University of Basel

DMI HPC

Using Malware Detection Techniques for HPC Application Classification

Thomas Jakobsche, Florina M. Ciorba
University of Basel, Switzerland

hpc.dmi.unibas.ch | dmi-hpc@unibas.ch | dmi-hpc.bsky.social | hpc-dmi-unibas
since 08/2015

17 November 2024 - SC24 - Workshop on Cyber Security in High Performance Computing (S-HPC'24) - Atlanta, Georgia, US

Thomas Jakobsche

Florina M. Ciorba

Why are we here today?

Problem Statement

- ➢ **HPC admins and researchers do not actually know what users are executing**
- Deviation from allocation purpose and/or terms-of-use
- Security and compliance issues, waste and misuse of HPC resources

Motivation

- ➢ **HPC admins and researchers would benefit from workload identifiers**
- **Admins:** Ensure the efficient and secure use of resources
- **Researchers:** Focus optimization efforts and future system design

Current Limitations

- ➢ **No retention of reliable information about workload identity & characteristics**
- Group accounting and allocation purpose are insufficient
- Job names and user-compiled executable names are unreliable (e.g., my_job and a.out)

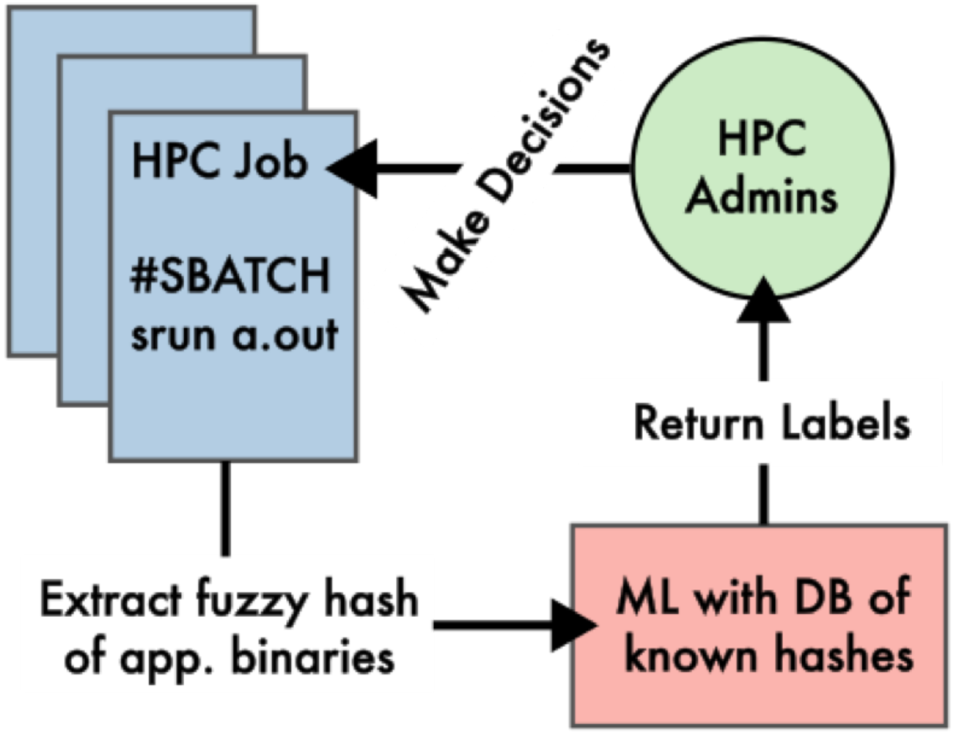
Our Proposed Solution

- ➢ **Provide access to application labels for HPC admins and researchers**
- Classifying application executables through **supervised ML**
- Using **fuzzy hash** features inspired by malware detection techniques

Context and Proposed Workflow

HPC application executables

- Application executables are the core of HPC jobs
- Different versions of the same application in circulation
- Executables can be accessed through Slurm's prolog



Guiding Questions

Is an application **similar to** a set of applications that

- A user or their group usually execute?
- Are normally used for a specific allocation?
- Are not allowed on the system at all?

Methodology
Dataset of Application Executables

Ground Truth Labels

- Evaluate classification of application executables through a dataset with ground truth labels

Scanning the Software Stack

- Bash script to collect ELF files from the pre-installed software stack on sciCORE*

The Dataset

- 92 application classes and 5’333 samples (executables)
- Splitting 2’688 for the training set and 2’645 for the test set
- The test includes 852 samples from completely unseen classes

Class Label	Version	Sample
/OpenMalaria	/46.0-iomkl-2019.01	/bin/openMalaria
/OpenMalaria	/43.1-foss-2021a	/bin/openMalaria
/Velvet	/1.2.10-goolf-1.4.10	/bin/velveth
...	...	...

**sciCORE, computing center at the University of Basel*
https://scicore.unibas.ch

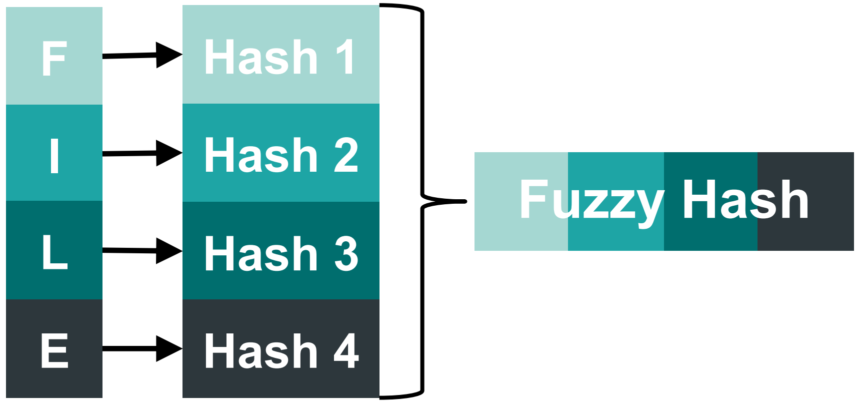
OpenMalaria: A simulator of malaria epidemiology and control

Methodology
Using Features inspired by Malware Detection Techniques

Hashing in Malware Detection
- Quickly classify malware with malware databases

Cryptographic hashes (e.g., SHA) for exact file matching
- *"Avalanche effect"* small changes → extreme differences

Fuzzy hashes (e.g., SSDeep) to match partial file similarities
- Detect variants of files with minor changes
- Can cover differences in compilation and code
- In this work: raw file, **strings** (characters), **nm** (symbols)



We use SSDeep to generate and combine hashes for file segments

Application	Version	sha256sum of symbols	SSDeep hash of symbols (in our work)
OpenMalaria	46.0-iomkl-2019.01	b33e2f1af03dedcb1e7bd2046d8c046e 9a56a0970ec0775126ef98a9fc3a7f52	1536:z5ujB2ip**prvzwz**K8l**8lPRCuN0L830XmR8c/dGS pTWK5f5Kuy1a**zM/M3rw**83**rw**La6Ftl**jyx:C5ujBf**Qz**r
	43.1-foss-2021a	96b5640230e0079367091258be34968 1f14867c5ff91ee747ea915ee339854f6	1536:3bn92z**prvzwz**e**8lPRCuN0L830XmR8c/dGS pTWK5f5Kuy1a**OMP**83**rF**La6Ftl**DJlzu:3bn9u**Qz**Y

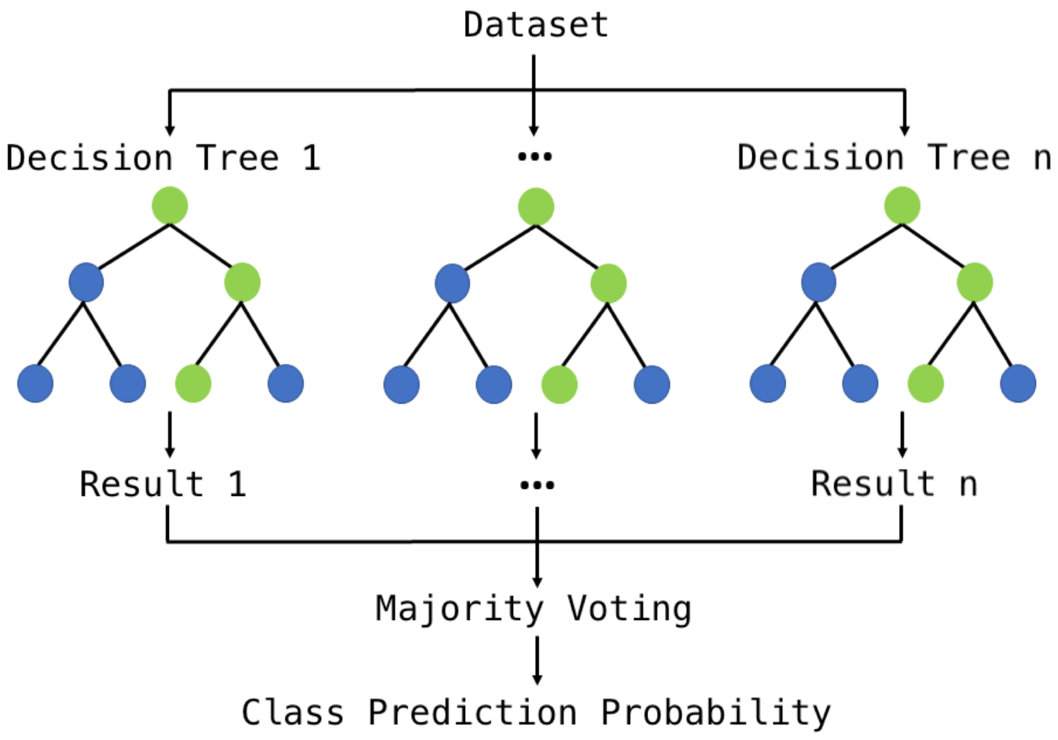
Methodology
New Fuzzy Hash Classifier for Application Executables

Fuzzy Hash Classifier: based on multiple Decision Trees (aka Random Forest RF) of Scikit-Learn

- **Non-linearity:** RF capture non-linear similarity of fuzzy hashes (which are not explicit Euclidean distances)

- **Confidence Threshold:** We enable the classifier to predict “unknown” if the class prediction probability is below a threshold (tuned on the training set)

  For example, if the model predicts "OpenMalaria" with a 65% probability, but the confidence threshold is set at 70%, the prediction will be labeled as "unknown."



Adapted from: https://de.wikipedia.org/wiki/Random_Forest

Results
Evaluation and the Classification Report

$$f1\ score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

89% micro f1-score
treating all samples equally

90% macro f1-score
averaging results per class

90% weighted f1-score
class-weighted averaging

Class	Precision	Recall	f1-Score	Support
"unknown"	0.92	0.75	0.83	852
Cell-Ranger	0.39	0.89	0.54	28
CellRanger	0.79	0.95	0.86	20
CapnProto	1.00	1.00	1.00	1
FSL	1.00	1.00	1.00	351
JAGS	1.00	1.00	1.00	1
kentUtils	1.00	0.99	0.99	352
...	...	...	...	...
micro avg	**0.89**	**0.89**	**0.89**	**2645**
macro avg	**0.92**	**0.92**	**0.90**	**2645**
weighted avg	**0.92**	**0.89**	**0.90**	**2645**

Results
A Closer Look at the Classification Report

Predicting the "unknown" class

- Confidently predicting a sample as "unknown" but not catching all "unknown" samples

Label noise in the dataset

- Some noise in the labels of the pre-installed applications (versions in different directories)

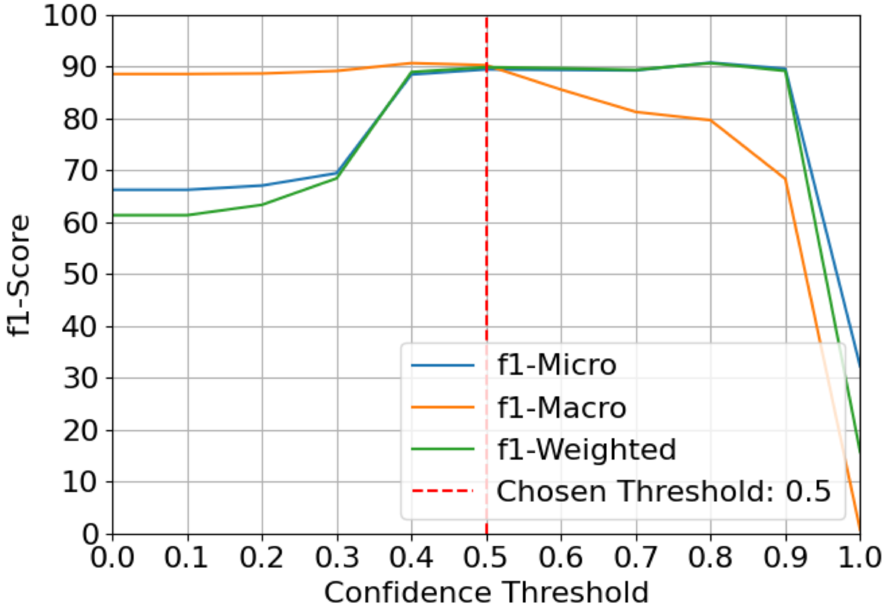
High application class imbalance

- Balanced weights assigned to classes, inversely proportional to sample counts

Class	Precision	Recall	f1-Score	Support
"unknown"	0.92	0.75	0.83	852
Cell-Ranger	0.39	0.89	0.54	28
CellRanger	0.79	0.95	0.86	20
CapnProto	1.00	1.00	1.00	1
FSL	1.00	1.00	1.00	351
JAGS	1.00	1.00	1.00	1
kentUtils	1.00	0.99	0.99	352
...	...	...	...	...
micro avg	**0.89**	**0.89**	**0.89**	**2645**
macro avg	**0.92**	**0.92**	**0.90**	**2645**
weighted avg	**0.92**	**0.89**	**0.90**	**2645**

Results

Confidence Threshold for “unknown” Prediction



Class	Precision	Recall	f1-Score	Support
“unknown”	0.92	0.75	0.83	852
Cell-Ranger	0.39	0.89	0.54	28
CellRanger	0.79	0.95	0.86	20
CapnProto	1.00	1.00	1.00	1
FSL	1.00	1.00	1.00	351
JAGS	1.00	1.00	1.00	1
kentUtils	1.00	0.99	0.99	352
…	…	…	…	…
micro avg	**0.89**	**0.89**	**0.89**	**2645**
macro avg	**0.92**	**0.92**	**0.90**	**2645**
weighted avg	**0.92**	**0.89**	**0.90**	**2645**

No configuration achieving more than ~90%

- Limitations of the current features + model
- Noise and inaccurate labels in the dataset

Discussion
Implications of Feature Importance

Comparing raw file content, strings, and symbols

- Raw file content and printable characters almost always change with compiler version and code modifications
- Symbol table information is much more robust, function and variable names tend to be **consistent even across versions**

Hash Feature	Importance
raw file	0.0718
strings	0.1404
symbols	**0.7879**

raw file: information as a mix of gibberish and sometimes readable characters

ELF>??P@?I<@8 @!
@@@@@?88@8@@@P?-P?-`?-
`?m`?m ?`??-??m??mpTT@T@
P?td\?,\?I\?I????Q?tdR?td`?-
`?m`?m?? /lib64/ld-linux-x86-64.so.2
GNU)0??I????Zs??!<}y?s??D ?

strings: readable printable strings in the executable

- /lib64/ld-linux-x86-64.so.2
- GLIBC_2.2.5
- libc.so.6
- perror
- _ZNKSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEE7compareEPKc

symbols: function and variable names in the executable

- _end
- _fini
- _init
- main
- _start
- _Z11print_errnov
- _Z14print_progressiRi

Discussion
Limitations of Our Fuzzy Hash Classifier Solution

Symbol table information can be removed "binary stripping"

- Save storage space (between 10%-50% of the executable)
- Prevent reverse engineering (e.g., proprietary software)

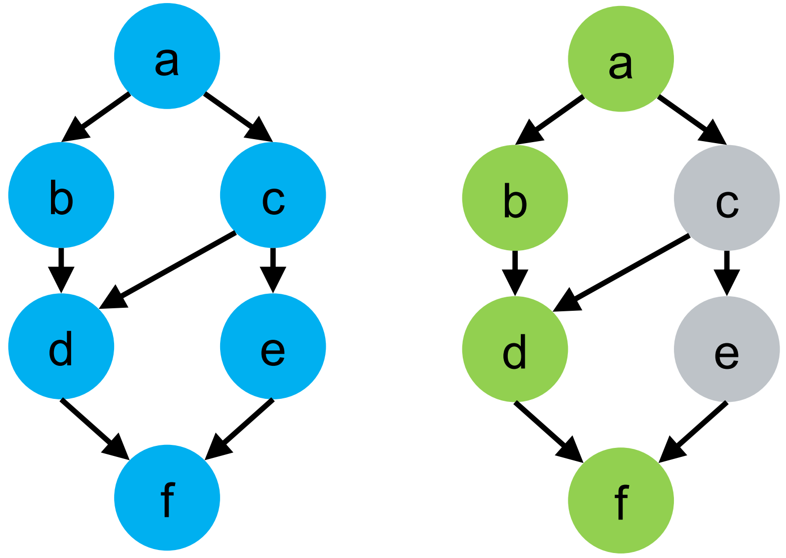
Hash Feature	Importance
raw file	0.0718
strings	0.1404
symbols	**0.7879**

Wrapper scripts and our proposed Slurm prolog approach

- Python executions will be classified as the Python interpreter itself

Reverse engineering potentially overcoming limitations

- **Control Flow Graph** (all possible execution paths)
- **Dynamic Call Tree** (actual function calls during execution)



Outlook in the Future

Next steps

Adding additional features of application executables (e.g., **ldd** shared libraries)

Deployment on Tier-0 systems and classification of user-compiled executables

Future work

Compare different machine learning approaches (e.g., SVM, KNN)

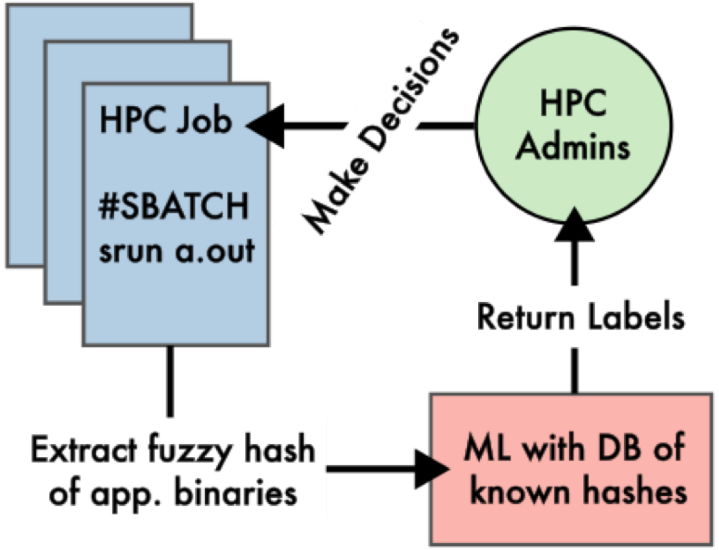
Combine static (executables) and dynamic (resource usage) classification

Test more ”invasive” analysis techniques (e.g., reverse engineering approaches)

Key Points and Take Aways

Contribution: New Fuzzy Hash Classifier for HPC applications

- Features based on fuzzy hashing,
  inspired by **malware detection techniques**
- Evaluation on pre-installed HPC applications
  ~90% micro, macro, and weighted f1-score
- Step forward toward ensuring the efficient and
  secure use of shared computational HPC resources



Take aways: Our Fuzzy Hash Classifier provides application labels for HPC jobs

- **HPC administrators** Can detect deviation from allocation purpose or terms-of-use
- **HPC researchers** Receive more reliable information and statistics about software usage

University of Basel

DMI HPC

Using Malware Detection Techniques for HPC Application Classification

Thomas Jakobsche, Florina M. Ciorba
University of Basel, Switzerland

hpc.dmi.unibas.ch | dmi-hpc@unibas.ch | dmi-hpc.bsky.social | hpc-dmi-unibas
since 08/2015

17 November 2024 - SC24 - Workshop on Cyber Security in High Performance Computing (S-HPC'24) - Atlanta, Georgia, US

Thomas Jakobsche

Florina M. Ciorba