

Logging System Architectures for Infrastructure as a Service Cloud

Winai Wongthai^{1,2}, Aad van Moorsel^{1,2}

¹*Department of Computer Science and Information Technology and Research Center for Academic Excellence in Nonlinear Analysis and Optimization, Naresuan University, Thailand.*

²*School of Computing Science, Newcastle University, UK.*

winaiw@nu.ac.th

Abstract—This paper proposes 93 possible architectures of a logging system in the cloud. The architectures can enable the systematic design, implementation and deployment of logging systems in the cloud. These architectures can be used to guide the future design and development of such logging systems. This should decrease the effort and time commitments required for the deployment of logging systems. This also can tenable logging systems to work in real world cloud production systems. To the best of our knowledge, these 93 architectures are not yet described in the literature.

Index Terms—Cloud Monitoring; Accountability; Logging System Architecture.

I. INTRODUCTION

The cloud can possibly be a key of IT businesses to enable software to be more attractive as a service and shape the design and investment approach of IT hardware [21]. This paper focuses on the infrastructure as a service or IaaS cloud such as Amazon Elastic Compute Cloud (Amazon EC2). IaaS consists of a cloud-based infrastructure offering customers raw computational resources, such as storage and networking, with a pay-per-use billing model [7]. This type of cloud is used by enterprises, government departments, and in academia [7, 8]. However, [1] argue that in 2015 cloud security is one of the five most significant cybersecurity market trends which will define the investment of organizations' cybersecurity budgets. A report called Top Threats to Cloud Computing [2] published by Cloud Security Alliance (CSA) expounds these concerns.

Accountability refers to approaches to mitigate the risks associated with the threats. Accountability in the cloud requires cloud behaviors that can be inspected by any party, as argued by many researchers [3, 4, 5, 6]. Wongthai et al [7] argue that a logging system is a major mechanism in accountability solutions to support mitigation of the top threats. They also define a logging system as being composed of logging processes and log files.

A logging process focuses on logging-related tasks, and log files store contents produced by these processes [7]. However, previous logging systems solutions with accountability components [7, 8, 9, 10, 11, 12, 13, 14, 15] have been provided without descriptions of all possible architectures of a logging system in IaaS. The form of these architectures is based on the critical components of the logging process and log files. This paper proposes all these possible architectures. The architectures can guide the systematic design, implementation, and deployment of logging systems in the cloud. That this system paradigm

should be considered when developing logging systems is also supported in [7, 8].

Knowing these architectures should decrease both the effort and time commitments required for the deployment of logging systems. Reducing these to the minimum is essential, especially in the ever changing, dynamically growing, and continuously evolving behavior of Internet-based services [22]. The architectures should also assist in defining and identifying logging system patterns. Stating these architectures as 'patterns' produces a number of benefits similar to the benefits of design patterns in object-oriented software design and development area (defined by Gamma et al. [20]).

The two main contributions of this paper: firstly, in order to describe all possible system architectures of a logging system in IaaS, we enhanced our previous work which we called a generic logging template for Infrastructure as a Service (IaaS) cloud [7], Section 2. Secondly, in Section 3, we introduced and discussed 93 possible architectures of a logging system, which we have divided into three categories. We then discussed the structure and gave examples of architectures in each category. This includes analysis of some existing work in relation to some architecture, in the end of Section III

II. ENHANCING A GENERIC LOGGING TEMPLATE FOR IAAS

A. IaaS Architecture

The architecture is defined in [7] and based on the Xen system. There are two sides in this architecture. The provider side can be an organization that offers virtual machines (VMs) to the customer side. The customer side can rent the VMs and remotely access them via the Internet. Hw is a machine that works as a host of a hypervisor and all guest operating systems (OSes). A hypervisor is a software that enables the machine to run more than one guest OS. Dom0 is a privileged domain guest OS that is launched by the hypervisor during system boot. It directly accesses the hw and manages domUs. A domU is an unprivileged domain guest OS that runs on top of the hypervisor. It is a VM and can be considered as an example of an IaaS cloud product that customers can purchase from the providers.

B. A Generic Logging Template for IaaS Cloud

Our previous work [7] provided the template as illustrated in Figure 2. It facilitates the understanding of the layout of logging system components in IaaS. The main aim of the template is to enable systematic analysis of logging systems

in terms of the systems' security themselves before deploying them in the production systems.

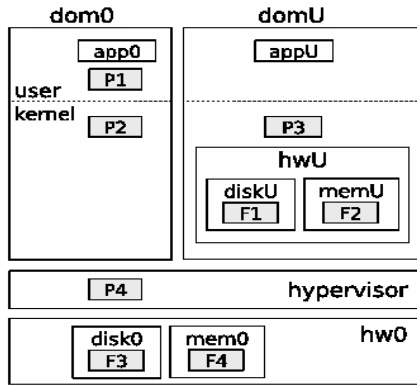


Figure 1: The overall view of a generic logging template from [7], its IaaS components (white boxes) and logging components (shaded colour, logging process: P1-4, and log files: F1-4).

In terms of logging systems in IaaS, this template clarifies the layout of critical components such as a provider's hosting system; dom0, or a customer's virtual machine; domU, and the logging processes, and log files.

In the template (Figure 1), IaaS components include hw0, hypervisor, dom0, domU, hwU, app0, appU, disk0, diskU, mem0, and memU. The first four components were already discussed in Section 2.1. Note that hw0 is the same as hw in Section 2.1, 0 indicates that it is physically managed and owned by a provider. AppU and app0 are applications that run inside domU and dom0 respectively, U indicates that it is virtually managed and owned by a customer. Disk0 is a physical disk of the hw0, and diskU is a virtual disk of a domU. Mem0 is the main memory of the hw0, and memU is the virtual main memory of domU. P1-P4 are the logging processes that perform logging-related tasks. F1-F4 are log files that are used for storing contents produced by the P1-P4.

C. The Enhancement of the Template

In order to describe all possible system architectures of a logging system in IaaS, we provided an enhancement to the template. The enhancement is for the discussions of logging system architectures. It describes the generic logging components for IaaS cloud, shown in Figure 2. Details of all these components are discussed in Section 2.1 and 2.2 except for components P3, P4 and P5. In [7], we first described P3, P4 but they are redefined here, whereas P5 is newly defined in this paper.

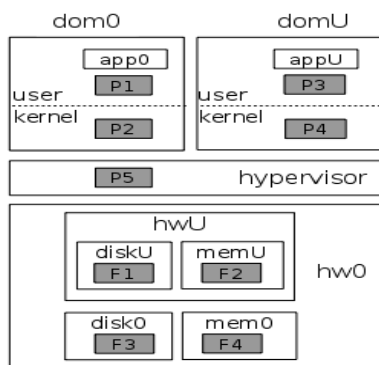


Figure 2: The overall view of generic logging components: logging process or Px (P1 to P5), and log files or Fy (F1 to F4).

HwU is moved to be inside hw0. P3 is a domU user level process such as a logging mechanism (in a logging system of [15]) to record the actual usage of data files that reside in a disk of a customer VM. We call this disk diskU. P3 could also be special domU user level libraries for logging purposes. P4 is a domU kernel level process to, for example, intercept domU file and network operations, then temporarily store such intercepted data into diskU. Flogger, in HP Floggers System is an example [16]. P5 is a process inside a hypervisor to, for example, record information about incoming and outgoing network packets of domU. A logger in [17] is an example of this process.

From Figure 2, all of the generic logging components can be divided into three sets. The first one is the IaaS set of components (the white boxes in the figure). This set of components includes: hypervisor, dom0, domU, hw0, hwU, app0, appU, disk0, diskU, mem0, and memU. The next set is the logging process set and the log file set of components (shaded boxes). The logging process set is composed of the logging processes (Px, x=1,...,5). The log file set is composed of Fy (y=1,...,4). Full details of all components of this set except P3, P4, and P5 are in Section II A and B. These generic logging components will be used through this paper.

III. ALL POSSIBLE ARCHITECTURES OF A LOGGING SYSTEM

There are 93 possible architectures, which are formed based on the critical components Px and Fy in the generic logging components in Figure 2. These components are in three domains: dom0, domU, and hypervisor. For Fy, it is assumed that if a logging system architecture deploys F1 in diskU or F3 in disk0, it then needs to eventually deploy F2 in memU or F4 in mem0 respectively. Any actual logging system will be based on one of the 93 possible architectures discussed in Section 3.1 to 3.3. Based on our review of the literature, a few of the possible architectures exist. These few are just a sub-set of possible architectures.

We have extended the sub-set of architectures already identified in the literature. These as yet undefined architectures are certainly of interest. For the purposes of this discussion, we call these as yet undefined architectures 'non-existing architectures'. For example, an architecture that deploys all nine Px and Fy, is the most complicated architecture not yet defined, and will be discussed in Section 3.3. One of the advantages of this architecture is it provides many abilities to facilitate logging tasks including: recording the necessary logging data as log files across domU and dom0. Reducing the size of the trusted computing base (TCB) of a logging system is an important concern for logging system development in IaaS, as argued by [7] and [8]. TCB is a term in computer security to refer to the set of all hardware, software, and procedural components, which enforce the security policy [18]. One disadvantage of having a complicated logging system architecture is having a big TCB size. This is because Px and Fy are deployed and distributed across all the three domains (domU, dom0, and hypervisor).

To simplify the presentation of all the possible architectures, they are divided into three categories: single domain, two domains, and three domains. A single domain category means that all Px of a logging system are deployed in either dom0, domU, or a hypervisor. The two domains category means that all Px of a logging system are deployed

in two domains among dom0, domU, or a hypervisor, and the three domains category means that Px are deployed in all three domains, thus at least one Px of a logging system is deployed in each domain. The 93 possible architectures are composed of: 21 architectures in the single domain category, 45 architectures in the two domains category, and 27 architectures in the three domains category. The following subsections will clarify the architecture composition of each category.

Conditions of all categories are: 1) each logging system architecture from these three categories has to deploy Px based on its category's conditions which will be discussed in Subsection 3.1-3.3. Eventually, the system has to deploy Fy by choosing one of these three different approaches of deploying Fy in disk0, in diskU, or in both disk0 and diskU, which we call disk0U, 2) when a system is deployed in dom0, its Px can be in dom0 user level as P1 or dom0 kernel level as P2, or both of them, 3) when a system is deployed in domU, its Px can be in domU user level as P3 or domU kernel level as P4, or both of them, 4) when a system is deployed in a hypervisor, its Px can be in only this hypervisor as P5, 5) the notation PaPb such as P1P2 means that an architecture or a system deploys both Pa and Pb. Thus, P1P2 means that an architecture or a system deploys both P1 and P2, 6) PbPa has the same meaning as the meaning of PaPb, and 7) in forms such as Pa/Pb, Pa/disk0, PaPb/PcPd, or PaPb/Pc/Pd/diskU, '/' is a separator notation among the elements of the forms. For example, Pa/disk0 indicates that Pa is in a domain and disk0 is deployed by a system that deploys Pa.

A. The single domain category

We formed the architecture of this category by firstly considering the deployment of Px of a system in either dom0, domU, or a hypervisor, as discussed in the conditions above. Then, to create a final architecture of this system, the system can choose to deploy Fy in appropriate locations: diskU, disk0, or both of them. Considering the deployment of Px, when Px is deployed in a domain, this creates one or more forms of deployment of Px. For example, when a system deploys one Px (such as Pa) in a domain, this creates one form of deployment as: a system, which deploys Pa called a Pa form.

When a system deploys two Px (such as Pa and Pb) in a domain, this creates three forms of deployment as: a system which deploys Pa called a Pa form; a system which deploys Pb called a Pb form; and a system which deploys both Pa and Pb (PaPb) called a PaPb form. Then, to create a final architecture of a system, each form above can choose to deploy Fy in disk0, diskU, or disk0U. For example, a Pa form can deploy disk0 to create a final architecture of a system, which deploys Pa and disk0. We represent this final architecture as 'Pa/disk0'. Thus, each form can create three final architectures. For example, a Pa form creates three final architectures as: Pa/diskU, Pa/disk0, and Pa/disk0U. Figure 3 presents all 21 possible final architectures of this category. From the figure, each branch can be a final architecture of a logging system, for example, observe the three shaded boxes with the dotted-lines labeled '1'. The lines create the branch of dom0, P1, and disk0. This branch is a final architecture of a logging system, and this architecture is represented as dom0/P1/disk0 or for short P1/disk0. The representation means that a logging system that follows this architecture deploys P1 in dom0 user level,

F3 in disk0, and F4 in mem0. The architecture is in the logging system in the spamming case study in [7]. Forming all the 21 final architectures or branches is discussed in the first three paragraphs below.

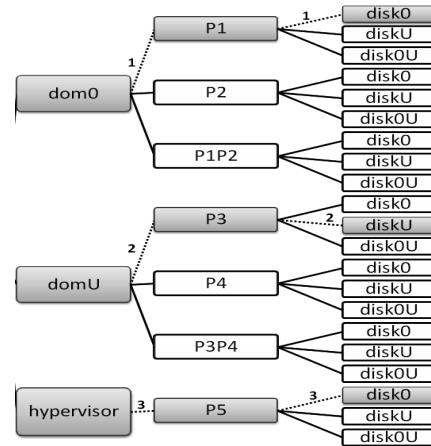


Figure 3: All possible architectures of a single domain category

The nine architectures when a system deploys Px in dom0: Figure 3 illustrates the three branches originating from dom0 box which creates three forms as: a system which deploys P1, a system which deploys P2, and a system which deploys P1P2. These three forms can deploy the three deployment approaches of Fy or condition 1) discussed above. This then creates nine architectures as: P1/disk0 (in the figure from dom0 box, see P1 box and its first branch), P1/diskU, P1/disk0U, P2/disk0, P2/diskU, P2/disk0U, P1P2/disk0, P1P2/ diskU, and P1P2/disk0U.

The nine architectures when a system deploys Px in domU: Figure 3 also shows the three branches originating from domU box which creates three forms as: a system which deploys P3, a system which deploys P4, and a system which deploys P3P4. These three forms can deploy the three deployment approaches of Fy. This, then creates another nine architectures as: P3/disk0 (in the figure from domU box, see P3 box and its first branch), P3/diskU, P3/disk0U, P4/disk0, P4/diskU, P4/disk0U, P3P4/disk0, P3P4/diskU, and P3P4/ disk0U.

The three architectures when a system deploys Px in a hypervisor: Lastly, see the branch originating from the hypervisor box in Figure 3. This creates only one form which is a system which deploys P5. The form can deploy the three deployment approaches of Fy. This creates three architectures as: P5/disk0, P5/diskU, and P5/disk0U. Not all 21 architectures of the single domain category already exist. We have identified three existing architectures. The first architecture is in the logging system in the spamming case study in [7]. Secondly, the dotted-lines labeled '2' in Figure 3 create the branch of domU, P3, and diskU. This is the JAR logging system architecture [15]. The dotted-lines labeled '3' in Figure 3 create the branch of hypervisor, then P5, and disk0. This architecture is for logging systems in [17] and [19].

Systems of the other 18 non-existing architectures are feasible to be built. We did not investigate if all these systems are useful or not; however, these architectures can be used as tools for analysis of systems (e.g., in terms of systems' robustness) that are built based on these architectures. For example, from Figure 3, the following non-existing architectures can be analyzed and compared.

When following the branch dom0/P2/disk0 in Figure 3, this is the first architecture or P2/disk0 non-existing architecture. When following the branch dom0/P1/disk0, the second one is the existing P1/disk0 architecture.

A system built based on the architecture of P2/disk0 should be more robust compared to the architecture of P1/disk0. It would therefore be more difficult for attackers to compromise P2 in the kernel of the first architecture, compared to compromising P1 in the user level of the second architecture. However, full analysis of systems built based on these non-existing architectures is out of the scope of this paper.

B. The two domains category

The two domains category means that all Px of a logging system are deployed in two domains among the three domains (dom0, domU, or a hypervisor). Figure 4 presents the 45 architectures of this category. The dotted-lines labeled '1' in the figure show the branch of dom0U, P2, P4, and disk0U. This branch represents a logging system architecture as P2/P4/disk0U. This means that the architecture deploys P2 in dom0 kernel, P4 in domU kernel, F1 in diskU, F2 in memU, F3 in disk0, and F4 in mem0. This architecture is represented in Figure 4. A group of boxes labeled disk0, diskU, or disk0U represent the architectures. For example, from the first group of boxes at the end of the {domOU @ P1 @ {P3,P4,P3P4}} branch, the top box of the stack is disk0 box which can be repetitively linked back to P3, P1, and dom0U box. This representation is dom0U/P1/P3/disk0 or P1/P3/disk0 architecture and also applies to the discussions of the next category.

We formed the architecture of this category by firstly considering the deployment of Px of a system in the first domain then in the second domain. Finally, to create a final architecture of this system, the system can choose to deploy Fy in disk0, diskU, or disk0U, see condition 1). This section represents a Pa form as just Pa, Pb form as Pb, and PaPb form as PaPb. For example, a P1 form, P2 form, and P1P2 form are represented as P1, P2, and P1P2, respectively. Dom0U is an abbreviation which means that Px are deployed in both dom0 and domU. Dom0H and domUH are also abbreviations which mean that Px are deployed in both dom0 and hypervisor (H) and in both domU and hypervisor, respectively. The first three paragraphs are discussions of forming the 45 architectures.

27 architectures when deploying Px in dom0 then in domU (dom0U): There are three forms of deploying Px in dom0: P1, P2, and P1P2. Figure 4 shows the three branches originated from dom0U box. Then, each form can deploy the other three forms of domU: P3, P4, and P3P4 are the three branches originating from each of the boxes labeled P1, P2, or P1P2 after box dom0U. Thus, this generates nine forms of dom0U: P1/P3, P1/P4, P1/P3P4, P2/P3, P2/P4, P2/P3P4, P1P2/P3, P1P2/P4, and P1P2/P3P4. Finally, these nine forms can choose to deploy Fy by the three different approaches or condition 1). When multiplying these 9 forms by the 3 different approaches of deploying Fy, this generates the 27 final architectures. These architectures are shown in Figure 4, from the first stack of boxes from the right of the figure see the first 27 boxes from the top of the stack.

9 architectures when deploying Px in dom0 then in hypervisor (dom0H): There are three forms of deploying Px in dom0: P1, P2, and P1P2 see the three branches originated from dom0H box in Figure 4. Then, each form can deploy a

form of hypervisor as P5 see the branch originated from each of boxes labeled P1, P2, or P1P2 after box dom0H in the figure. Thus, this generates 3 forms of dom0H: P1/P5, P2/P5, and P1P2/P5. These three forms can choose to deploy Fy by the three different approaches. When multiplying these 3 forms by the 3 different approaches of deploying Fy, this generates the 9 final architectures. They are shown in Figure 4, from the first stack of boxes from the right of the figure see the 10th to 18th boxes from the bottom of the stack.

9 architectures when deploying Px in domU then in hypervisor (domUH): Figure 4 illustrates three forms of deploying Px in domU: P3, P4, and P3P4 being the three branches originating from domUH box. Then, each form can deploy a form of hypervisor as P5 in the branches originating from each of the boxes labeled P3, P4, or P3P4 after box domUH. Thus, this generates three forms of domUH: P3/P5, P4/P5, and P3P4/P5. These three forms can choose to deploy Fy from the three different approaches. These are the 9 final architectures. We have found two existing architectures of this category. One is the architecture of a logging system called Flogger [16] which is the branch of P2/P4/disk0U in Figure 4 and the second is PASSXen system architecture [11] represented by the dotted-lines labeled '2' which create the branch of P1P2/P4/disk0. Systems with the other 43 non-existing architectures are feasible to be built. We did not, however, investigate if all these systems are very useful. However, these architectures can be used as tools for analysis of the systems (such as in terms of robustness of the systems) that are built based on them. For example, from Figure 4, the two following non-existing architectures can be analyzed and compared.

A system built based on the branch dom0H/P2/P5/disk0 (in Figure 4) should be more robust compared to a system built based on the branch dom0U/P1/P3/disk0. It will be more difficult for attackers to compromise P2 in the kernel and P5 in the hypervisor of the first architecture, than compromising P1 and P3 in the user levels of the second architecture. However, full analysis of systems built based on these non-existing architectures is out of the scope of this paper.

C. The three domains category

The three domains category means that at least one Px of a logging system is deployed in each of the three domains. Figure 5 presents 27 possible architectures of the catalogue. We formed an architecture of this three domain category by firstly considering the deployment of Px of a system in the first, second and third domains. Finally, to create a final architecture of this system, the system can choose to deploy Fy with the three different approaches, see condition 1). Dom0UH (the second box from the left of Figure 5) is an abbreviation which means that Px are deployed in dom0, domU, and hypervisor.

To elaborate, the dotted-lines labeled '1' in Figure 5 create the branch of P1/P3/P5/disk0. It is a logging system architecture which deploys P1 in dom0 user level, P3 in domU user level, P5 in a hypervisor, F3 in disk0 and F4 in mem0. The most complicated architecture would follow the dotted-lines labeled '2' in Figure 5. This is the P1P2/P3P4/P5/disk0U architecture which deploys all the nine critical logging components; five Px and four Fy. To form all the 27 architectures, there are three forms of

deploying Px in dom0: P1, P2, and P1P2 which are the three branches originating from dom0UH box in Figure 5. Each form can then deploy the three forms of domU: P3, P4, and P3P4 being the three branches originating in each of the boxes labeled P1, P2, or P1P2 after box dom0UH in Figure 5. Thus, this generates nine forms: P1/P3, P1/P4, P1/P3P4, P2/P3, P2/P4, P2/P3P4, P1P2/P3, P1P2/P4, and P1P2/P3P4.

Each of the nine forms described can deploy a form of domH as P5, illustrated by the branch originating from each of the boxes labeled P3, P4, or P3P4 after the boxes labeled P1, P2, or P1P2 in Figure 5. Thus, this generates nine forms; P1/P3/P5, P1/P4/P5, P1/P3P4/P5, P2/P3/P5, P2/P4/P5, P2/P3P4/P5, P1P2/P3/P5, P1P2/P4/P5, and P1P2/P3P4/P5. Finally, these nine generated forms can choose to deploy the three deployment approaches of Fy or condition 1). When multiplying these 9 forms by the 3 different approaches of deploying of Fy, this generates the 27 final architectures. These architectures are all illustrated in Figure 5, as the stack of boxes at the right extreme of the figure.

We have not found existing architectures of this category so far; however, systems of these architectures are feasible to be built. We also did not investigate if all these systems are in fact useful. However, the architectures in this category can be used as tools for analyzing the robustness of systems based on these architectures in the same way as has been done on the non-existing architectures of the first two categories.

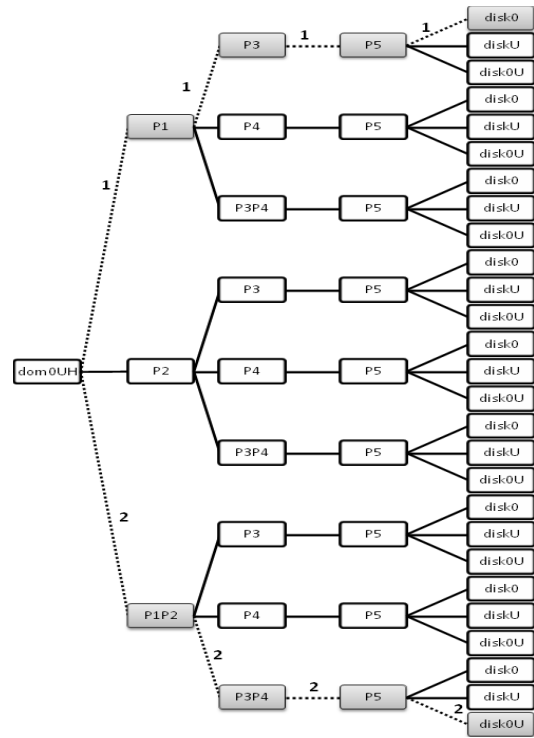


Figure 5: All possible architectures of a three domains category

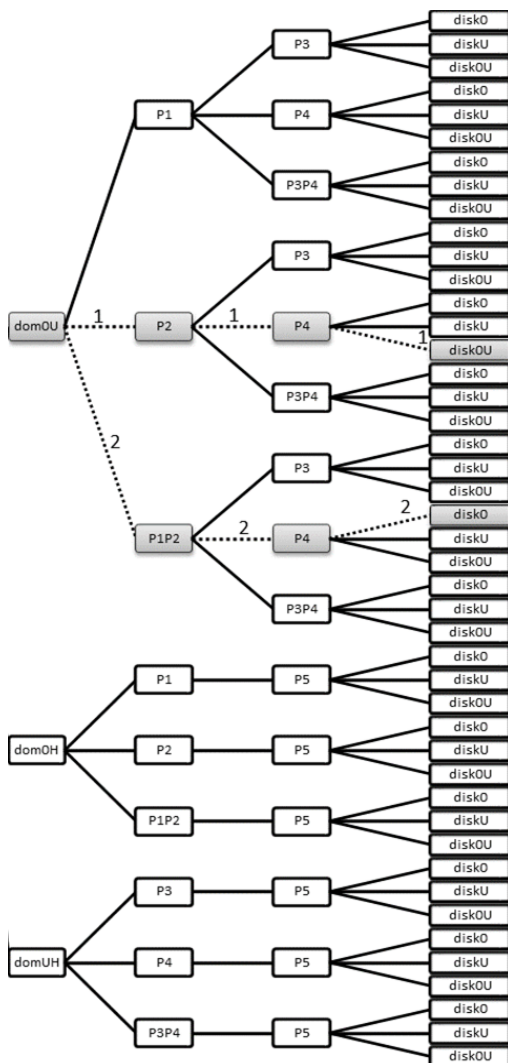


Figure 4: All possible architectures of a two domains category

IV. CONCLUSIONS

We investigated and introduced 93 possible architectures of a logging system, which could exist in complex and abstract cloud environments. These architectures can be used to guide the future design and development of logging systems. This should decrease the effort and time commitments required for the deployment of logging systems. To the best of our knowledge, these 93 architectures are not yet described in the literature. We argue that more research is needed on the topic of architectures of logging systems in the cloud, and encourage other researchers to participate in providing solutions to meet these concerns. The architectures should assist in systematically defining and identifying logging system patterns. Such a design pattern approach would provide a number of benefits similar to that arising from the design patterns concept in object-oriented software design and development. This would truly enable logging systems to work in real world cloud production systems.

ACKNOWLEDGEMENT

Many thanks to Mr. Roy Morien of the Naresuan University Language Center for his editing assistance and advice on English expression in this document.

REFERENCES

- [1] Leitersdorf Y. and Schreiber O.. 2014 Cybersecurity Hindsight And A Look Ahead At 2015. TechCrunch.
- [2] CSA, 2013. The notorious nine: Cloud computing top threats in 2013, The Cloud Security Alliance (CSA), Tech. Rep.
- [3] Armbrust M., Fox A., Griffith R., Joseph A. D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A., Stoica I., and Zaharia M., 2010. A View Of Cloud Computing, Commun. ACM,
- [4] Haeberlen A., A Case For The Accountable Cloud, SIGOPS Oper. Syst. Rev., 44(2):52–57, 2010

- [5] Santos N., Gummadi K. P., and Rodrigues R., 2009. Towards Trusted Cloud Computing, in Proceedings of the 2009 conference on Hot topics in cloud computing
- [6] Lyle J. and Martin A., 2010. Trusted Computing And Provenance: Better Together, in Proceedings of the 2nd conference on Theory and practice of provenance.
- [7] Wongthai W., Rocha F., and van Moorsel A., 2013. A Generic Logging Template For Infrastructure As A Service Cloud, in *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops*.
- [8] Wongthai W., Rocha F., and Moorsel A. v., 2013. Logging Solutions To Mitigate Risks Associated With Threats In Infrastructure As A Service Cloud, in *Proceedings of the International Conference on Cloud Computing and Big Data*.
- [9] Ko R. K., Jagadpramana P., Mowbray M., Pearson S., Kirchberg M., Liang Q., and Lee B. S. 2011, Trustcloud: A Framework For Accountability And Trust In Cloud Computing, *IEEE Congress on Services*.
- [10] Haeberlen A., Aditya P., Rodrigues R., and Druschel P. 2010. Accountable virtual machines, in the USENIX conference on Operating systems design and implementation.
- [11] Macko P., Chiarini M., and Seltzer M., 2011. Collecting Provenance Via The Xen Hypervisor, in *3rd Workshop on the Theory and Practice of Provenance*.
- [12] Dolan-Gavitt B., Payne B., and Lee W., 2011. Leveraging Forensic Tools For Virtual Machine Introspection, *Georgia Institute of Technology*, Tech. Rep.
- [13] Payne B., de Carbone M., and Lee W., 2007. Secure and Flexible Monitoring Of Virtual Machines, in Annual Computer Security Applications Conference.
- [14] Payne B., Carbone M., Sharif M., and Lee W., 2008. Lares: An Architecture For Secure Active Monitoring Using Virtualization, in *IEEE Symposium on Security and Privacy*
- [15] Sundareswaran S., Squicciarini A. C., and Lin D., 2012. Ensuring Distributed Accountability For Data Sharing In The Cloud, *IEEE Transactions on Dependable and Secure Computing*
- [16] Ko R., Jagadpramana P., and Lee B. S., 2011. Flogger: A File-Centric Logger For Monitoring File Access And Transfers Within Cloud Computing Environments, in *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*
- [17] Haeberlen A., Aditya P., Rodrigues R., and Druschel P., 2010. Accountable Virtual Machines, In *Proceedings Of The USENIX Conference On Operating Systems Design And Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association
- [18] Anderson R. 1996 The Trusted Computing Base.
- [19] Beck F. and Festor O., 2009. Sycall Interception in Xen Hypervisor, Inria, Technical Report, [Online]. Available: <http://hal.inria.fr/inria-00431031>
- [20] Gamma E., Helm R., Johnson R., and Vlissides J., 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional
- [21] Hogan M., Liu F., Sokol A., and Tong J., 2011. NIST Cloud Computing Standards Roadmap, Tech. Rep.
- [22] Parkin S. E. and Morgan G., 2012. Toward reusable sla monitoring capabilities, *Software: Practice and Experience*,
- [23] Chow R., Golle P., Jakobsson M., Shi E., Staddon J., Masuoka R., and Molina J., 2009. Controlling data in the Cloud: Outsourcing Computation Without Outsourcing Control, in *Proceedings of ACM workshop on Cloud computing security*.