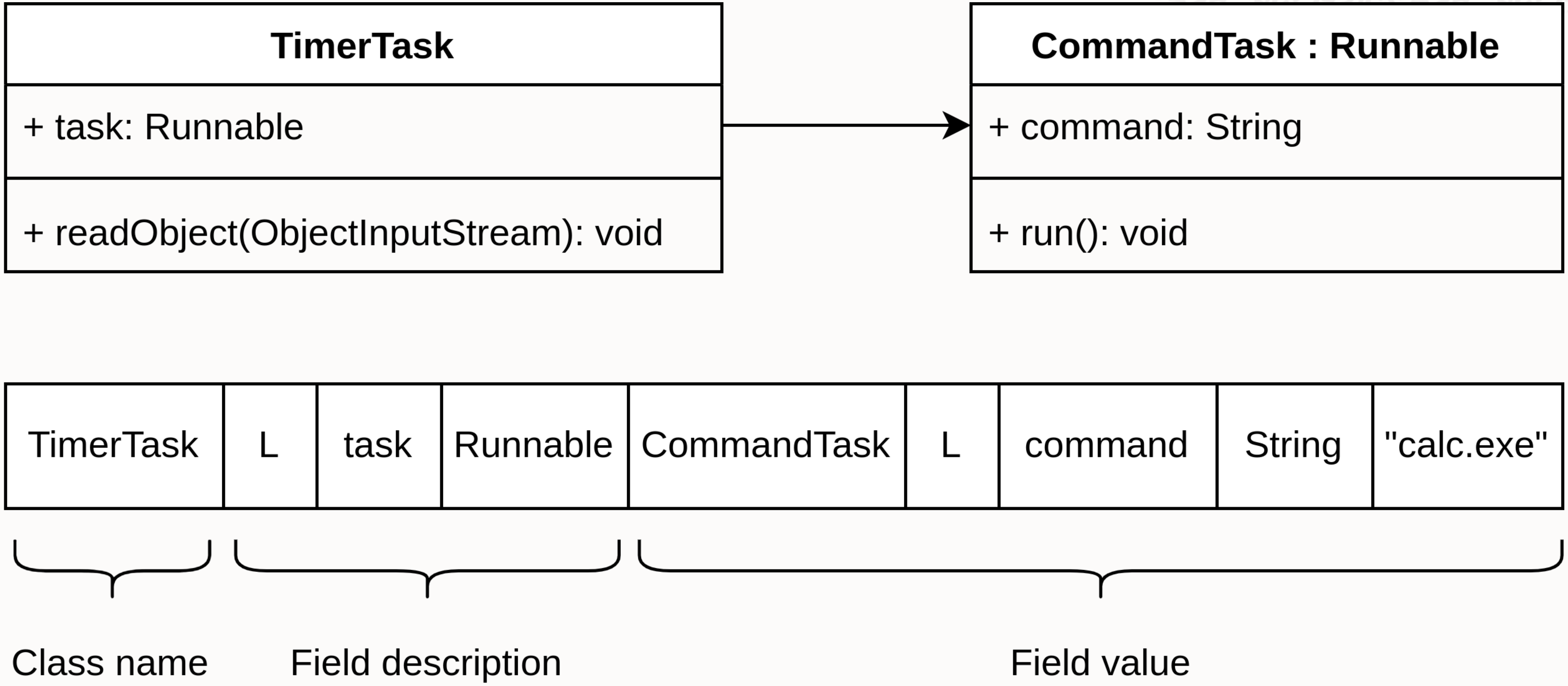
- State-of-the-art approaches (e.g. JDK[1] deserialization filters), ask developers to block or allow classes individually, without any context, despite the sequential nature of gadget chains.

- We show how Markov chains can help detect sequences of features that are typical of gadget chains to detect and prevent deserialization attacks.

[1]: JDK is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

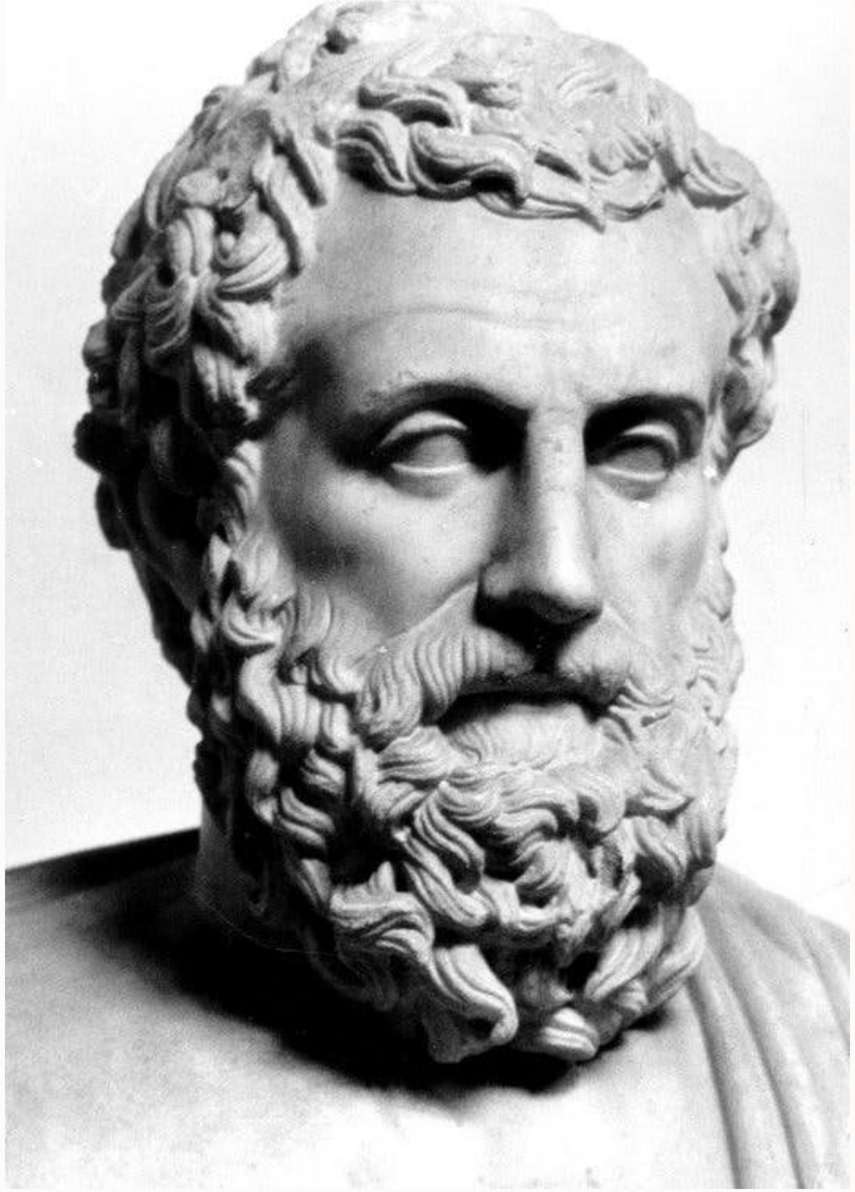
Copyright © 2022, Oracle and/or its affiliates

The Java[1] Serialization Format



[1]: Java is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Copyright © 2022, Oracle and/or its affiliates

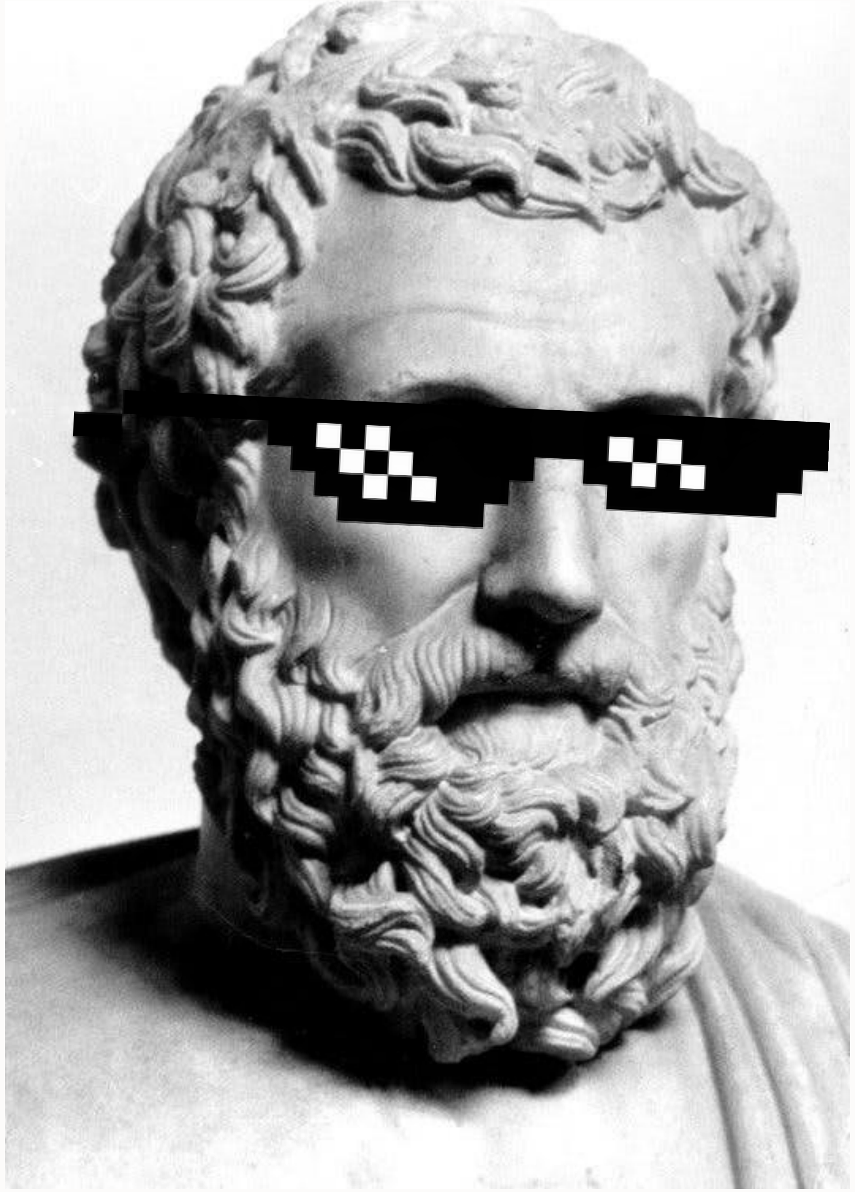


What Makes A Good Gadget Chain?

"A whole is what has a beginning and middle and end"

Aristotle, Poetics (335 BCE)

Copyright © 2022, Oracle and/or its affiliates



What Makes A Good Gadget Chain?

Aristotle got it right. A good gadget chain needs:

- A method that will be called at the *beginning* of deserialisation to hand control to…
- Linker classes in the *middle* that will set the scene for…
- The target method that will be invoked at the *end* of deserialisation.

Aristotle, Poetics (335 BCE)

Copyright © 2022, Oracle and/or its affiliates

Hypothesis

The sequence of classes and their features differ significantly between benign and malicious deserialization chains.

Copyright © 2022, Oracle and/or its affiliates

Modelling Java Deserialisation as Markov Chains (1)

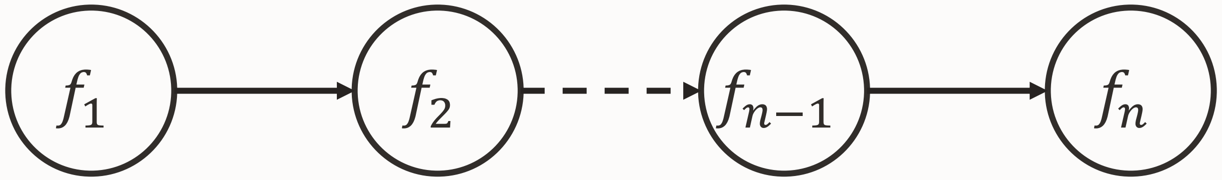
Given a class C and a set of Boolean class features F, we can abstract C to a set of Boolean features f:

$$C \rightarrow f, f \in P(F)$$

Similarly, we can abstract any gadget chain G as a sequence of feature sets:

$$G \rightarrow (f_1, f_2, \cdots, f_{n-1}, f_n), f_i \in P(F)$$

Graphically:



Copyright © 2022, Oracle and/or its affiliates

Features - From Manual Review of `ysoserial` Gadget Chains

Id	Feature	Description
1	Uses reflection	True if the class calls any of the following from `java.lang.reflect`: - `Constructor.newInstance()` - `Field.set()` - `Method.invoke()`
2	Overrides readObject	True if the class overrides the method `Object readObject(ObjectInputStream ois)`
3	Overrides hashCode	True if the class overrides the `int hashCode() method`.
4	Has generic field	True if the class has a field of any of the following type: - `java.lang.Object` - `java.lang.Comparable` - `java.util.Comparator`
5	Implements Map	True if the class implements the `java.util.Map` interface.
6	Implements Comparator	True if the class implements the `java.util.Comparator` interface.
7	Calls hashCode	True if the class calls any of the following methods: - `int java.util.Objects.hash(Object... values)` - `int java.util.Objects.hashCode(Object o)` - `*.hashCode()`
8	Calls compare	True if the class calls any of the following methods: - `*.compare()` - `*.compareTo(...)`

Copyright © 2022, Oracle and/or its affiliates

Modelling Java Deserialisation as Markov Chains (2)

A Markov chain represents a system that:

- Has a finite number of states:

$$S = \{s_1, s_2, \cdots, s_n\}$$

- Starts in any given state $s \in S$ with probability $p_i \in p_{init}$, its initial state probability vector:

$$p_{init} = (p_1, p_2, \cdots, p_n)$$

- Transitions between states with some probability p at each step t:

$$p_{tr} = \begin{pmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{pmatrix}$$

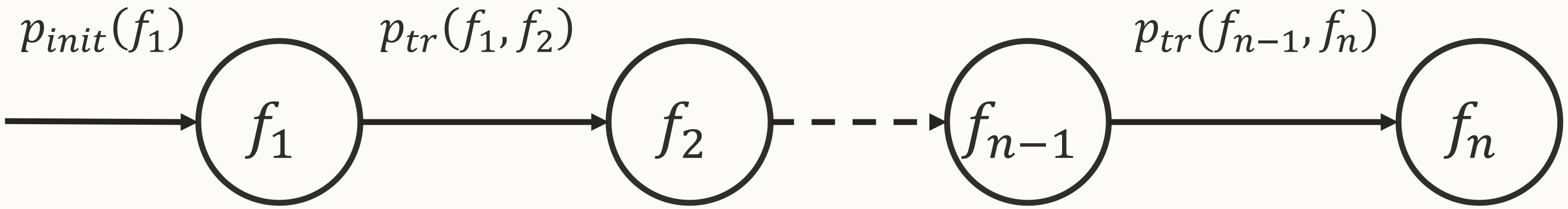
Copyright © 2022, Oracle and/or its affiliates

Modelling Java Deserialisation as Markov Chains (3)

Given a Markov chain and an observed sequence of states, one can estimate the probability that the chain generated the sequence with a simple product of probabilities:

$$p(f_1, f_2, \cdots, f_n) = p_{init}(f_1) \cdot \prod_{i=2}^{n} p_{tr}(f_{i-1}, f_i)$$

Graphically:



 Copyright © 2022, Oracle and/or its affiliates

Estimating Transition Probabilities From Data

Our goal is to build two Markov chains B and M from benign and malicious datasets respectively.

Empirical approach:

Use the observed transition probabilities directly. Works well with large dataset.

Bayesian approach:

Model each row of p_{tr} as the output of a known probability distribution (i.e. Dirichlet) and explore the space of Dirichlet parameters $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_K)$ to find the distribution**s** $Dir(\alpha_i)$ that best characterize the data.

Copyright © 2022, Oracle and/or its affiliates

Black magic involving probabilistic programming (Bayesian inference), Dirichlet distributions, and Markov Chain Monte Carlo sampling…

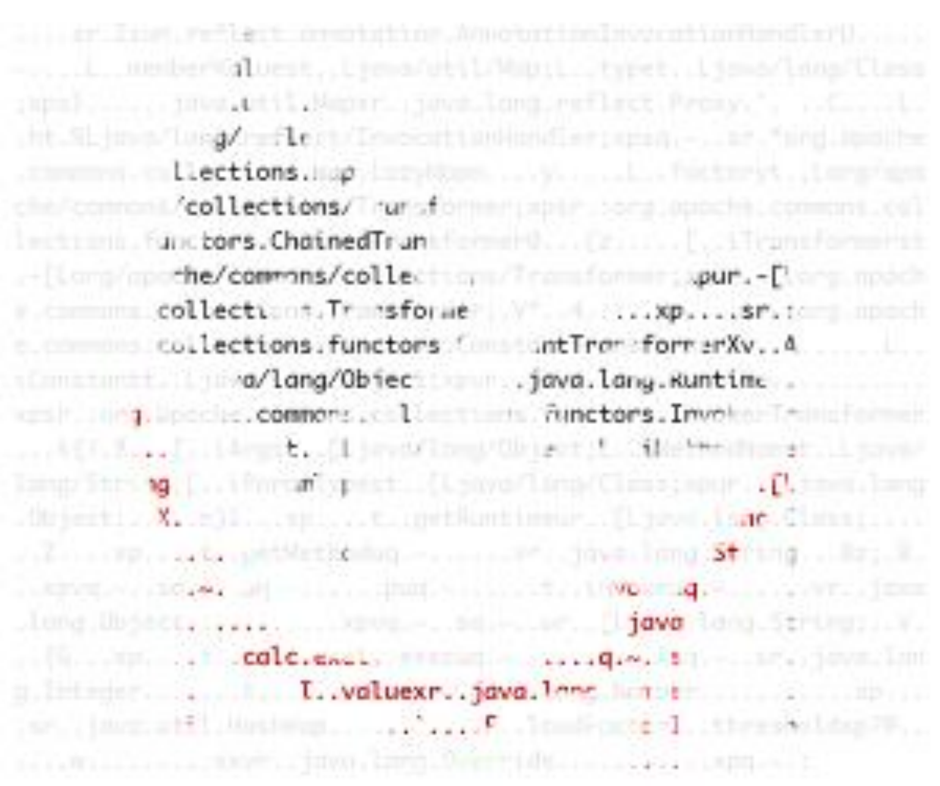
Output: Two sets of benign (B) and malicious (M) Markov chains

Copyright © 2022, Oracle and/or its affiliates

Creating A Deserialization Dataset (1)

`Ysoserial` is a public repository of deserialization gadget chains.

1. Create an ASM agent to dynamically extract features from loaded classes.
2. Build a harness to (de)serialise ysoserial payloads, and extract features.



Copyright © 2022, Oracle and/or its affiliates

Creating A Deserialization Dataset (2)

ORACLE
WebLogic Server

WebLogic Server (WLS)[1] heavily relies on deserialisation for common operations.

1. Instrument WebLogic Server with the ASM agent mentioned above.
2. Load WLS and exercise its console to extract features from “benign” deserialization chains .

[1]: Oracle®WebLogic Server is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Copyright © 2022, Oracle and/or its affiliates

Dataset Description

	Unique chains	Average length	Median length
Benign (WLS)	227	38.96	13
Malicious (`ysoserial`)	37	16.68	6

Copyright © 2022, Oracle and/or its affiliates

Detecting Deserialization Attacks

Input: $\mathcal{B}$, $\mathcal{M}$, **t**, **l**

Output: status $\in$ {`accepted`, `rejected`, `undecided`}

$seq \leftarrow$ new List()

Function *MarkovFilter(class, end)*:

  $features \leftarrow \textsc{ExtractFeatures}(class)$

  seq.append($features$)

  $\overline{P_{\mathcal{B}}} \leftarrow mean(P(seq \mid \mathcal{B}))$

  $\overline{P_{\mathcal{M}}} \leftarrow mean(P(seq \mid \mathcal{M}))$

  $disjoint \leftarrow ((\overline{P_{\mathcal{B}}} \pm \mathbf{t}\sigma) \cap (\overline{P_{\mathcal{M}}} \pm \mathbf{t}\sigma) = \emptyset)$

  if end **and** $disjoint$ **then**

    return $\overline{P_{\mathcal{M}}} > \overline{P_{\mathcal{B}}}$? `rejected` : `accepted`

  else if end **and** $\neg disjoint$ **then**

    return `rejected`

  else if $disjoint$ **and** $|seq| \geq$ **l** **and** $\overline{P_{\mathcal{M}}} > \overline{P_{\mathcal{B}}}$ **then**

    return `rejected`

  else

    return `undecided`

  end

end

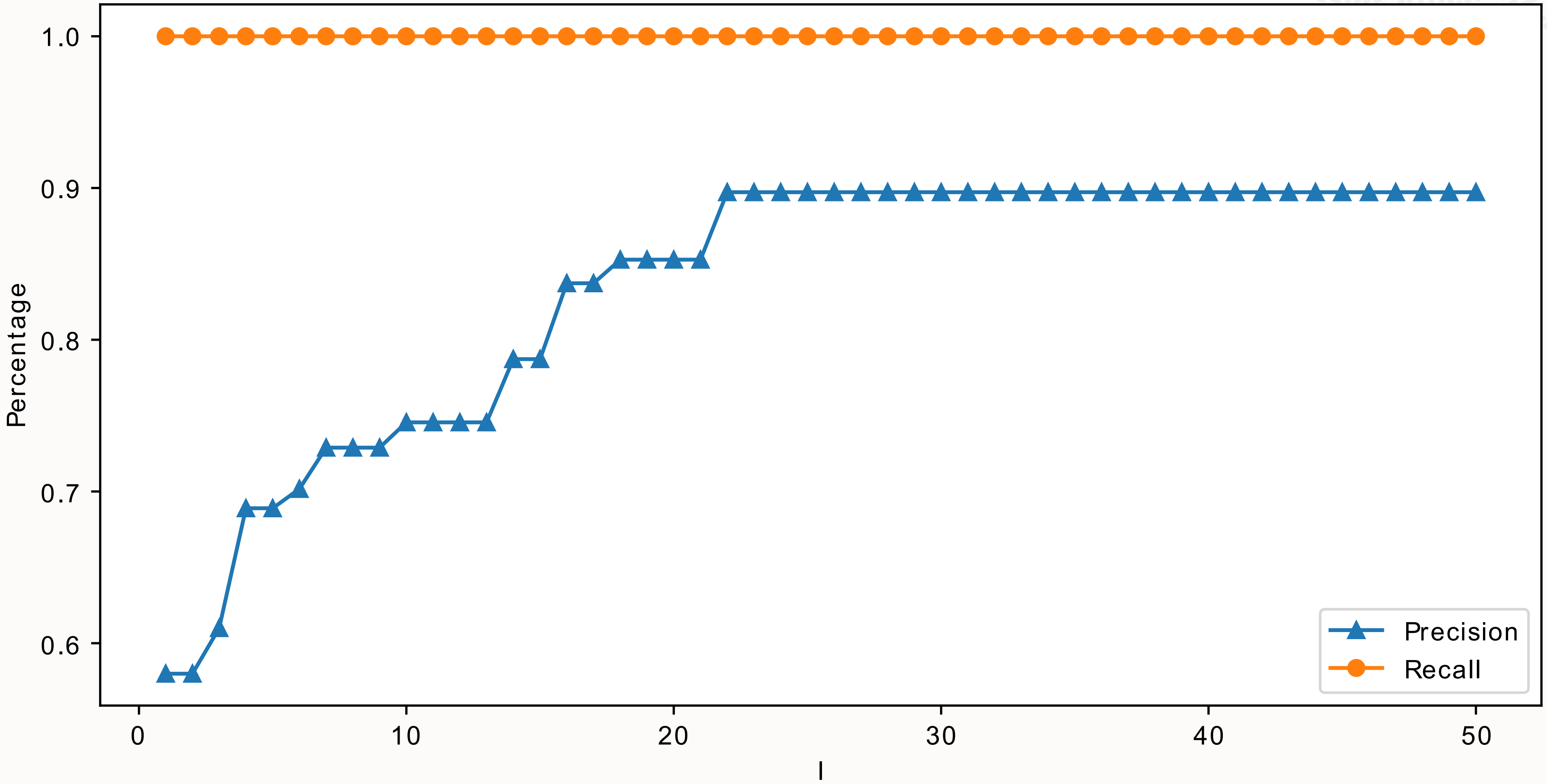
Copyright © 2022, Oracle and/or its affiliates

Bayesian vs. Empirical, 5-Fold Cross-Validation

	t	Precision	Recall	F1-score	Time (sec)
Bayesian	0	91.67±6.97	96.67±6.67	0.94±0.03	7163
	1	91.67±6.97	96.67±6.67	0.94±0.03	
	2	89.72±8.94	100.0±0.00	0.94±0.05	
	3	88.17±11.26	100.0±0.00	0.93±0.07	
Empirical	—	72.95±14.27	100.0±0.00	0.84±0.09	0.7

Copyright © 2022, Oracle and/or its affiliates

Impact of Aborting Deserialization Early



 Copyright © 2022, Oracle and/or its affiliates

Digging Into False Positives

After manual review, all the false positives fall into one of these categories:

- Java Transaction API
- Networking
- Instrumentation
- Remote Method Invocation (RMI)
- Managed Bean

which indeed look like attractive targets for deserialization attacks.

Copyright © 2022, Oracle and/or its affiliates

Thank you

francois.gauthier@oracle.com
sora.bae@oracle.com

https://labs.oracle.com/

 Copyright © 2022, Oracle and/or its affiliates