Modelling the HUBCAP Sandbox Architecture In VDM: A Study In Security

Tomas Kulik[1], Hugo Daniel Macedo[1], Prasad Talasila[1], and Peter Gorm Larsen[1]

DIGIT, Department of Engineering, Aarhus University

Abstract. In this paper, we report on the work in progress towards the security analysis of a cloud based collaboration platform proposing a novel sandboxing concept. To overcome the intrinsic complexities of a full security analysis, we follow the formal methods approach and used separate teams: one developing a model and one implementing the platform. The goal is to create an abstract formal model of the system, focusing on the critical dataflows and security pain points, which create results to be shared among teams. The work is in progress, and, in this paper, we provide insights on a first model covering: the main features of the platform's sandboxing concept, and an analysis on the table of roles and profiles by the users, that regulates the user access to the different sandboxing capabilities. The model is specified in VDM-SL and was validated using the Overture tool combinatorial testing. The analysis has uncovered some potential issues, that have been shared with the platform implementation team, who have used it to more clearly define the table of roles and profiles within the system. Although useful, the work on implementing and securing the sandboxing is still in its initial stages, and given the novelty of the sandboxing concept, a definite answer regarding its security aspects is unclear, but a challenging research question.

Keywords: Sandbox · Collaborative Platforms · Model Based Engineering.

1 Introduction

A new collaborative platform for model based development (MBD) is emerging from the HUBCAP project [11, 12]. Differently from the traditional collaborative platforms, where users are given access to shared documents or models (with an interface to perform changes), the HUBCAP project provides multiple virtual machines (VMs) that are fully accessible via a browser web page. Users are given access to read and write to the operating system's file systems and can jointly edit the state of the sandbox components. Whether the concept will be widely adopted, or it suits only the needs of the MBD community is uncertain. What is known is that there is a need to perform a thorough analysis of the security implications of the design.

Conceptually, a HUBCAP sandbox is a group of virtual machines managed and hosted by a trusted third-party cloud provider. The users of a sandbox are MBD providers or adopters, which despite not being granted access to the source code of the platform details, receive VMs prepared with all the required dependencies to run and develop tools and models. The advantages of the concept are twofold. Firstly, the users can explore and interact with a system without having to give permission to either their local machine OS facilities or data. This goes beyond remote assistance tools that allow an external party to provide training or help configure the requesting users' local machine. Secondly, the users are able to manipulate a whole system, beyond a file or a view of the model data, which is interesting when the collaboration involves the development of cyber-physical systems and its associated need to use multiple tools and configure various system parameters. On the down side, and beyond the obvious resource consumption, the concept is new and the security requires trust between all the users and the cloud third party provider.

This paper is the first step into a full analysis of the security aspects of the HUBCAP platform. The work lays the foundation for future extensions to the Sandbox concept and evaluates whether it is possible to ensure security in multiple scenarios. For example, if the third party cloud provider is compromised, or proven to be untrustworthy.

To study the security properties of the HUBCAP platform we are developing a Vienna Development Method (VDM) model of the system components and functionality that is security critical. VDM is a formal method prescribing the development of models of computer systems, which are used to generate code for the system implementation, or to provide insight to the implementation team. The models have a precise semantics and have proven to be useful in the process of software development and security analysis.

Within this analysis we have used VDM-SL a specification language standardised by the International Standardisation Organisation [7, 15]. It is a model-oriented formal specification language which is based around logic (e.g., for invariants, pre-conditions and post-conditions) and a number of abstract data types for different kinds of collections. It has been used industrially since the seventies where it was invented at IBM's laboratories in Vienna [10, 6]. For the last two decades the Overture/VDM tool support has evolved after the VDMTools area [8, 9].

Our work provided feedback to the implementation team, and led to the refinement of the platform user roles and profiles tables, which functions as an "access-control list" to the features available to the different users. Although we are far from being able to fully assert the security properties of the platform, our work was able to elicit the dataflow and pinpoint several security checkpoints that are necessary to ensure the security of the platform. We further intend to iterate our model continuously to align with potential updates to the roles and profiles table.

The rest of the paper is organized as follows, Section 2 introduces the concepts of sandboxing and how it is utilised within the HUBCAP platform. It further introduces a table of profiles and roles governing access of clients to specific functionality within the platform. Section 3 describes the formal model of the sandboxing platform created in VDM-SL with focus on the functionality of creation of sandboxes and features present in the table of profiles and roles. Section 4 provides an overview of the formal analysis carried out to ensure that the platform is secure based on the table of profiles and roles. Section 5 then defines related work in the area of using formal methods in security and sandboxing assurance. Finally Section 6 provides concluding remarks and introduces expected future work.

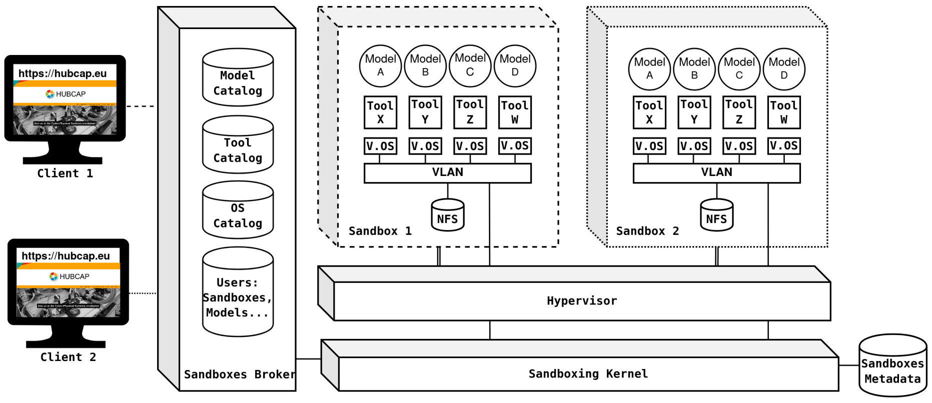

Fig. 1. The HUBCAP Sandbox architecture (taken from [11])

2 Sandboxing Platform For Collaboration

A sandbox is implemented as an isolated set of VMs (each one running a CPS tool) that interact with each other sharing a virtual dedicated subnet and a dedicated (Network File System) NFS storage service. No interaction is permitted between the VMs belonging to different sandboxes. The sandbox capability integrated with the web-platform is therefore a sort of private cloud service provider plus the middleware to manage and mediate the access to those cloud services. In addition, as many cloud service providers offer the capability to select a combination of hardware and operating systems, the HUBCAP Platform offers you to select a combination of OS environments, tools, and models to run an experiment using the HUBCAP sandbox feature.

The sandbox service is outlined in Figure 1. The web-platform is enhanced with a broker component (labelled as *Sandboxes Broker* in the figure), which hosts a web application mediating the access of different users (*Client 1* and *Client 2*) to the sandboxes they requested (*Sandbox 1* and *Sandbox 2* respectively). All the users will use an Internet browser to access the tools in the sandbox and all the interactions are mediated by the broker.

The *Sandbox Broker* has access to the catalogues of different models, tools, and preconfigured OSs that are available, so an end user can simply pick a valid combination to request a sandbox. In addition to those catalogues, the *Sandbox Broker* keeps user information, such as the user's models (private copies of the model in the catalogue, which may have been modified by the user while using the sandbox) and the sandboxes the user created. This information is important to allow the creation of new sandboxes.

The operation of user requests and the sandboxing logic is provided by the *Sandboxing Kernel*, which is a component that interacts with the system *Hypervisor* to launch the different constituents of a sandbox, namely:

- *NFS* - Network File System providing storage in the form of shared folders where model files and tool outputs are placed.
- *VLANs* - Virtual networks restricting the communications of the VMs inside a sandbox to the set of VMs composing it and those only.

– *VOS* - Virtual machines running the OSs supporting a tool, a remote desktop protocol to provide the clients access to the tool display, and other monitor and interoperability tools to operate the VM inside the Kernel.
– *Tools* - The tools running a model or a multi-model.
– *Models* - A mathematical/formal description of a component.

The operation relies on a database of metadata about the different sandboxes (the *Sandboxes Metadata* component in the figure). This component stores and keeps track of the sandboxes' states (running, suspended, ...) and user ownership of the resources. It is worth highlighting that the Kernel has direct network connections to the Sandboxes' VLANs. We base our model on the building blocks of the sandboxing platform presented in this section, focusing on the components necessary for analysis of the client access based on roles and profiles, while abstracting away details such as VLAN addresses, specifics of the operating systems, file storage and tool properties.

2.1 Designated Roles and Activities

The sandbox platform as presented within this paper offers several features that are accessible by use of clients. Each feature is protected by requiring the client being of a specific role and profile. The full overview of features and their associated roles and profiles is shown in Table 1. The profiles within the system are provider and consumer, where the provider can add new content to the sandboxing platform such as operating systems or tools, while the consumer could utilise the content already present within the platform. Furthermore the system defines two roles on the sandboxes themselves, where the owner role is assigned to a client creating the sandbox, while a guest role is assigned to clients that have been invited by the owner to the sandbox. Without the invitation the guest cannot access the sandbox.

Table 1. Overview of profiles and roles

Feature	**Provider Owner**	**Provider Guest**	**Consumer Owner**	**Consumer Guest**
Access to remote viewer	X	X	X	X
Upload archive	X	X	X	X
Download archive	X		X	
Invite guests	X		X	
Destroy sandbox	X		X	
Select tool	X		X	
Select model	X		X	
Select operating system	X			
Save tool	X			
Upload new model	X			
Delete repository tool	X			

3 Formal Model

The model consist of several sandbox building blocks, namely (1) Client, (2) Broker, (3) Gateway, (4) Sandbox, (5) Server and (6) System. Each of these components, excluding the gateway is represented as a VDM-SL file with varying level of detail. The model considers a single default module tracking the state of all of the components. The primary purpose of the model is to determine if the system functionality behaves according to the provided table of roles and profiles. This section provides detailed overview of the different model components.

3.1 Client

The client is the entry point into the sandbox. It is a component used by the user to access and interact with the system. The system can have multiple clients, where each client has an explicit identity, expressed as `ClientId = nat`. Furthermore the clients that can access the system must be considered valid, this is handled by ensuring that the client identity is present in a database of valid identities stored within the system. We express this check as a pre condition on all of the client calls as `pre cId in set validClients;`. The client can call several operations against the broker, namely starting a new sandbox, destroying a sandbox, selecting a model from a repository, selecting a tool from a repository, selecting an operating system from a repository, inviting a guest to a sandbox, removing own repository item, connecting to a sandbox and disconnecting from a sandbox. In order to create new sandbox the client needs to first select the tool, the operating system and optionally a model that shall be present within this sandbox. The selections of the client are recorded within a `ClientSt` record and are presented together with launching of new sandbox in Listing 1.1. Once the sandbox is launched the client can access the sandbox as presented in Listing 1.2. The actual deployment of the sandbox and establishment of connections is handled by different components, namely, the broker and the system.

```
SelectOS: ClientId * OSId ==> ()
SelectOS(cId, osId)==
  clientst.selectedOS := SelectOperatingSystem(osId, cId)
pre cId in set validClients;

SelectToolFromRepository: ClientId * ToolId ==> ()
SelectToolFromRepository(cId, tId) ==
  clientst.selectedTool := SelectTool(tId, cId)
pre cId in set validClients;

SelectModelFromRepository: ClientId * ModelId ==> ()
SelectModelFromRepository(cId, mId) ==
  clientst.selectedModel := SelectModel(mId, cId)
pre cId in set validClients;

LaunchNewSandbox: ClientId ==> ()
LaunchNewSandbox(cId) ==
  StartNewSandbox(cId, clientst.selectedTool,
    clientst.selectedModel, clientst.selectedOS)
pre cId in set validClients;
```

Listing 1.1. Sandbox elements selection and sandbox launch

```
-- Connect to a sandbox
AccessSandbox: ClientId * SandboxId ==> ()
AccessSandbox(cId, sId) ==
  if not BrokerInitiateSandboxAccess(cId, sId)
  then GetError(cId, mk_token(1))
pre cId in set validClients;
```

Listing 1.2. Client accessing a sandbox

3.2 Broker

The broker is a component of the overall system that the client interacts with, before being handed over a direct connection to a sandbox. This component is aware of currently active sandboxes, their respective ownership and records the items available within the repository, specifically the operating systems, tools and models. The broker also records ownership of these repository items. This is represented by mappings as for example ownership of a sandbox recorded in `Owners = map ClientId to set of SandboxId`, while the ownership of tools is recorded as `ToolOwners = map ClientId to set of ToolId`. Furthermore the broker records how specific tools, models and operating systems are utilised within different sandboxes, an example of what is a mapping `SandboxTools = map SandboxId to set of ToolId`. A client selecting for example an operating system calls a broker operation as shown in Listing 1.3, where the precondition states that the client either needs to have a profile of a provider or a consumer as presented in Table 1.

```
SelectOperatingSystem: OSId * ClientId ==> [nat]
SelectOperatingSystem(osId, cId) ==
  return if osId in set brokerst.validOSs
        then osId
        else nil
pre cId in set brokerst.providers or
    cId in set brokerst.consumers;
```

Listing 1.3. Broker providing an operating system

The broker is also responsible for starting a new sandbox. This operation registers the sandbox within the system and it becomes available for connections. This operation further registers the requested operating system, tool and optionally a model within the sandbox and is presented in Listing 1.4. The precondition checks that the user (via the client) has made valid selections, while the post-condition ensures that the sandbox is registered correctly with a newly generated sandbox identity.

```
StartNewSandbox : ClientId * ToolId * [ModelId] * OSId ==> ()
StartNewSandbox(cId, tId, mId, oId) ==
let sId = GenerateNewSandboxId()
in
  (brokerst.sandboxTools := brokerst.sandboxTools munion
                            {sId |-> {tId}};
   brokerst.sandboxOSs := brokerst.sandboxOSs munion
                          {sId |-> {oId}};
   if mId <> nil
   then brokerst.sandboxModels := brokerst.sandboxModels munion
                                  {sId |-> {mId}};
   systemSandboxes := systemSandboxes munion
                      {sId |-> mk_Sandbox(sId, {1})};
   if cId in set dom brokerst.owners
   then brokerst.owners(cId) := brokerst.owners(cId) union {sId}
   else brokerst.owners := brokerst.owners munion
                           {cId |-> {sId}})
pre tId in set brokerst.validTools
    and oId in set brokerst.validOSs
    and mId <> nil => mId in set brokerst.validModels
post card dom systemSandboxes = card dom systemSandboxes + 1
    and cId in set dom brokerst.owners
    and Max(systemSandboxes) in set brokerst.owners(cId);
```

Listing 1.4. Broker starting a new sandbox within the system

In order to provide connectivity to the sandbox the broker checks if the client is associated with the requested sandbox. This means that the client either needs to be an owner of the sandbox or a guest invited for access to this sandbox. This operation is shown in Listing 1.5. The precondition checks that the client actually has some role within the system.

```
BrokerInitiateSandboxAccess: ClientId * SandboxId ==> bool
BrokerInitiateSandboxAccess(cId, sId) ==
  let sandboxes = GetSystemSandboxes(),
      servers = dunion {s.sandboxServers
                       | s in set rng sandboxes
                       & s.sandboxId = sId}
  in
    (for all x in set servers do
       UpdateConnections(cId, x, sId, true);
     return true)
pre not ClientIsNull(cId,brokerst.providers,brokerst.consumers,
                     brokerst.owners, brokerst.guests) and
    ((cId in set dom brokerst.owners and
     sId in set brokerst.owners(cId)) or
    (cId in set dom brokerst.guests and
    sId in set brokerst.guests(cId)));
```

Listing 1.5. Broker initiating access to a sandbox for a client

The broker further handles operations such as assigning a guest to a sandbox invited by a sandbox owner updating the repository of items, i.e. by uploading new items or removing them, based on the wishes of the client and destroying the sandbox, thus removing it from the system.

3.3 Gateway

The gateway is a connection component providing connectivity between the sandbox and the client. More specifically the gateway opens a connection to every server that exists under a sandbox and registers which client holds the open connection to this sandbox. In the current model the focus is on analysing the specific roles of the clients against the actions they can take and hence the gateway connections are simply recorded under the system state as `GatewayConnections = map ClientId to set of ServerId` and also `GatewayConnectionsSandbox = map ClientId to set of SandboxId`. The client can connect to and disconnect from a sandbox- a functionality captured by an operation shown in Listing 1.6. The listing further shows the broker keeping a set of active sandboxes (i.e. sandboxes with an open connection).

```
UpdateConnections: ClientId * ServerId * SandboxId * bool ==> ()
UpdateConnections(cId, sId, sbId, connect)==
  if connect
  then (if cId in set dom gatewayConnections
        then atomic
        (gatewayConnections(cId):= gatewayConnections(cId)
                                   union {sId};
         gatewayConnectionsSandbox(cId):=
              gatewayConnectionsSandbox(cId) union {sbId};
         brokerst.activeSandboxes:= brokerst.activeSandboxes
                                    union {sbId}))
  elseif cId in set dom gatewayConnections
  then atomic
  (gatewayConnections(cId):= gatewayConnections(cId) \ {sId};
   gatewayConnectionsSandbox(cId):=
        gatewayConnectionsSandbox(cId) \ {sbId};
   brokerst.activeSandboxes:= brokerst.activeSandboxes \ {sbId})
```

Listing 1.6. Gateway recording the connections to a sandbox

3.4 Sandbox

Sandbox is a secure container encapsulating servers, tools and a model that the user decided to work with. The nature of the sandbox is to provide a demo environment to users that want to try use of model based engineering within a hosted environment. The sandbox provides an isolation from other sandboxes and the wider system and users working with sandboxes follow functionality roles as shown in a Table 1. The user can, via clients, create new sandboxes, connect to sandboxes, disconnect from sandboxes and destroy sandboxes. If a user creates a sandbox, this user can further invite another user as a guest to the sandbox, via the identity of the client that the guest uses. Each sandbox within the system could be understood as a collection of servers with specific tools installed and each sandbox carries an unique identity. The definition of the sandbox is shown in Listing 1.7.

```
types
SandboxId = nat;
SandboxServers = set of ServerId;
Sandbox::
 sandboxId : SandboxId
 sandboxServers : SandboxServers
```

Listing 1.7. Sandbox specification

In the current iteration the model is only capable of creating sandboxes consisting of a single server, as this is sufficient for analysis of the access and functionality roles.

3.5 Server

Within the modelled system, the server is a machine running an operating system with a specific tool allowing a user to manipulate models with a possibility of a remote access. Each server has to be a part of a sandbox in order to be accessible and each server carries a unique identity as `ServerId = nat`. Within the model presented in this paper, the only considered functionality of the server is, the server being a part of a sandbox.

3.6 System

The System VDM file captures functionality of the system as a whole and facilitates operations not suitable for other components. The system for example generates new identities for new sandboxes as shown in Listing 1.8. Furthermore the system contains the state for the entire system and its components as shown in Listing 1.9. The system represents an abstract container for the system components with its own high level functionality.

```
pure Max: SystemSandboxes ==> nat
Max(ss) ==
  let max in set dom ss be st forall d in set dom ss & d <= max
  in
    return max
pre ss <> {|->};
GenerateNewSandboxId: () ==> nat
```

```
GenerateNewSandboxId() ==
  return Max(systemSandboxes) + 1;
```

Listing 1.8. Operations generating identity for new sandbox

```
state SystemSt of
  gatewayConnections : GatewayConnections
  gatewayConnectionsSandbox : GatewayConnectionsSandbox
  systemSandboxes : SystemSandboxes
  toolOwners : ToolOwners
  modelOwners : ModelOwners
  brokerst : BrokerSt
  clientst: ClientSt
  validClients : ValidClients
inv ss == dom ss.gatewayConnections =
          dom ss.gatewayConnectionsSandbox
init s == s = mk_SystemSt({|->},{|->},{|->},{|->},{|->},
     mk_BrokerSt
         ({},{},{},{},{},{},{|->},{|->},[],{|->},{|->},{|->}),
     mk_ClientSt(nil,nil,nil),{})
end
```

Listing 1.9. System state

4 Formal Analysis

This section specifies the traces that have been used to analyse several system features against the table of profiles and roles. The most important functionality analysed was the creation of new sandboxes and ensuring that uninvited clients do not have access to the sandboxes, an important aspect for the integrity of the platform.

4.1 Traces under analysis

To analyse the specific cyber security properties, we express several scenarios as traces and use the combinatorial testing feature of overture. The scenarios considered within this paper are creation of sandboxes, connection and disconnection to sandboxes and creation of sandboxes considering invitation of guests to the newly created sandboxes. These scenarios have been selected based on the communication with the implementation team in order to ensure that the table of roles and profiles provides the intended permissions in order to access the system. The analysis has uncovered a potential permissions issue within the system. In order to create a sandbox a trace as shown in Listing 1.10 could consider several scenarios, where the user with a profile consumer, a user with a profile provider (profiles as specified in Table 1) and a user without a specific profile attempt to create a new sandbox. To do this, the users select the operating system, the tool and the model that shall be used within the sandbox (only the operating system is mandatory as the tool and the model could be uploaded later to an existing sandbox). Once the selection is complete the users simply launch a new sandbox. The initial operation calls in the listing simply sets up the environment, ensuring that the system has the clients, the operating systems, the tools and the models registered.

```
CreateSandboxMultipleClients:
SetupClients(1);
SetupClients(2);
SetupClients(3);
SetupProviders(1);
SetupConsumers(2);
SetupOSs(1);
SetupTools(1);
SetupModels(1);
let clientId in set {1, 2, 3}
in
(SelectOS(clientId, 1);
 SelectToolFromRepository(clientId, 1);
 SelectModelFromRepository(clientId, 1);
 LaunchNewSandbox(clientId));
```

Listing 1.10. A trace of different clients launching a sandbox

In order to initialize client within the system a `SetupClients` operation is called with further operations setting up the client as a consumer or a provider. This is shown in Listing 1.11.

```
SetupClients: ClientId ==>()
SetupClients(cId) == validClients:= validClients union {cId}

SetupProviders: ClientId ==> ()
SetupProviders(cId) ==
  brokerst.providers := brokerst.providers union {cId};
```

Listing 1.11. Setting up a client within the system

This scenario has uncovered an issue where a user without a specific role and without a specific profile (but still a valid user within the system) attempts to launch a sandbox. While this user is allowed to launch a sandbox, it is only the users with profiles *consumer* or a *provider* that are allowed to select an operating system. This is captured as a precondition on the `SelectOS`. Since the user without an explicit profile can not select an operating system, it is not possible for this user to create a new sandbox. This led to a suggestion that users that are created on the platform but do not have assigned access rights should not be able to create a new sandbox.

Another trace considered was the trace specifying a scenario where a user (using a client) creates a sandbox and another user (using a separate client) attempts to connect to this sandbox without first receiving an invitation to this sandbox. This trace is shown in Listing 1.12. The analysis of this trace has uncovered that the permissions model based on roles and profiles specified in Table 1, expressed as a pre-condition for the operation `AccessSandbox` prevents this from happening, ensuring that only users with legitimate claim could access the sandbox, i.e. only the owners or invited guests can access the sandbox.

```
CreateSandboxAndUninvitedConnect:
SetupClients(1);
SetupClients(2);
SetupProviders(1);
SetupOSs(1);
SetupTools(1);
SetupModels(1);
let clientId in set {1}
in
(SelectOS(clientId, 1);
 SelectToolFromRepository(clientId, 1);
 SelectModelFromRepository(clientId, 1);
 LaunchNewSandbox(clientId);
 AccessSandbox(2, 1));
```

Listing 1.12. Uninvited client attempting connection to a sandbox

Several other traces have been analysed, covering the launching of the sandbox, where these traces have confirmed that the permission table does allow access as specified. One of these flows is shown in Listing 1.13, representing a simple scenario of a user with a provider profile launching a new sandbox.

```
CreateSandboxNoModelNoTool:
SetupClients(1);
SetupProviders(1);
SetupOSs(1);
let clientId in set {1}
in
(SelectOS(clientId, 1);
 SelectToolFromRepository(clientId, 1);
 SelectModelFromRepository(clientId, 1);
 LaunchNewSandbox(clientId)
);
```

Listing 1.13. Legitimate user creating a sandbox

4.2 Results and suggestions

The security analysis of the sandbox platform has uncovered potential improvements to the table of roles and profiles. The first suggestion is to explicitly state what roles and profiles are needed in order for the client to be able to create a new sandbox. Second suggestion is to carry out a consistency check on the roles and profiles after each update of the roles and profiles table, a task well suited for VDM as during the modelling work one such update has been provided by the implementation team with minimal amount of effort to update the model. This first update has been a result of inconsistencies uncovered during encoding of the table into the VDM model.

In total 11 traces have been created, most of them considering only a single value for an input parameter, while a single trace has considered a selection of three values for the input parameter. This lead to creation of 13 tests covering the actions of interest from the roles and profiles table as well as considering an important action of creating a new sandbox. The evaluation of traces has taken less than two seconds, providing a well performing solution for analysis of client permissions. Since the roles and profiles table is continuously updated it is expected that the VDM model pre- and post-conditions on different operation calls will be updated as well in order to carry out an analysis ensuring the security of the sandbox platform.

5 Related Work

Given its popularity, there are several works covering formal verification of cloud systems [5]. In fact, it is not unusual for major cloud vendors to adopt, use, and have internal departments doing research in formal methods and verification tools. Our work is different from the usual in the sense that cloud operators typically do not provide access to the same sandboxed OSs to multiple users at the same time. The work in [16] is an example of the application of formal methods to secure the assets of organisation that wish to use federated cloud systems (systems where several providers and local clouds are combined), but are afraid of compromising the security of their solution. The paper shows how the security of information flow can be analysed. Our work focus is also at the dataflow level, although information security is not our current focus. Our goal is to achieve a solution similar to the one presented in [14] and [3], but generalised beyond network checks and the cloud machinery provisioning respectively. We foresee the development of a security middleware applying checks at the pain points elicited during the platform modelling and auditing tasks. At project proposal phase we envisaged the possibility to specialise and apply malware detection techniques as the one in [13], but the sandbox prototype based on sandboxed OS images is much more suited to generalist and COTS malware checkers.

Sandboxing is a widely used to establish security. Whether we think of the popular browser tabs, an operating system kernel, or a C program memory sandboxes, their security involves a sandbox. Formal methods have been widely used to verify the good application of sandboxing. In [4], the authors propose to extract a browser sandboxing kernel with leak-proof security. Our work does not leverage on theorem proving, and it does not intend to substitute the current implementation with a correct-by-construction one, but that is still a possibility. Our current focus is the usage of lightweight specification in the form of pre/post-conditions and invariants to reason about the system security. In [1] the authors extract an executable COQ model of an hypervisor, a "real-world" C++ implementation of a virtualisation architecture. The executable model is then used to as an oracle for the expected behaviour. Our work threads along the same lines, we expect to model our sandboxing platform in VDM, from which we expect to derive an oracle and expected traces exhibiting secure and insecure dataflows. In [2], a similar approach is used to program data memory. In all the approaches implementations are expected to be secure, after modelling and using tools in the security audit.

6 Conclusion

This paper reported on findings of use of VDM-SL in security analysis of online (potentially cloud based) collaboration platforms. The system architecture presented in this paper is based on a real life sandboxing platform, a part of a project aimed at collaborative use of model based engineering while preserving the information security and intellectual property. We created a model of the system, considering features provided within a table of roles and profiles by the implementation team. To model this feature set we have considered the mentioned actions and modelled the roles and profiles as pre conditions. We have then analysed several specific scenarios within the model by use of the combinatorial testing. The analysis has uncovered a potential issue, presenting an inconsistency within the intended sandbox creation functionality, specifically an issue within a hierarchy of operation calls, where an action can be requested without specifically defined roles, however it is dependent on another action requiring specific role. These results have been shared with the implementation team, that has used it to more clearly define the table of roles and profiles within the system. The analysis has in total considered 13 tests and could be carried out in seconds.

As a future work we intend to align with the progress of the implementation team and update the model to analyse every update to the table of roles and profiles. Furthermore we plan to extend our model to be able to analyse aspects of federated cloud, where some sandbox servers could exist within an on premises data-center while other servers would exist within a public cloud service. To this end we intend on proposing an access model, i.e. and updated list of roles and profiles, that has been modelled using VDM and share the findings with the implementation team.

Acknowledgements. The work presented here is partially supported by the HUBCAP Innovation Action funded by the European Commission's Horizon 2020 Programme under Grant Agreement 872698.

References

1. Becker, H., Crespo, J.M., Galowicz, J., Hensel, U., Hirai, Y., Kunz, C., Nakata, K., Sacchini, J.L., Tews, H., Tuerk, T.: Combining mechanized proofs and model-based testing in the formal analysis of a hypervisor. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) FM 2016: Formal Methods. pp. 69–84. Springer International Publishing, Cham (2016)
2. Besson, F., Blazy, S., Dang, A., Jensen, T., Wilke, P.: Compiling sandboxes: Formally verified software fault isolation. In: European Symposium on Programming. pp. 499–524. Springer, Cham (2019)

3. Bleikertz, S., Vogel, C., Groß, T., Mödersheim, S.: Proactive security analysis of changes in virtualized infrastructures. In: Proceedings of the 31st Annual Computer Security Applications Conference. p. 51–60. ACSAC 2015, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2818000.2818034, https://doi.org/10.1145/2818000.2818034
4. Jang, D., Tatlock, Z., Lerner, S.: Establishing browser security guarantees through formal shim verification. In: Presented as part of the 21st USENIX Security Symposium (USENIX Security 12). pp. 113–128. USENIX, Bellevue, WA (2012), https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/jang
5. Kulik, T., Dongol, B., Larsen, P.G., Macedo, H.D., Schneider, S., Tran-Jørgensen, P.W.V., Woodcock, J.: Formal methods in security survey (in preparation) (2020)
6. Kurita, T., Nakatsugawa, Y.: The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip. Intl. Journal of Software and Informatics **3**(2-3), 343–355 (October 2009)
7. Larsen, P.G., Hansen, B.S., Brunn, H., Plat, N., Toetenel, H., Andrews, D.J., Dawes, J., Parkin, G., et al.: Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language (December 1996)
8. Larsen, P.G.: Ten Years of Historical Development: "Bootstrapping" VDMTools. Journal of Universal Computer Science **7**(8), 692–709 (2001)
9. Larsen, P.G., Battle, N., Ferreira, M., Fitzgerald, J., Lausdahl, K., Verhoef, M.: The Overture Initiative – Integrating Tools for VDM. SIGSOFT Softw. Eng. Notes **35**(1), 1–6 (January 2010). https://doi.org/10.1145/1668862.1668864, http://doi.acm.org/10.1145/1668862.1668864
10. Larsen, P.G., Fitzgerald, J., Brookes, T.: Applying Formal Specification in Industry. IEEE Software **13**(3), 48–56 (May 1996)
11. Larsen, P.G., Macedo, H.D., Fitzgerald, J., Pfeifer, H., Benedikt, M., Tonetta, S., Marguglio, A., Gusmeroli, S., Jr., G.S.: An Online MBSE Collaboration Platform. pp. 263–270. INSTICC, Proceedings of the 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH (July 2020). https://doi.org/10.5220/0009892802630270
12. Larsen, P.G., Soulioti, G., Macedo, H.D., Alifragkis, V., Fitzgerald, J., Livanos, N., Pfeifer, H., Pasquinelli, M., Benedict, M., Thule, C., Tonetta, S., Stritzelberger, B., Marguglio, A., Sutton, L.F., Obstbaum, M., Gusmeroli, S., Beutenmüller, F., Jr., G.S., Wijnands, Q., Talasila, P.: Enabling combining models and tools in an online mbse collaboration platform. In: Model Based Space Systems and Software Engineering (MBSE2020). ESA-ESTEC, Noordwijk, The Netherlands (September 2020)
13. Macedo, H.D., Touili, T.: Mining malware specifications through static reachability analysis. In: European Symposium on Research in Computer Security. pp. 517–535. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
14. Madi, T., Jarraya, Y., Alimohammadifar, A., Majumdar, S., Wang, Y., Pourzandi, M., Wang, L., Debbabi, M.: Isotop: Auditing virtual networks isolation across cloud layers in openstack. ACM Trans. Priv. Secur. **22**(1) (Oct 2018). https://doi.org/10.1145/3267339, https://doi.org/10.1145/3267339
15. Plat, N., Larsen, P.G.: An Overview of the ISO/VDM-SL Standard. Sigplan Notices **27**(8), 76–82 (August 1992)
16. Zeng, W., Koutny, M., Watson, P., Germanos, V.: Formal verification of secure information flow in cloud computing. Journal of Information Security and Applications **27**, 103–116 (2016)