Securing Virtual Applications Using Software Process Controls

Dr. Edward G. Amoroso
Chief Executive Officer, TAG Cyber LLC
Mob: (201) 454 – 1854
eamoroso@tag-cyber.com

Abstract
As information technology shifts from physical endpoints communicating over networks, to virtual cloud workloads making functions calls over APIs, cyber security protections must be adjusted accordingly. Software process maturity management will reduce the risk of malware corruption during the compile and run-time phases of the software lifecycle in support of virtual computing. The Building Security in Maturity Model (BSIMM) thus exemplifies an important new type of control required for securing cloud applications and infrastructure.
(Analyst Note: This short paper was inspired by technical discussions about software security and BSIMM in March 2016 with the Cigital team led by Gary McGraw.)

From Hardware to Virtual Threat Management

The traditional distributed computing model includes two endpoints, Alice and Bob, communicating over a physical communication medium such as an Internet protocol (IP) network. Alice and Bob are autonomous in the model, and could be implemented in various mutual arrangements, including peer-to-peer. But in the vast majority of cases, Alice is a client and Bob is a server.

The model is generally skewed toward the endpoints being geographically remote, sending and receiving frames locally, and forwarding and handling IP packets over a wide area network. That is, Alice is usually associated with a human being carrying something like a mobile device, and trying to access resources from Bob, who is a server accessible over a broadband connection. In the model, Eve is a malicious entity with access to the endpoints as well as the communication medium they share.

This familiar Alice-Bob-Eve model is central to modern networking, and is usually depicted in the first chapter of just about every modern information security textbook as follows:

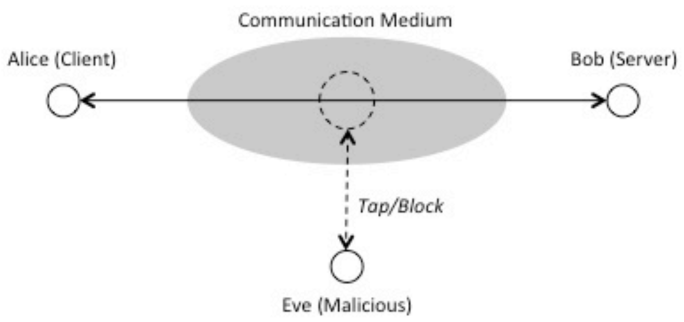


Figure 1. Traditional Distributed Computing Model for Cyber Security

As one would expect, this traditional model encourages exactly the type of cyber security functional controls available from vendors today in the marketplace. Firewalls, intrusion prevention systems, and encryption, for example, overlay perfectly onto the model with emphasis on protecting the endpoints and the data they are exchanging over a network. In the model, Eve also relies on physical means to hack the desired target, using network probes for eavesdropping or botnets to create denial of service attacks.

Recently, however, information technology has begun to shift toward a new method of distributed computing based on a software-oriented technique known as *virtualization*. A new communications model thus emerges, with software endpoints vAlice and vBob making function calls across application programming interfaces (APIs). The underlying infrastructure in this model will be a cloud operating system such as OpenStack or VMWare, rather than a physical network. The primary motivation for virtualization, by the way, is reduced cost, simplified operations, and faster service deployment – even though the security benefits are many.

Here is how the new virtualized model might be now depicted in modern software engineering, operating systems, and cyber security texts:

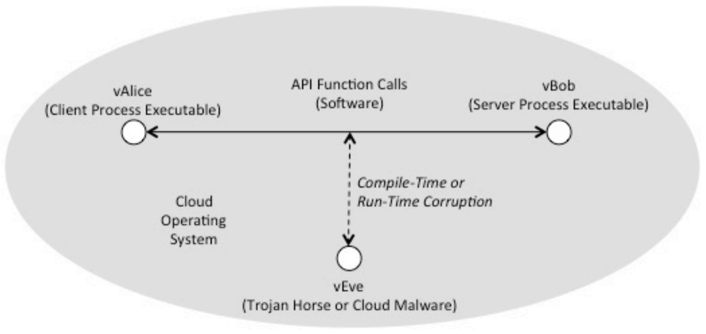


Figure 2. New Virtualized Cloud Operating System Model for Cyber Security

The threat in this new virtual model involves malware, depicted as vEve, being inserted into the endpoint source code at compile time, or into the cloud operating system, which degrades the run-time environment for the endpoint and API interaction. These software attacks require planning by the adversary to determine which phase of the software process is best to attack. This shift is profound because the cyber threat landscape expands to include the software development lifecycle (SDLC) rather than just the visible entry-points of a system on a network.

Since this newly expanded threat vector involves both compile and run-time attacks, the security controls must cover all phases of the SDLC. During software design and development, for example, source code reviews, non-reuse of potentially unsafe code, and attendance to good programming practices are all recommended. These deliberate controls are especially important in Agile Dev/Ops, where the pressure to move more quickly based on customer demands might result in developers skipping through security steps.

In addition to compile-time controls, run-time security will also be required in the underlying cloud operating system. A good example security control would be resource and memory protection through the use of software containers wrapped around cloud workloads, both on the client and in the server. These advanced software-based controls will help to ensure that executing cloud objects will not exceed their allocated privilege or resource allocations.

Software Process Improvement as a Security Control

Virtual endpoints making API calls across a cloud-operating environment is a reprise of the operating system model in the 1980's. Readers will recall early operating system security approaches building on software processes employing interprocess communication (IPC) data structures to interact over Unix or Windows. The primary difference between the original model thirty years ago and the current model is that physical networks are now replaced almost entirely by software.

Because of this difference, software endpoints will be obligated to demonstrate security compliance with audit and regulatory requirements, and this will have to be done through the use of software controls. Such new emphasis changes the compliance process for frameworks such as PCI-DSS because it shifts the focus to software architectures. Unfortunately, software engineers have long struggled with the process of demonstrating that software meets its requirements, so clearly a new compliance challenge is emerging. Let's look at a simple example of how this manifests for security:

Suppose that a retail company connects its physical, in-store Point of Sale (POS) terminals to its credit card processing servers over an enterprise network. PCI-DSS requirements demand that the integrity of this familiar arrangement be assured end-to-end for all so-called "east-west" traffic. Traditional compliance processes would include direct inspection of the physical equipment, verification of traffic characteristics based on scan output, review of applicable system audit logs, inspection of data leakage prevention output, and examination of Web application firewall (WAF) rules.

If that same POS and card-processing environment were virtualized over a cloud operating system, however, then the PCI-DSS compliance process changes. In the new process, what is being required is essentially a demonstration that the cloud workload and operating environment, both implemented entirely in software, meet the corresponding PCI-DSS requirements. This measurement of software trust is incredibly hard to achieve – in fact, it might be intractable – by simply examining source code or running a series of tests. Software engineers have known for many years that this singular approach helps show the presence of problems, but rarely their absence.

As a result, compliance managers are urged to think more holistically to achieve levels of desired software trust. Specifically, they are urged to consider focusing on the characteristics and maturity of the *software process* used to create the code. This focus is best accomplished through data collection and monitoring of the software process against some sort of qualitative and quantitative maturity framework. Developers presumably would thus work hard to achieve higher levels of process maturity to achieve desired levels of compliance – and the software would be much more likely to meet its requirements.

One excellent side effect of this process emphasis is that the application and operating system security controls move from being reactive to preventive. In additional, higher levels of assurance can be achieved that the cloud workload and cloud operating system source code was created in a manner that supports process security and compliance requirements. PCI-DSS compliance assessments thus become focused on building greater trust based on the software process, rather than by just trying to run a series of compliance tests on the code. Further good news is that every software engineer will agree that proactive design protections are always less expensive than react testing.

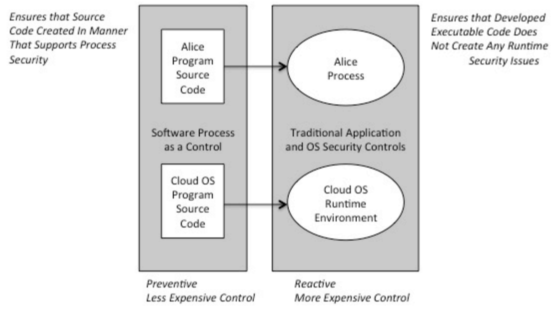

Ensures that Source Code Created In Manner That Supports Process Security
Alice Program Source Code
Software Process as a Control
Cloud OS Program Source Code
Alice Process
Traditional Application and OS Security Controls
Cloud OS Runtime Environment
Ensures that Developed Executable Code Does Not Create Any Runtime Security Issues
Preventive Less Expensive Control
Reactive More Expensive Control

Figure 3. Software Maturity as a Security Control

The maturity estimation should not be viewed as a separation from tradition, familiar controls such as testing. In fact most maturity models demand evidence that code review, penetration testing, documentation control, software integrity checking, and other proactive controls are embedded in the process. The use of maturity as a security control is thus more holistic and comprehensive, and places the burden of compliance responsibility on the SDLC – which is where it belongs.

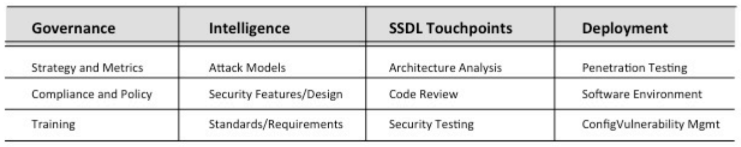
BSIMM V6 Overview

Perhaps the most prominent model for software security process maturity estimation and management is known as the Building Security in Maturity Model (BSIMM). Spearheaded by Cigital, the model is built on three fundamental notions:

- *Measurement* –Maturity of a given software process is believed to be best measured through continuous data collection.
- *Comparison* – An organization presumably can use BSIMM data to compare its process maturity with peer organizations.
- *Science* – An organization should be driving its software security initiatives and related improvements based on sound, data-driven fundamentals.

Existing data has been collected for BSIMM (now up to Version 6) from almost a hundred participating companies in the financial, technology, software, consumer electronics, healthcare, insurance, retail, and telecommunications sectors. Each of the companies within these sectors makes use of BSIM in a structured manner, with twelve different software security practices supporting 112 different activities. All of these practices and activities roll up to four different software security domains.

The first domain is *governance*, which supports strategy, metrics, compliance, and policy; the second domain is *intelligence*, which supports attack models, security features, standards, and requirements; the third domain is the *software security development lifecycle (SSDL)*, which includes touch points for architecture, analysis, and code review; and the fourth domain involves *deployment*, which supports penetration testing, software environment, configuration management, and vulnerability management.



Governance	Intelligence	SSDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features/Design	Code Review	Software Environment
Training	Standards/Requirements	Security Testing	ConfigVulnerability Mgmt

Figure 4. BSIMM v6 Domain Structure

By using the BSIMM for process maturity, software security teams establish commonality in the underlying process framework, which is likely to ease the transition for compliance managers. Furthermore, because the BSIMM is data-driven, security regulatory and compliance managers are more likely to accept a good rating as an acceptable control for virtualized systems requiring certification against standards such as PCI-DSS.