All.Net Analyst Report and Newsletter

<u>Welcome to our Analyst Report and Newsletter</u>

B-BOM and SID

Most current technology for detecting problems, malicious or not, in computer systems and networks, involves methods for detecting known bad, changes from known good, and tracing provenance. But many, perhaps most, actual detections seem to stem from somebody noticing something they don't expect and looking at it further. I describe a pair follows:

- **Behavioral Bill of Materials:** A description of the expected behaviors of components and resulting composites.
- **System Inconsistency Detection:** A method for detecting two or more things that should never both be true if a system is working as intended.

Bills of material

Lists of things that should be present along with associated descriptions / parameters / operating conditions, etc. have long been used to describe a combination of inventory and specifications used by designers, engineers, implementers, maintainers, operators, and users. When I but self-assembled furniture, I get a bill of materials and descriptions I use to figure out how to follow the assembly instructions, and if I use it properly, the chair doesn't collapse when I set in it. The Digital Bill of Materials (DBOM) is a recent invention based on a longstanding concept, extended to the networking environment and supply chain provenance problems associated with local and interdependent components and composites. While this is in its infancy, the Software Bill of Materials (SBOM) has been improving since the "make" command and makefile were used in early versions of Unix and probably elsewhere before.

We also have so-called "indicators of compromise" (IOCs) that are used to indicate things you can look for to determine whether a known malicious mechanism is or has been present. These indicator range from so-called rat droppings (i.e., actual animal feces or detectable residual digital traces of Remote Access Trojans), which indicate the current or previous presence of a rat (or RAT). Again, these tend to be things we are looking for. There are also methods for seeking things that should be present and are not (inconsistency detection is later in the article). 3-factor authentication, as a term of art, stems from the concept of something you have, something you are, or something you know. Again these are 'something's.

- It is important to note that these methods tend to be inventory related. You have or do not have something and you can find it or it's absence by looking for it.

Behavioral-based methods are also out there, and long have been. But our capacity to detect behavioral anomalies or known bad behaviors has long been limited by our ability to describe behaviors. We can describe the bit sequence of a program, and the URL used in a "get" request, and we can look for these things. But while a thing (hardware, software, etc.) can be examined at rest to determine what is and is not there, predicting its behavior based on its appearance is problematic in many ways, not the least of which that the detection of behavior from appearance is undecidable. This was shown for computer viruses in the 1980s, but it is widely accepted as similar for a wide range of other security-related issues.

Behavioral Bills of Material

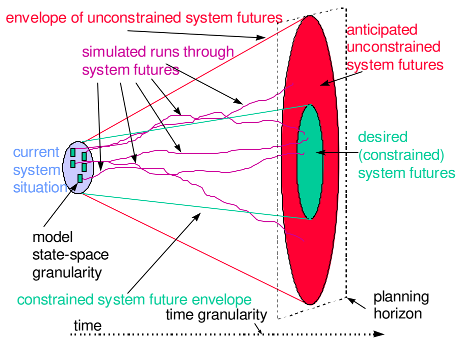
The logical extension of other BOMs is the Behavioral Bill of Materials (B-BOM). Just as any other inventory system, we start with a list of things (including living things), and add in descriptions of how they are supposed to behave. Problem solved… not really…

Important to note is the distinction between how something is supposed to behave and behavioral anomaly detection and behavioral intrusion detection.

- **Anomalies** are things that are different from the norm, but not necessarily supposed to happen or not. They are just different, and we have to investigate to try to determine if things are going right or wrong.
    - *For example, every Tuesday at 8PM, the system sends a stream of content to another system, and this has been true for a year, when all of a sudden, this Tuesday, it didn't happen. That's different from usual, an anomaly, but we have no idea of whether it's something desired or undesired until we investigate.*
- **Intrusions** are generally conceived as known bad event sequences that we know do not want to happen, and described by what we expect the known bad thing to do.
    - *For example, multiple attempts to use network login protocols with a sequence of identifiers and authenticators that are not authorized or don't exist in the system, followed by a single successful attempt, followed by internal uses, all in a short time frame and from the same apparent source. That's a rat dropping detectable by audit mechanisms that record activities and is reasonably concluded to be a strong indicator of an intrusion.*

A B-BOM is, notionally, a bill of materials which provides an envelope of behaviors expected of a component or composite. In other words, the set of behaviors we anticipate can and cannot happen, that constrain the behaviors we should see and not see if we observe them. This is different from these other things discussed here, in that

- We can definitively say, based on the B-BOM specification, that if we see some sequences of observables, they indicate something that should not happen, and if we do not see some other sequences of observables, this indicates something that should not happen.


envelope of unconstrained system futures
anticipated unconstrained system futures
simulated runs through system futures
current system situation
desired (constrained) system futures
model state-space granularity
constrained system future envelope
time granularity
time
planning horizon

With this as a concept, and using our longstanding notion of model-based situation anticipation and constraint, it makes sense that we should be able to identify desired and undesired future situations en route from the current situation as we observe behaviors in the context of a given B-BOM specification. Even if we have not exceeded the desired anticipated future state, observing something the specification in the B-BOM does not indicate as anticipated or indicates as unanticipated, may have the potential to drive the future outside of the desired envelope. In such cases, the moves we make to constrain the newly anticipated potentially undesired future state may be used to adapt the overall composite system so as to remain withing the desired anticipated future state over the planning horizon.

System Inconsistency Detection (SID)

Major challenges associated with implementing an S-BOM include how to specify observable sequences that are known to be present and absent during the normal operation of a mechanism, and given such a specification, how to operationalize it to detect actual things that should not happen.

- **Inconsistencies** are two or more things that cannot happen in conjunction with each other under normal operating conditions. They are not just anomalies, in that they could be just fine; and they are not known intrusions in that observation does not imply we know what went wrong; they are identified sets of observable sequences (behaviors) that cannot possible be present together in a properly operating environment.

In order to operate a B-BOM-based system that identifies and detects, and possibly even reacts and adapts, to failure to meet the identified constrained behavioral characteristics, we need both the ability to specify these constraints and the ability to detect and identify behavioral inconsistencies within a composite and/or between components in such a composite. In other words, System Inconsistency Detection (SID).

So let’s start using them!

To be clear, only a very limited amount of the underlying fundamental work has been published to date in the development of mechanisms for operational B-BOM and SID systems.

The history of this area includes things like indications and warnings systems in which indications (e.g., mobile launchers have been moves into a known launch configuration …) are identified for known undesired event sequences (e.g., atomic weapons launch) and associated warnings are given (alarms start sounding in a control room somewhere along with explicit pre-defined information on what is anticipated). Based on the warnings, trained operators prepare for the anticipated sequence and perhaps undertake identified responses. As such systems are identified and defined, indicators lead to the development of sensors and analysis mechanisms for those sensors, and ultimately this leads to an overarching control system design.

But doing this for nuclear weapons launches is quite different from doing it for everyday operations of everyday systems. The behavioral sequences for strategic defense take years of effort to identify, develop, deploy, and operationalize, and this level of cost and attention is not justified for my cell phone. Or is it?

There have been minimal versions of limited SID used for forensic analysis in high-valued legal matters, and some tools have been developed for this purpose. But the current sets of tools are limited to two general classes; tools like those that that match program specifications to programs for software correctness proofs and similar sorts of mathematical methods; and tools created for limited analysis in special circumstances, such as detecting a login by a person at a physically constrained location when the person was identified as being in some other location at the time (or more generally not being at the constrained location).

We simply don’t have a good language for describing behaviors and the capacity to turn such a language into operational detection capabilities.

What about change control as an interim step?

OK – so we don't know what it's supposed to do or not supposed to do and cannot describe it yet. Why not just start with something we assume is good and go from there?

This is the concept underlying the integrity shell and cryptographic hashes for change detection. It has been around for a long time, and so-called "white listing" methods today use this technique to detect, and in real-time use detection to prevent the use of, unauthorized changes. It works great. But what about authorized changes?

What's supposed to happen when there are changes in systems that require reasonably high surety (certainty that they operate as desired or specified) is that changes are supposed to be controlled through a known method called (loosely) sound change control.

- **Sound change control** consists, generally of the following:
    - Changes are identified as to what is to be changed and why, and the reason for the change is checked against the identified change to determine that the change is applicable (will in fact achieve) to the reason, while the reason (the why) is checked to determine that the change itself is indeed worth making.
    - The specification of the change is provided to "change control" and "R&D".
    - The change is made in the R&D environment separated from other environments, and tested there for operational effectiveness, efficiency, security, performance, and whatever else is desired, and that the change actually achieves the purpose intended and no other purpose. Once done, the change (not the result of the change but the steps required to make the change – typically for software a "diff" file, and for other things, an engineering change order or similar) is provided to change control.
    - Change control cannot make changes on their own. All they can do to change the mechanisms planned for use is to use a standard automated process to apply the proposed change to the last "known good" version, and test it. But before that, they are tasked to:
        - Examine the changes in the context of the thing to be changed and the specification for the change. Each change to any part of the mechanisms must:
            - (1) be simple to understand and clearly and easily understood by the people doing change control; otherwise the change is rejected until properly explained, and the individuals who made the change subject to scrutiny for the lack of obviousness in their change.
            - (2) only change things that must be changed in order to achieve the objective of the change; otherwise the change will be summarily rejected and the person making the change removed from the development process.
            - (3) change things that do in fact achieve the objective of the change; otherwise the change will be rejected and quality control folks asked what they were doing (or failed to do) while this ineffective change was put in.
        - Implement and test the change (if it passes the previous tests) in the change control isolated environment to verify that it operates under a wide range of test

    conditions required to re-certify the system it operates as part of after such a change is made and before operation as can continue with the change in place.
  - Pass the changed version to operations for implementation during the next appropriate maintenance window.
- Operations then implements the change at the appropriate time using the change authorization and verification mechanism and the proper controls to allow for reversion in case something unanticipated goes wrong. For widespread software deployments, this uses the reasonably well developed mechanisms for distribution of software changes (and perhaps the SBOM as the delivery and instantiation mechanism).

Too slow and painful for the modern world?

Today, we don't even do the basics of sound change control for applications having global reach. The result is things like the recently discovered "SolarWinds" attack, which once embedded was then used to move from place to place extending to corrupt the mechanisms of yet untold mechanisms and systems, including the supply chains of the sourcing for more systems and mechanisms that still other things depend upon. In essence, to the transitive closure of information flow from the source, limited only be the extent to which the threat actors were able, willing, or interested in pushing it.

What will be the cost of cleanup? Will it be cleaned up very thoroughly? How long will we all live with the consequences of unsound change control? Al these are unknowns today, but what we do know clearly is that change control did not work, likely because it was not in place at the source or at the many destinations that simply accepted the change as sent and shoved it into the systems and mechanisms we depend on for everything from national and global security to making a call to wish others a happy birthday.

The vast extent to which this failure in the basics of reasonable and prudent operations and management in the cyber arena is stunning to some, but to those in the cyber security field, it indicates only that those in charge continue to ignore those who know and advise them.

Why B-BOM and SID?

If we are not going to go about implementing protection by being careful, or assuming reasonably that no natter how careful we are, someone somewhere will eventually be careless and the system won't have enough redundancy and disaggregation to keep the consequences below management defined thresholds, then we have the optional course of better and better detection, reaction, and adaptation. B-BOM and SID present another path to detection and response that don't have the same limitations of intrusion and anomaly detection, and apply to behavior as opposed to state. Behavior is how we tend to recognize attacks today, but we don't specify the envelope of designed or accepted behavior, and this limits our ability to automate.

Conclusions

Economies of scale and widely used applications with high levels of interdependency have very substantial consequences that may indeed justify far more effort than we put into them today. One approach we have not pushed far enough fast enough is the B-BOM and SID approach. But this requires significant research and development to operationalize, and today, we can't seem to even do the basics of sound change control.