Towards Purpose-Driven Virtual Machines

Naod Duga Jebessa Guido van 't Noordende Cees de Laat

University of Amsterdam
Science Park 904, 1098XH Amsterdam, The Netherlands
{jebessa,noordende,delaat}@uva.nl

Abstract

Virtual machines (VMs) are often generic though they are meant to serve a specific purpose. The redundancy of generic VMs may incur costs in *security* (due to bigger attack surface and a larger trusted computing base) and *performance* (due to extra VM image size as well as overheads in CPU and memory). We are working on techniques to build minimal, application-specific and secure virtual machines from declarative descriptions. An application running in a VM has dependencies on other applications, libraries, kernel features and (virtual) hardware. We model such a dependency as a graph. The VM is treated as an optimized system built specifically to satisfy this dependency out of a set of interdependent components such as packages and kernels from OS distributions. Such distributions and installations based on them are inherently complex networks that could benefit from a formal, rigorous treatment in order to perform security and performance optimization. In this paper, we describe our motivation and vision for this project, outline our approach, briefly report on current status, discuss challenges and research problems that we intend to work on in the future.

1 Introduction

The virtual machine concept is one of the key enablers of clouds, and has found applications in mobility, security, fault tolerance, grids, testing, and development. VMs are often used to run a set of applications and as such can be thought of as having a specific purpose. Nonetheless, VMs are commonly instantiated from ready-made, generic images or are composed from off-the-shelf components like applications, libraries, operating systems, and device drivers meant for physical machines. For example, a VM image with a general purpose kernel with millions of lines of code (LOC) in its trusted compute base (TCB) can result in an unnecessarily large attack surface. There are vulnerable libraries, applications, protocol stacks and service configurations that could be omitted or otherwise optimized without breaking the functionality of the application in question.

VMs are prone to misconfiguration by users who have to work with generic images that are not optimized, either due to lack of expertise or simply because it is easier to just use an existing, general purpose image. This may incur costs in terms of *performance*, due to extra VM image size, overheads in CPU and memory usage, as well as *security*, due to bigger attack surface and a larger TCB. Moreover, users are faced with VM sprawl as images are created, deployed, changed, snapshotted, cloned and archived, making lifecycle management difficult. Many of these problems can be attributed to the black-box nature of VM images.

We are advocating declarative descriptions of virtual machines so we can build an optimal virtual machine on

In: M. Heisel, E. Marchetti (eds.): Proceedings of ESSoS-DS 2013, Paris, France, 27-Feb-2013, published at http://ceur-ws.org

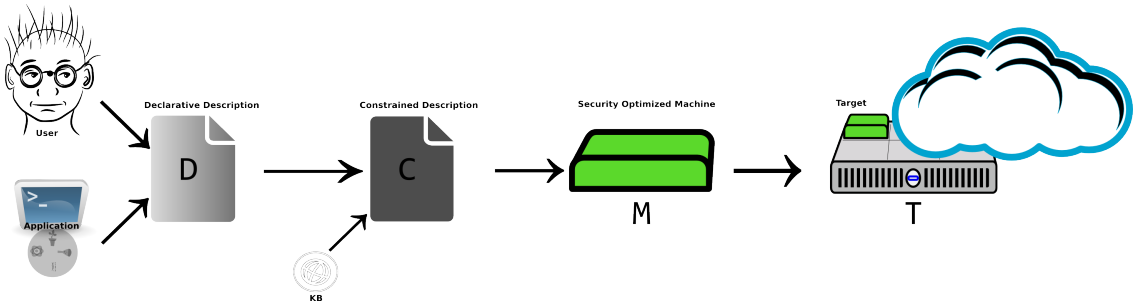

Figure 1: Outputs D, C, M & T after each of the 4 stages in the VM building pipeline

demand from existing components. Doing so should allow us to do fine grained and automated optimization. Moreover, it would be easier to reason over the security or other properties of the system using a semantically rich model, than over an unstructured virtual machine image.

2 Approach

We use graph models for OS distributions and virtual machines built from them. The purpose of the VM is declared using a domain-specific language (DSL [17, 23]). The declarative description is then used as an input to a pipeline used to build an optimized VM. The problem is partitioned into two major subproblems. The first subproblem, corresponding to the two stages in our four stage pipeline (see D and C in figure 1), focuses on descriptions and as such involves language and semantics related issues as well as ways to capture user/application requirements. The second subproblem is about translating the descriptions to concrete systems that meet constraints derived from the second stage. For brevity, we have shown the outputs of each stage while arrows represent intermediate blocks.

2.1 Operating Systems as Complex Networks

It is possible to model an operating system distribution as a graph of interdependent components. An operating system OS_i can have v versions, each comprising a set P of packages with a dependency graph G_p and a kernel whose features [4, 7] can be represented as a dependency graph G_k. It is worth noting that there is an implicit dependency between G_p and G_k as applications and libraries expect a kernel to interact with and because many distributions treat the kernel itself as a package.
Because we can annotate the graph model with external information about, say, security-related statistics on packages, we will be able to employ security-aware constraints in our optimization pipeline. This pipeline is part of a toolchain for building virtual machines that we are developing.
An ongoing work is to use ideas from network science and complexity [1, 2] to understand the structure and dynamics of OS distributions and installations [5]. As an example, we created an annotated semantic graph of `Debian Squeeze` that comes with a `Linux 2.6` kernel for `i386`. This graph has approximately 40 thousand nodes and 170 thousand edges. Figure 2 shows a visualization of the graph. One can imagine how challenging it is to manipulate this model and gain insight from it. However, we simplify the problem by focusing on the dependency graph of a small set of applications on a specific OS, as shown in figures 3 and 4.

2.2 Declarative Descriptions and Constraints

The VM is declaratively described in a domain-specific language (DSL). The description specifies minimal requirements from the VM such as set of packages, kernel features and system configurations for software, (virtual) hardware, networks, and so on. This description is then analyzed to fill in details and create a constraint for optimization, as shown in figure 1. The DSL should be expressive enough and should allow explicit declarations

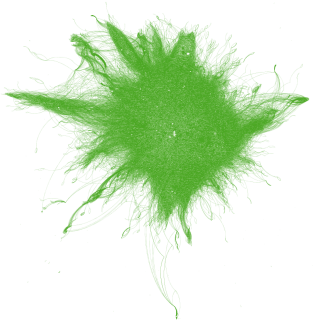

Figure 2: An OS as a complex network

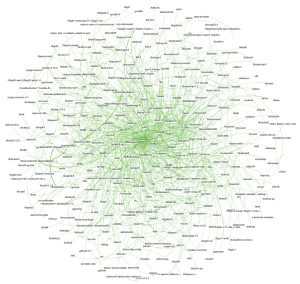

Figure 3: Minimized network for a specific application. The center node is `libc`, the C runtime library for Unix-like systems.

through conjunction, negation, and disjunction. For example, a description may have a list of requirements that must be satisfied (conjunction), unneeded functionality (negation), and alternatives for a certain feature (disjunction). Implicit requirements in the VM will be incorporated as default constraints upon analysis. For example, if the declarative description does not state what version of an OS to use, a default version may be selected. Declarative descriptions and constraints have several advantages. They can be versioned to allow rollback; can be based on existing templates; and can be reasoned about (say, to map a declaration/constraint to an already built VM).

2.3 Optimization and Reasoning About VM Security

We look at the VM as the minimal composition from a set of interdependent components in package dependency graphs G_{p_i} and kernel feature graphs G_{kj} that satisfies the constraint set forth by the declarative description. Dependency resolution is an NP-complete problem [3]. The fact that we add constraints might lead to declarations with constraints that are not satisfied. From the package dependency graph point of view (see figure 4 for a specific application[1]), the problem is finding an optimal installation that satisfies constraints pertaining to packages [8]. For the kernel, this translates to (de)selecting features based on constraints on the kernel, specifically device drivers, file systems and protocol stacks [4, 7].

Let us consider the case where there is a security advisory for a network exploitable vulnerability in package `X` version `a.b.c` for a specific OS. A query to the declarative description and the blueprint of the VM would show if we have that specific version in the virtual machine. This could be automated and can be an integral part of a simplified security lifecycle management pipeline.

A practical problem with this approach is that security advisories (like CVE/DSA[2]) are not semantically rich enough. We see promising research with practical significance in employing semantic technologies to describe, consume and process security-related information.

2.4 Building Virtual Machines

Having installed an optimized package set on top of a possibly optimized kernel, a VM image can be configured with cryptographic keys, users, passwords and the like along with extra dependencies like memory, processor, storage and network settings to a specific target hypervisor.

We are building a proof-of-concept setup to validate the ideas presented so far. There are at least four engineering challenges. First, there are quite a few hypervisors to target, each with a somewhat different view of a guest

[1] `http://fsl.fmrib.ox.ac.uk`

[2] `http://cve.mitre.org/`, `http://www.debian.org/security/`

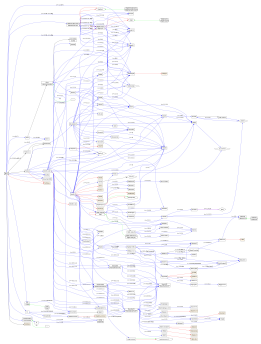

Figure 4: Partial dependency graph for a specific application

VM. To this end, we plan to have a modular and extensible VM building pipeline. The design of the hypervisor might also constrain how we optimize the VM and it is worth considering hypervisor-VM optimization in this regard, e.g. with respect to drivers.
Second, there are a range of VM image formats and containers to target. The *qcow2* format used by KVM/QEMU, for instance, is quite different from formats like Xen raw disk, AMI (Amazon, with separate kernel and ramdisk), Microsoft Hyper-V's *vhd*, VirtualBox *vdi*, and *vmdk* (VMware). However, a modular design should allow us to build a VM and target it to different formats. This is possible because most formats are more or less indifferent to how we optimize the internal subsystems of the VM and because there are readily available conversion tools.
Third, there are a plethora of operating systems that we can build virtual machines from. Modeling each OS as an annotated dependency graph is a difficult endeavor. Hence, we plan to work on a set of representative OS distributions. Even though our approach is ideal for the Unix philosophy and package-based source/binary OS distributions, it should be possible to extend the idea to other platforms, as the formalism we are developing is fairly generic.
Last but not least, there is the issue of efficient VM building. One might consider having a ready made 'base' VM setup for commonly used virtual machines and configure a candidate based on the constraints set forth by the declarative description. This approach may be useful when the time to build and deploy is of significant importance, as building a custom kernel takes several minutes and package installation and configuration takes significant time as well.

3 Research Challenges and Future Work

1. *How do we manage complexity and heterogeneity?*: One challenge is coming up with a fairly generic abstraction that can be used to model a VM based on any one of a range of OS distributions; built for one or more hypervisors and processor architectures; and can be targeted at one or more cloud stacks. We believe that dependency graphs and complex networks are a viable foundation to start with, together with an information model for each component from which the VM is composed from. In such an approach, we treat the VM as a layered stack of dependencies in hardware, hypervisor, kernel features, libraries, applications and configurations. Such a graph model allows us to separate theoretical problems from implementation issues. The model would benefit from existing research in complex networks [1], graph theory [2, 12, 21], boolean satisfiability and dependency resolution [3, 19, 13, 14]. We could leverage existing tools and demonstrate our ideas through a proof of concept pipeline instead of a complete implementation that supports all major operating systems and hypervisors.

2. *How do we describe the specific purpose of a VM?*: People often use natural languages to describe what a specific machine is for, be it physical or virtual, even though the machine is, in principle, general-purpose.

The design of a declarative language for VM description calls for a tradeoff between simplicity and expressiveness. The first challenge arises from the fact that a user cannot be expected to explicitly declare everything. So the language [20, 23] should have constructs to support implicit declarations. Second, the expressive power of the language dictates much of what is done after processing a description. For example, a very fine-grained description (e.g. with conjunctive normal form constraints) may be difficult to satisfy, due to a limited search space and making the optimization problem at hand difficult (as in too many variables and constraints) and time-consuming (translating the solution to a concrete VM). A coarse-grained description, on the other hand, may introduce indeterminism, sub-optimal or too-optimized solutions, and an increased search space.

3. *How do we fill the semantic gap in security?*: To our surprise, much of the available security-related information (e.g. from CVE or specific vendors) is not structured enough to be used by automated tools. In our particular case, for example, it is not easy to gather vulnerability information so as to measure or evaluate the security of a VM instance due the ambiguity, incompleteness and a non-standard vocabulary, to mention a few, of security advisories. Along with our work in measuring security of a VM and on attack surface evaluation metrics, we plan to suggest a semantically rich, machine-readable format to describe vulnerabilities and develop a logic formalism for attack scenarios, specially for use by OS distributions like Debian.

4. *What is this all for?*: In cloud parlance, our idea is conceptually equivalent to offering a PaaS on demand, targeted at an existing IaaS. A good example might be creating a VM cluster for a DNA processing application that expects a certain runtime environment, including an OS installation and all required dependencies. Hence, our project essentially maps high-level application requirements to low-level infrastructure details. Moreover, some application scenarios are security and privacy sensitive. Hence, we are advocating a disciplined way of building VMs with the hypothesis that a pipeline that translates declarative descriptions to concrete VMs would allow for 'white-box machines' (whose internal manifest is clearly known), hence allowing us to reason about certain attributes of such a VM like its trusted compute base (TCB) size, the size of its attack surface, the possibility of exploitation given an attack scenario, or trustworthiness in general. While security is the main goal of our work, the approach could have advantages in smaller VMs that could perform well [18] and are easy to maintain, allowing users and frontend tools to automatically build virtual machines. As future work, we plan to build demos and explore use cases and usability as the feedback from such endeavors will allow us to refine our approach to the problem.

5. *Validation and Evaluation Techniques*: We are working on the assumption that optimized, purpose-driven VMs are likely to be more secure and could possibly perform better. In addition, our approach of employing a declarative VM description language might improve usability, flexibility and maintainability. All the above hypotheses need to be tested. The security of a VM is not straightforward to quantify due to the absence of an evaluation metric. However, having the knowledge of the VM internals should allow us to propose novel metrics that we can use to compare the security of optimized VMs with their generic counterparts. Performance of the VM building pipeline and the built VMs, on the other hand, can readily be evaluated in terms of time taken to build a VM (compared to manual building and off-the-shelf deployment of an existing VM image); the CPU and memory usage while the VM is running; and VM image size (which is important during migration, for example). Usability, flexibility and maintainability could be evaluated through use cases that make use of our tools, albeit subjectively.

4 Related Work

There are quite a few projects that aim at building VMs [11], image manipulation[3], configuration tools & APIs[4], and interaction with hypervisors[5], the majority of which are open source. We are aware of at least half a dozen solutions focusing on VM building, with varying maturity and OS support. Our work differs in two major ways and is, in essence, orthogonal to most. First, building a VM is only part of what we do and as such it can be considered as an implementation issue. Second, we are introducing a purpose-driven (declarative) VM-building paradigm wherein we do fine-grained optimization with implications in security, performance and flexibility, as opposed to straightforward installation as is often done by existing solutions[6].

[3] `http://libguestfs.org/`
[4] `http://augeas.net/`, `http://cfengine.com/`
[5] `http://libvirt.org/`
[6] `http://wiki.debian.org/VMBuilder`, `http://virt-manager.org/`

Few authors have studied operating system distributions as complex networks [1] and we believe there is more to be done, specially in the context of security, OS complexity and virtual machines. The authors in [3, 5] have done detailed studies on formalisms and tools to manage complexity of package-based OS distributions showing that they exhibit the small world property [2] of many networks. We hope to work towards a graph theoretic attack surface metric [10] that takes into consideration semantic metadata about components and the composition thereof, of a virtual machine. This would allow us to evaluate the security advantages of minimal and application-specific VMs.
In the domain of software engineering, researchers have studied automatic generation [15, 16, 22], predictable assembly [6], variability and feature models [4, 7, 9] as applied to OS kernels.

5 Conclusion

For many users, a virtual machine is a black box that is difficult to reason about, trust, configure and maintain. While this is most certainly true for inexperienced users, expert users also find themselves working with pre-configured or generic VM images, which is anything but transparent. Furthermore, automated infrastructure provisioning tools are increasingly using virtual machine technology to serve computational needs by deploying VMs on demand, often from pre-built images. We believe that there is a missing link between the *purpose of a VM* on one hand and *the way a VM is built* on the other. We need a generic and disciplined process to build our virtual machines.
Our work fits in the wider problem of security and privacy in clouds. One aspect is having trustworthy VM installations that can be (remotely) attested. The user might want to just declare and expect to get a fairly secure VM while the cloud provider might expect to have some guarantee that the VM is 'safe' to deploy. Depending on the use case, the declarative description and intermediate steps in our pipeline could be used or extended to incorporate mutual trust, where both parties can verify relevant properties of the virtual machine. Another aspect is the issue of privacy in clouds which we believe can benefit from a trustworthy execution environment. If a VM can be attacked and owned, it will be a breach of the necessary condition of confidentiality of the data it is supposed to process (as part of its *purpose*), secret keys it has stored, etc.
In this paper, we have described the idea of a *purpose-driven virtual machine*. Our motivation is the observation that a VM is often meant to serve a specific purpose and the fact that generic VMs have redundant components that may be omitted, giving advantages in security and performance. Using declarative descriptions of application requirements, we are able to generate virtual machines on demand out of component ecosystems (i.e. OS distributions) modeled as complex networks. We have discussed our approach and outlined some theoretical and practical problems that we intend to work on in the future.

Acknowledgment: This research is supported by the Dutch national research program COMMIT (*http://www.commit-nl.nl*). We would like to thank the anonymous reviewers whose comments helped improve the paper.

References

[1] M. Newman, A-L. Barabasi, and D.J. Watts. 2006. The Structure and Dynamics of Networks: (Princeton Studies in Complexity). Princeton University Press, Princeton, NJ, USA.

[2] J. Kleinberg. 2000. The small-world phenomenon: an algorithm perspective. In Proceedings of the thirty-second annual ACM symposium on Theory of computing (STOC '00). ACM, New York, NY, USA, 163-170.

[3] F. Mancinelli, J. Boender, R. di Cosmo, J. Vouillon, B. Durak, X. Leroy, and R. Treinen. 2006. Managing the Complexity of Large Free and Open Source Package-Based Software Distributions. In Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE '06). IEEE Computer Society, Washington, DC, USA.

[4] T. Berger, S. She, R. Lotufo, A. Wąsowski, and K. Czarnecki. 2010. Variability modeling in the real: a perspective from the operating systems domain. In Proceedings of the IEEE/ACM international conference on Automated software engineering (ASE '10). ACM, New York, NY, USA, 73-82.

[5] P. Abate, R. di Cosmo, J. Boender, and S. Zacchiroli. 2009. Strong dependencies between software components. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09). IEEE Computer Society, Washington, DC, USA, 89-99.

[6] S.A. Hissam, G.A. Moreno, J.A. Stafford, and K.C. Wallnau. 2002. Packaging Predictable Assembly. In Proceedings of the IFIP/ACM Working Conference on Component Deployment (CD '02), J.M. Bishop (Ed.). Springer-Verlag, London, UK, UK, 108-124.

[7] R. di Cosmo and S. Zacchiroli. 2010. Feature diagrams as package dependencies. In Proceedings of the 14th international conference on Software product lines: going beyond (SPLC'10), J. Bosch and J. Lee (Eds.). Springer-Verlag, Berlin, Heidelberg, 476-480.

[8] P. Abate, R. di Cosmo, R. Treinen, and S. Zacchiroli. 2013. A modular package manager architecture. Inf. Softw. Technol. 55, 2 (February 2013), 459-474.

[9] L. Passos, K. Czarnecki, and A. Wąsowski. 2012. Towards a catalog of variability evolution patterns: the Linux kernel case. In Proceedings of the 4th International Workshop on Feature-Oriented Software Development (FOSD '12), Ina Schaefer and Thomas Thm (Eds.). ACM, New York, NY, USA, 62-69.

[10] P.K. Manadhata and J.M. Wing. 2011. An Attack Surface Metric. IEEE Trans. Softw. Eng. 37, 3 (May 2011), 371-386.

[11] I. Krsul, A. Ganguly, J. Zhang, J.A. B. Fortes, and R.J. Figueiredo. 2004. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In Proceedings of the 2004 ACM/IEEE conference on Supercomputing (SC '04). IEEE Computer Society, Washington, DC, USA, 7-.

[12] M. Newman. 2010. Networks: An Introduction. Oxford University Press, Inc., New York, NY, USA.

[13] G. Jenson, J. Dietrich, and H.W. Guesgen. 2010. An empirical study of the component dependency resolution search space. In Proceedings of the 13th international conference on Component-Based Software Engineering (CBSE'10), L. Grunske, R. Reussner, and F. Plasil (Eds.). Springer-Verlag, Berlin, Heidelberg, 182-199.

[14] L. Bordeaux, Y. Hamadi, and L. Zhang. 2006. Propositional Satisfiability and Constraint Programming: A comparative survey. ACM Comput. Surv. 38, 4, Article 12 (December 2006).

[15] L. Guthier, S. Yoo, and A. Jerraya. 2001. Automatic generation and targeting of application specific operating systems and embedded systems software. In Proceedings of the conference on Design, automation and test in Europe (DATE '01), W. Nebel and A. Jerraya (Eds.). IEEE Press, Piscataway, NJ, USA, 679-685.

[16] A. Sarmento, L. Kriaa, A. Grasset, M-W. Youssef, A. Bouchhima, F. Rousseau, W. Cesario, and A.A. Jerraya. 2005. Service dependency graph: an efficient model for hardware/software interfaces modeling and generation for SoC design. In Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '05). ACM, New York, NY, USA, 261-266.

[17] A. van Deursen, P. Klint, and J. Visser. 2000. Domain-specific languages: an annotated bibliography. SIGPLAN Not. 35, 6 (June 2000), 26-36.

[18] N.D. Jebessa, G. van 't Noordende, C. de Laat. 2012. Optimizing Security for Virtual Machine Applications. In HPDC 2012. The 21st International ACM Symposium on High-Performance Parallel and Distributed Computing (Poster Abstract). (July 2012)

[19] T. Zimmermann, and N. Nagappan. 2007. Predicting subsystem failures using dependency graph complexities. In Software Reliability 2007. ISSRE'07. The 18th IEEE International Symposium on, pp. 227-236. IEEE, 2007.

[20] R.J. Stainton. 1996. Philosophical perspectives on language. Peterborough, Ont., Broadview Press.

[21] S.E. Schaeffer. 2007. Survey: Graph clustering. Comput. Sci. Rev. 1, 1 (August 2007), 27-64.

[22] K. Czarnecki and U.W. Eisenecker. 2000. Generative Programming: Methods, Tools, and Applications. ACM Press/Addison-Wesley Publ. Co., New York, NY, USA.

[23] M. Mernik, J. Heering, and A.M. Sloane. 2005. When and how to develop domain-specific languages. ACM Comput. Surv. 37, 4 (December 2005), 316-344.