IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661 Volume 4, Issue 1 (Sep-Oct. 2012), PP 45-48
www.iosrjournals.org

Cloud Forensics- An IS Approach

Vipul S. Chawathe[1], Prof. Dr. Bandu B. Meshram[2]
[1]VJTI, student, [2]HoD, Computer Engineering Department

Abstract : *Digital Forensics is an increasingly important and diverse R&D area charged with providing vital evidence to legal proceedings and gathering data to determine vulnerabilities exploited during system attacks. Clouds' forensics requires Intelligent Systems owing to the voluminous data content within the data store of the service provider. The functional programming that dominates data center data processing is observed.*
Keywords: *Cloud, Data Center, Digital Forensics, Functional Programming, Intelligent Systems*

I. INTRODUCTION

The Cloud hosting is where data centers really shine in electronic media. Every major IT firm has an offering of how consumers can be served by data centers wherein their cloud will be scaled as per Service Level Agreement (SLA) bounded only by the capacity of the data center. Hence, almost any cloud related product offering for cloud services providers will offer to visualize logical clubbing amongst the data centers internal resources.

When the data source is a cloud in some data center, the data center host OS is responsible for guarantying independence amongst present clouds. Also, data source resources are meant to be utilized for clouds as per SLAs and the host OS should never over-utilize resources causing availability issues to the cloud. For processing data processing of data while it's still of forensic relevance requires a light-weight reactive model. Such an Intelligent Systems (IS) agent model is explored in this paper.

The remaining material is laid out as follows. Works related with IS agents are briefly inspected in Section II. Section III takes a peek at the functional programming taking inside an agent instance for forensics case. Section IV summarizes observation. Section V gives the conclusion.

II. RELATED WORKS

The terms Autonomic Computing, Self-Managing Systems, and Self-Adaptive Systems are used interchangeably. [2] An agent can perform some activities autonomously. [1] Self-Management properties of agents include Autonomy, Social Ability, Reactivity, Proactiveness. [2] This is very different from Object Oriented Paradigm. Such differences with Object Oriented Programming have yielded multi-paradigm languages. [3]

For effective and accurate agent operation, the metamodels corresponding to the individual language have to be grasped by the programmer. The metamodels are used by the architect when design is driven by the model. Such an exhaustive approach having roots in UML, of MDD with agent as the target is FAML. [4] UML is restrictive in its centered around pre-existing real world model of class-object hierarchy. Any adaptive intelligent system software agent's architecture is better understood by domain specific way. [5]

Multiple toolkits for developing agents have been implemented. [6][7][8] With understanding of points covered by above works, it's possible to head forth for prototyping Intelligent agent system targeting the cloud forensics data.

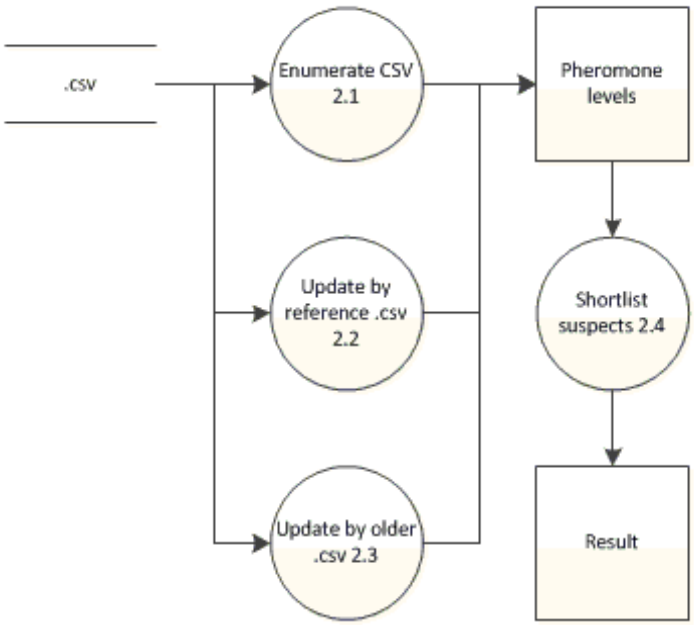
III. FUNCTIONAL PROGRAMMING IN CLOUD FORENSICS

A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers. [9]

All the VMs are tiers of multi-tier services. From every tier of data center, forensic data will get collected. For simplicity, before proceeding further to implement the agent lets agree on the data collected as sequences of tuples of user id, process, path, & process architecture(32/64-bit). Such sequences will be either from the system suspected by the forensics expert, its snapshot before being possibly corrupted or another reference tier of the cloud. Such data is easily obtained using guest view casting techniques developed using product specific documentation such as MSDN Technet combined with RemoteApp or an equivalent functionality.

As all data center operations are data centric, the beginning will be modeling the data flow functioning as DFD. The sequences mentioned above are for purposes of actual system taken to be represented using well-

defined .csv files. Instead of accounting working from level 1 overview, level 2 is directly shown to look directly inside the interesting agent component..



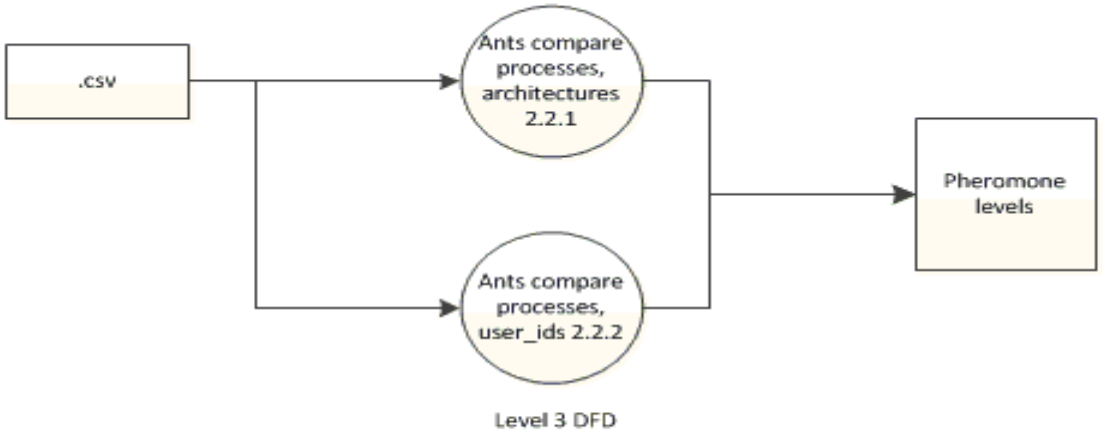
Level- 2 DFD

The pheromone level refers to suspiciousness of individual entry of initial CSV. Thus, they will be the data to be shared amongst agents every time new CSV is available. These agents have been identified as ants by some. [10]

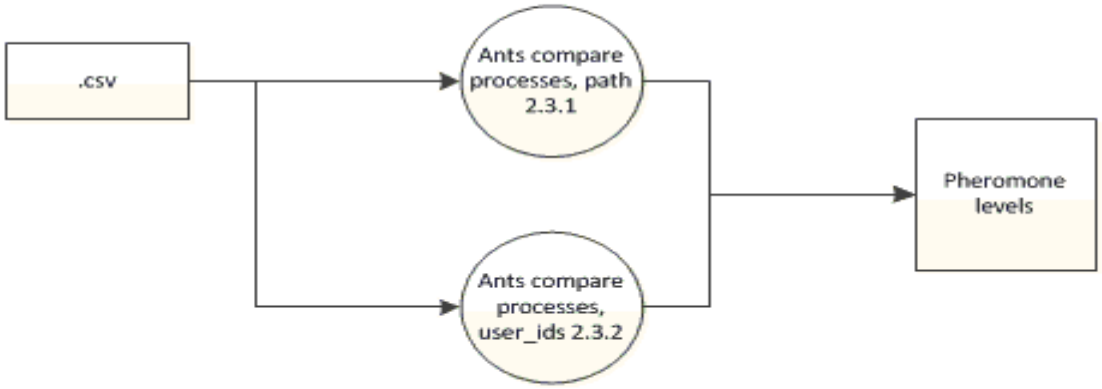
Observe that whenever .csv arrives, pheromones might be updated from either path asynchronously, depending on the agent for which .csv arrived. This is asynchronous operation. The functioning of agent on data is well defined yet generic. Also the invocation functionality is asynchronous. The data structure assumed earlier is fixed. Thus, statically typed asynchronous functional programming is required by the ant. F# is the statically typed programming language that's also based on asynchronous operation. [11] The above diagram maps to F# MVC pattern controller code like:

```
iniB.Click.Add (fun _ -> mine 'i' |>nextView(*initM*)
                         extB.IsEnabled<- true
                         intB.IsEnabled<- true)
extB.Click.Add (fun _ -> mine 'r'|> nextView
                         resulB.IsEnabled<- true)
intB.Click.Add (fun _ -> mine 'o'|> nextView
…
```

Observe that the control flow eventually is retargeted to generate new MVC view. The three different click event handlers correspond one-to-one with the three processes from level 2 DFD. Such empowering programming language implemented control flow retargeting feature another strength making F# functional programming more suited than other options at avail. [12] Now let's examine more closely how ant agents are updating the pheromones.



Level 3 DFD



Level 3 DFD

This will translate into MVC model code as:

```
    let containsElem2 anotherSeq ( proc, path, user_id, arch) = /// proces+ userid:
common domain cloud
        Seq.exists (fun (a, _, c, _) -> proc=a && user_id=c) anotherSeq
    let evalPhero cond i =( if( cond) /// Pheromone computation by Ant: "i"
                            then phero.currLs.[i]<- phero.currLs.[i]- 1 ///
increment pheromone
                            else phero.currLs.[i]<- phero.currLs.[i]+ 1 ) ///
evaporate pheromone
    match op with/// Futures
    |'i'-> fileNames.curr<- dlg.FileName
           phero.currLs<- Array.create(( CSVFileEnumerator fileNames.curr|>
Seq.length)- 1) 5
           ( CSVFileEnumerator fileNames.curr, phero.currLs)
    |'r'-> fileNames.outsider<- dlg.FileName
           let containsElem (proc, path, user_id, arch)= /// process+ Arch: separate
m/c tiers
            Seq.exists (fun (a, _, _, d) -> proc=a && arch=d) (CSVFileEnumerator
fileNames.outsider)
           let b =containsElem2( CSVFileEnumerator fileNames.outsider)
           phero.currLs |> Array.Parallel.iteri (fun i x -> /// Ant Colony of source
process number of ants
           evalPhero (CSVFileEnumerator fileNames.curr |> Seq.nth i |> containsElem) i
           evalPhero (CSVFileEnumerator fileNames.curr |> Seq.nth i |> b)
i)///currying
           ( CSVFileEnumerator fileNames.outsider, phero.currLs)
    |'o'-> fileNames.old<- dlg.FileName
```

```
            let c = containsElem2( CSVFileEnumerator fileNames.old)
            let containsElem4( proc, path, user_id, arch)= /// process+ path: separate
sample instances
             Seq.exists( fun( a, b, _, _)-> proc= a && path= b)( CSVFileEnumerator
fileNames.old)
            phero.currLs|> Array.Parallel.iteri( fun i x -> /// Another Ant Colony of
source process number of ants
            evalPhero(CSVFileEnumerator fileNames.curr|> Seq.nth i|> c) i ///currying
            evalPhero(CSVFileEnumerator fileNames.curr|> Seq.nth i|> containsElem4) i)
            (CSVFileEnumerator fileNames.old, phero.currLs)
```

Of particular interest is the match statement. In any functional language functions are evaluated as first class values. And in F# values are statically typed, but match statement will be given valid input only during runtime. This is done using the futures pattern. [13]

IV. OBSERVATIONS

Agent ant instantiation corresponding to each pheromone level maintained is done in parallel. Finally, remember the updating of pheromones was initiated as reaction to forensic expert's "Click" user action. Thus, it occurred concurrently when the user interface was responsive to the user.

Reactive, concurrent, parallel, asynchronous programming occurring together is termed as "bot" in Intelligent Systems domain and the functional language implementation is known as "agent".

Agents are coupled to their environment, designed for reactive, proactive behavior having social ability as can be seen through pheromone activity.

V. CONCLUSION

With forensics as trial, that can be replaced by guestviews relevant for predicting trends for future scaling, advertising services and so on, the concept for applying an IS is evolved. When agent is serving very specific task, it's simpler to implement using functional programming. Statically-typed data such as that from data center which hosts clouds is well-suited for functional programming by F#. Repetitive complex task of classification is submitted to agent.

Acknowledgements

This paper will be incomplete without acknowledging the thoughtful arguments and counter-arguments happening within VJTI's Computer Engineering Department's highly intellectual faculty and post graduate students as well.

REFERENCES

[1] Caroline C. Hayes, Agents in a Nutshell- A Very Brief Introduction, *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 11, NO. 1, JANUARY/FEBRUARY 1999*

[2] Huebscher, M. C. and McCann, J. A. 2008. A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv., 40, 3, Article 7 (August 2008), 28 pages DOI = 10.1145/1380584.1380585 http://doi.acm.org/10.1145/1380584.1380585

[3] Bjarne Stroustrup, Why C++ is not just an object-oriented programming language, In {OOPSLA} '95: Addendum to the proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications (Addendum) (1995), pp. 1-13, doi:http://dx.doi.org/10.1145/260094.260207

[4] Ghassan Beydoun, Graham Low, Brian Henderson-Sellers, Haralambos Mouratidis, Jorge J. Gomez-Sanz, Juan Pavo´ n, and Cesar Gonzalez-Perez, FAML: A Generic Metamodel for MAS Development, *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 35, NO. 6, NOVEMBER/DECEMBER 2009*

[5] Barbara Hayes-Roth, Karl Pfleger, Philippe Lalanda, Philippe Morignot, and Marko Balabanovic, A Domain- Specific Software Architecture for Adaptive Intelligent Systems, *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 21, NO. 4, APRIL 1995*

[6] FRANCO ZAMBONELLI, NICHOLAS R. JENNINGS, MICHAEL WOOLDRIDGE, Developing Multiagent Systems: The Gaia Methodology, *ACM Transactions on Software Engineering and Methodology, Vol. 12, No. 3, July 2003, Pages 317–370.*

[7] MICHAEL J. NORTH, NICHOLSON T. COLLIER, and JERRY R. VOS, Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit, *ACM Transactions on Modeling and Computer Simulation, Vol. 16, No. 1, January 2006, Pages 1–25.*

[8] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, Y. Diao, ABLE: A toolkit for building multiagent autonomic systems, *IBM SYSTEMS JOURNAL, VOL 41, NO 3, 2002*

[9] http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[10] Juan Luis Olmo, José Raúl Romero, Sebastián Ventura, Using Ant Programming Guided by Grammar for Building Rule-Based Classifiers, *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 41, NO. 6, DECEMBER 2011*

[11] Don Syme, Tomas Petricek, Dmitry Lomov, The F# Asynchronous Programming Model, In Proceedings of PADL 2011.

[12] Tomas Petricek, Don Syme, Joinads: a retargetable control-flow construct for reactive, parallel and concurrent programming, Joinads: a retargetable control-flow construct for reactive, parallel and concurrent programming, *Proceedings of Practical Aspects of Declarative Languages (PADL), 2011*

[13] Tomas Petricek, Alan Mycroft, Don Syme Extending monads with pattern matching. In Proceedings of the 4th ACM symposium by SIGPLAN Not. 46, 12 (September 2011), 1-12. DOI=10.1145/2096148.2034677 http://doi.acm.org/10.1145/2096148.2034677