Cloud Forensics- An IS Approach

Vipul S. Chawathe\textsuperscript{1}, Prof. Dr. Bandu B. Meshram\textsuperscript{2}
\textsuperscript{1}VJTI, student, \textsuperscript{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


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.

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:

```fsharp
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.
This will translate into MVC model code as:

```ml
let containsElem2 anotherSeq (proc, path, user_id, arch) = /// proces+ userid:
  Seq.exists (fun (a, _, c, _) -> proc=a && user_id=c) anotherSeq

let evalPhero cond i =
  if cond then phero.currLs.[i]<- phero.currLs.[i]- 1 ///
  else phero.currLs.[i]<- phero.currLs.[i]+ 1 ) ///

match op with
  |'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)
      b =containsElem( 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
```

---

**Cloud Forensics - An IS Approach**

This will translate into MVC model code as:

```ml
let containsElem2 anotherSeq (proc, path, user_id, arch) = /// proces+ user_id:
  Seq.exists (fun (a, _, c, _) -> proc=a && user_id=c) anotherSeq

let evalPhero cond i =
  if cond then phero.currLs.[i]<- phero.currLs.[i]- 1 ///
  else phero.currLs.[i]<- phero.currLs.[i]+ 1 ) ///

match op with
  |'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)
      b =containsElem( 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
```
Cloud Forensics: An IS Approach

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

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