The Good, The Bad, And The Ugly: Stepping on the Security Scale

Mary Ann Davidson
Oracle Corporation
Redwood Shores, California
mary.ann.davidson@oracle.com

Abstract: **Metrics are both fashionable and timely: many regulations that affect cybersecurity rely upon metrics – albeit, of the checklist variety in many cases – to ascertain compliance. However, there are far more effective uses of security metrics than external compliance exercises. The most effective use of security metrics is to manage better, which may include:**

- **Make a business case for needed change**
- **Focus scarce resource on most pressing problems (with the biggest payoff for resolution)**
- **Help spot problems early - or successes early**
- **Address "outside" concerns or criticisms fairly and objectively**

A successful security metric should:

- **Motivate good/correct behavior (not promote evasive tactics just to make the numbers look good)**
- **Prompt additional questions ("Why? How?") to understand what is influencing the numbers**
- **Answer basic questions of goodness (e.g., "Are we doing better or worse?")**
- **Be objective and measurable, even if correlation may not equal causality**

This paper explores the qualities of good security metrics and their application in security vulnerability handling as well as a software assurance program.

Keywords-security metrics, vulnerability handling, software assurance

I. WHY SECURITY METRICS?

There are many management fads that wax and wane in terms of the rate of adoption and the number of breathless presentations, seminars, and books on the topic. One of the current waves sweeping the information technology (IT) security world is metrics: the idea that security can be and should be measurable. In a way, this is not an unusual development. Businesses in general rely on measurement as a key management indicator. For example, what are audited financial statements but measurement of business activity, and what are ratios such as earnings per share (EPS), net present value (NPV) and internal rate of return (IRR) but key comparison metrics? Information technology is a business enabler and should be subject to management oversight (including reasonable and appropriate measurement) just as other lines of business are.

Add to that the fact that information is becoming (either directly or indirectly) more regulated through vehicles such as the Federal Information Security Management Act (FISMA) and Sarbanes-Oxley and often regulation requires one to attest to business practices to the satisfaction of independent auditors, which usually necessitates measurable proof points (metrics). The US Federal Government is also becoming security metrics-focused through such programs as the Federal Desktop Core Configuration (FDCC) and through sponsorship of such programs as the creation of security measurement schemes such as Common Weakness Enumeration (CWE) and Common Vulnerabilities and Exposure (CVE) under the tagline "making security measurable."

Lastly, the practice of information security naturally lends itself to measurement if for no other reason than the difficulties in determining "How much security is enough?" and "Am I doing the right things?" Anyone tasked with protecting an enterprise's networks must have a basic handle on such indicators as "What are my most critical systems?" and "What is the 'security-readiness state' – e.g., patch level and secure configuration status – of those systems?"

While faddishness is never a good reason to adopt a business practice, there are clearly many good reasons to implement metrics programs around security. One of the most often-cited bromides is "If you can't measure it, you can't manage it." (This isn't necessarily true, by the way, since many managers supervise the people who work for them without explicitly measuring their work. You can imagine the revolt if employee compensation were based on such metrics as number of emails sent and received, number of hours at work, volume of phone calls placed, and the like.)

It is true, however, that a good metrics program can help you manage *better*. In fact, the primary justification for implementing a metrics program in security or any other discipline – absent a regulatory requirement – is simply, to manage better. Then-Mayor Rudy Giuliani famously and dramatically improved the governability and livability of the city of New York through a vigorous metrics program (including, for example, measuring the number of guerrilla window washers accosting commuters).

"Manage better" may include the following:

- Answering basic questions about operational practice, such as "What are we doing and not doing?"
- Spotting trends – both positive and negative – in operational practice, such as "Are we doing better or worse since the last measurement period?"
- Serve to raise additional questions (e.g., "Why is X happening?")
- Comparing entities (e.g., "How is group X doing vs. group Y?"), so long as the comparisons are "apples to apples"

- Allocating resources well (e.g., "The volume of X has increased by Z, and we've historically needed Y resource to handle the increment")

Metrics can be either external or internal (e.g., as noted, audited financial statements and the financial analysis ratios derived from them are a form of external metrics). They need not be different; however, metrics used for external purposes may likely be focused on looking good externally while those used internally can more objectively focus on "managing better." Putting it differently, internal metrics provide more room for improvement if the numbers are not "good." That's the point, actually. If metrics do not ultimately help an organization be honest with itself, they might as well not exist.

II. Follow the Yellow Brick Road

Metrics do not grow on trees: thus, implementing a metrics program requires resourcing just as any other project does, with time, money, and people to support the desired outcome. To that effect – given that most organizations do not have unlimited time, money or people – it is important to consider the value of more information or better information as captured in a metrics program. That is, there may be a number of things you would like to measure. In a perfect world, you could measure how many angels dance on the head of a pin if that were of interest. However, unless you are already capturing the information you'd like to measure, it will cost you something to obtain the data or measurement.

The cost to obtain the metric (such as investment in changing processes or creating repositories to capture information) needs to be weighed against the value of more information. If capturing and analyzing data will not lead to your ability to make a better decision or manage better, you are better off not capturing the information and using the time and resource to obtain another metric that will lead to better decision-making. This – being explicitly cognizant of the value of more information – is the difference between academic discussions of process- and artifact-focused metrics, and what resource-constrained entities do to manage better.

A good starting place for a metrics program is simply date mining what you already have. For example, many organizations have processes and organizations associated with the IT function that track many aspects of their IT systems (such as help desk tickets, configuration management actions, and the like). Such data stores may already contain information relevant to security that can be incorporated into a metrics program. Adding to that baseline information by creating a metrics plan – a priority-ordered listing of "What else would we like to know?," "How can we obtain that information?" and "How will we use this data?" is key to creating an implementable metrics program.

For example, at Oracle, the security metrics program associated with software assurance focuses in part upon the security vulnerability handling function (there are also metrics associated with program management of assurance). The foundation of the metrics program was the Oracle bug database, since the development organization already had processes and data stores for logging, tracking, and resolving bugs of all kinds. Furthermore, security bugs were already flagged separately, because Oracle vulnerability handling processes mandate that security bugs are not published to customers and also that "need to know" is enforced on security bugs, as implemented by row-level data access in the bug database (so that only developers working on a security bug fix and their management chain can access details about a security bug). The reason security bugs are not published is that Oracle believes it puts customers at risk to publish security vulnerability details where there is no patch available.

The foundation of the security metrics wiki – the internally accessible metrics page to analyze security vulnerability trends – is the Oracle bug database (since to do otherwise would be to create an entirely separate and possibly less up-to-date data store). Also, the focus on security vulnerabilities is pragmatic: over time, as part of the Oracle software security assurance program, one expects the rate of serious security defects in software to go down. Also, the rate of security vulnerabilities goes directly to Oracle Corporation's security brand, and represents a cost to Oracle and to its customers. Specifically, Oracle has so many products, running on so many operating systems (e.g., the Oracle database runs on 19 operating systems) and so many versions in support that almost anything you can do to prevent security vulnerabilities - or find and fix them faster - represents significant cost avoidance.

Even in the short run, it is important to ensure that:

- security vulnerabilities are handled reasonably promptly (that is, the percentage of vulnerabilities by aging bucket is not skewing to older buckets)
- the backlog of unfixed security vulnerabilities is not growing (that is, worst case, there is a steady state of find-and-fix) or is shrinking
- the bulk of security vulnerabilities are found internally and not by customers or third party security researchers

Another reason to focus metrics on vulnerabilities is that development already has many metrics around development process and practice (including tracking open bugs in software). The primary goal of creating the security metrics wiki is to give development the tools it needs to manage their *own* workload. Development at Oracle "owns" its own code and in general does "take care of business." Highlighting security-specific bug trends of interest and exposing those to development is more likely to result in a positive outcome than producing metrics that are only used by the software assurance team (that does not own code or fix defects). Putting it differently, a self-policing model where developers have the tools to manage their own work is preferable to a "security police" model in which a small group uses metrics offensively, to punish development infractions.

III. Accentuate the Positive

A general difficulty in a metrics program is creating measurement that is not only useful but creates incentives for the correct behavior. “Correct behavior” can be demonstrated by a positive trend (e.g., fewer security defects in software) or can be as simple as a “hygiene metric,” such as how many people have completed a mandatory security training class and passed it. Note: the metric is “hygienic” in that it cannot necessarily be proved that there is a cause and effect between “do X and Y results,” but it is nonetheless considered a positive achievement towards a program objective. Putting it differently, it’s not clear how strongly correlated driver’s education training and a reduction in accidents involving teenage drivers are, but few people would turn a fifteen-year-old loose with the car keys without basic driver’s education.

A subtlety of “incentivizes the correct behavior” is making sure that the metrics are a fair measurement and not easily “gamed” by the participants (which may have unintended and often perverse side effects). That is, a poorly formed metric may create a circumstance in which those being measured change their behavior to optimize the metric instead of optimizing behavior whose natural outcome leads to an improved metric. For example, one Oracle development organization many years ago ran bug reports for senior management every Friday at noon (that reported the number of bugs still open in the product suite). Product managers would log into the bug database Friday morning and change the status of a number of bugs to “closed” or “not a bug,” wait until the reports were run, then reopen the bugs later to continue working on them. It was the easiest way to make the numbers look good at noon on Friday.

Using a “point in time” metric, instead of either analyzing the overall trend, or triangulating several data points, resulted in “gaming” behavior. In addition to the fact that the gaming activity made the metric unusable (the number of open bugs at noon Friday was largely a meaningless number), the activity consumed in changing bug status crowded out more useful work. This is what you don’t want if you use metrics as a management tool instead of a blame-assignment tool.

Triangulated metrics, in which several factors are used to arrive at a combined attribute, are more resistant to gaming as well as potentially providing better measurement (navigators, for example, require both latitude and longitude to “fix” their position in the ocean: neither latitude nor longitude is sufficient. Nor is a GPS system sufficient to fix your position if you drop it overboard).

For example, consider the area of producing patches for software vulnerabilities: most particularly, producing patches for security vulnerabilities. Most development organizations want fairly speedy closure of bugs, particularly for customer-reported bugs of high severity (that is, in which customer systems are down or their ability to function is impaired). Therefore, many development organizations keep track of their backlog of vulnerabilities, and “age” them. The utility of aging is similar to that of other business functions that do aging (accounts receivable, for example). It isn’t necessarily the total number of bugs that is a problem (after all, pre-release, you want developers to find bugs, log them, and fix them), but an aged backlog of bugs in a production product that is both growing and aging raises a red flag.

Developers may be measured on how fast they close bugs as well as by the absolute number of outstanding bugs. These metrics may be used for a number of goals, such as ensuring that no critical (high severity) bugs are outstanding before a product shipment milestone (e.g., major product release or a patch set release) and that the most critical bugs are addressed quickly. Both of these are worthy goals. However, in the case of security vulnerabilities, the picture is a little more complex.

Security vulnerabilities are different than non-security vulnerabilities in that speed of closure is not as important as ensuring that the root cause of the bug is addressed (or, “fixed completely”), and, moreover that the bug is fixed correctly *the first time*. One could argue that these are important attributes in any bug, but they are particularly critical in security bugs because of their nature. Specifically:

- Security bugs generally require the vendor to produce more patches than is the case for non-security bugs
- Security patches are generally more broadly applied by customers than non-security patches
- Security bugs that are not fixed correctly may (perversely) increase risk to customers

Regarding the first point, many vendors’ vulnerability handling policies include some notion of treating customers equally – more specifically, protecting customers to the same degree. As a result, the vendor produces security patches on all (or mostly all) supported systems so that customers are protected to the same degree, rather then telling customers to upgrade to the newest version (which often takes months and not days to do). The record at Oracle for a single-issue security patch (that is, a patch that fixed a single security issue on all affected versions and platforms) is 78 patches. The cost to produce those patches easily exceeded $1,000,000 and that was exclusive of costs to Oracle Corporation to patch their own product instances and exclusive of costs to customers of patching *their* product instances.

Regarding the second point, customers generally apply security patches more broadly then other (non-critical) patches for, say, performance issues. That is, customers don’t wait until they encounter a security problem (i.e., by experiencing a data breach) before applying the patch if they believe their software installations are affected by an issue and there is no other mitigation. They may also be required by regulatory pressures to apply security patches more broadly than is the case for non-security patches.

Regarding the third point, either a reporter of a security bug or other third parties may find other instances of a vulnerability if it is not fixed correctly. For example, suppose a third party security researcher reports that a particular sequence of characters (Ctrl-X) inputted to an application causes a core dump (and therefore, anyone sending that sequence of characters to an application could cause a denial of service attack). The real security problem is not that Ctrl-X causes a core dump; it's that a failure to validate input correctly leads to a denial of service attack. Should a developer only modify the code to handle Ctrl-X gracefully, the likelihood is that the reporter will log additional bugs in succeeding weeks regarding Ctrl-Y, Ctrl-Z, and so forth. Even if the reporter does not try additional combinations to force a core dump, should the vendor release a security patch, it may well be decompiled (by other researchers) to figure out "What changed" to determine "How can we exploit the vulnerability?" The original coding error is now compounded – and the risk to customers increased – because:

- The fix was not complete or correct
- A customer may have applied a costly emergency patch, yet is still vulnerable to a variant of the weakness
- And may either experience an exploit or have to apply another expensive emergency patch

In short, if you only measure developers on how fast they close a security bug, their incentive is to fix the exact issue reported and not look at the root cause. You get the behavior you measure for, but not what customers actually need.

As a result of the above three factors that make security vulnerabilities different from non-security vulnerabilities, it is critical to capture metrics that measure the desired outcome and incentivize the correct behavior. A more robust metric than "closing the bug quickly" is to triangulate what is being measured to factor in "completeness of fix" and specifically, capture whether a bug is ever reopened (i.e., because the developer made a mistake and the security bug was not fixed correctly the first time). Incentivizing the correct behavior can be facilitated by using normal bug correction mechanisms to remind developers that "security bugs are different."

Oracle uses the bug database to both "flag" security bugs specifically and add notations to the bug text as to the requirements for "fix completely" (a URL to the secure coding standards and "case law" on what "fix completely" means in the context of security bugs). The purpose of metrics is to manage better; therefore, reminding developers of their responsibilities vis-à-vis fixing security bugs (and that correctness is as much or more important than a fast fix) is considered more effective than merely tracking how poorly the organization is doing regarding meeting that requirement.

Another triangulation example involves the development equivalent of solving the worst problems first. Assigning a severity to security bugs using a standard measure, such as Common Vulnerability Scoring System (CVSS), allows "bucketing" of bugs by severity. Capturing aging information (e.g., aged backlog – which bugs are 0-30 days old, 30-60, 60-90 days old, and so forth) can be refined by using aging buckets to help measure not merely the status of the backlog, but whether more critical issues are being addressed first. Most development organizations have to use some kind of scoring or vulnerability rating anyway since no organization can reasonably fix all security bugs of all severity on all old product versions. Also, it is important to customers (as well as efficient resource allocation) to ensure that at least the most severe issues (that are likely to become public or that are already public) are fixed for affected and supported platforms.

IV. Mirror Mirror On The Wall, Which Product is the Least Buggiest of All?

Metrics gaming becomes an issue when metrics moves from a management function to a reporting function, or from private measurement to public disclosure. The temptation to "game" metrics is never greater than in instances where numbers will be used for comparative purposes or competitive purposes, most especially when they are public.

Many vendors routinely take pot shots at one another over a metric that can broadly be categorized as "number of published security vulnerabilities in competing products." Absent much transparency or disclosure in how products are developed or "how buggy the code base actually is," "published vulnerabilities" becomes the security metric vendors use against one another (and that customers rely upon as a pseudo-security quality metric). The metric can be compiled in as simple a fashion as counting the number of line items in security advisories that a competitor releases during a year in a competing product. The metric as used is inherently flawed for reasons that will become clear below.

First of all, a metric "comparison," to be relevant, needs to compare apples to apples, not apples to road apples. There are few equivalent ways to measure security defects in software, and the ones that are equivalent sometimes fail because of lack of transparency and disclosure (as noted earlier, public metrics are likely to not only measure different things but potentially, actually report different (i.e., "prettied up") numbers since they are not audited or otherwise independently verified).

An obvious comparative metric for multiple bodies of code is "defects per thousand lines of code (KLOC)." Comparing defects per KLOC in two bodies of code, assuming the sample is large enough, is a fair comparison. However, defects per KLOC is not a comparison of "security" per se; rather, it is a quality metric that may nonetheless have security implications. That is, some percentage of the defects per KLOC may be security vulnerabilities, but absent knowing how many are, it's not a security comparison at all, merely a quality metric.

Another aspect of comparing security vulnerabilities is that not all security vulnerabilities are of comparable severity. Clearly severity is important when talking about security vulnerabilities, since some defects are far more relevant than others. A low severity, or non-exploitable, or not-easily-exploitable security defect is not in the same category as one in which anybody can (remotely) become SYSADMIN, for example. Note: it is true that lower severity security vulnerabilities may be exploited in a so-called combined attack, but it is difficult to anticipate these and, absent an combined attack exploit in the wild, vendors are better off using an objective (stand-alone) severity ranking for resource allocation purposes.

In fact, security vulnerabilities are inherently non-comparable unless they are "rated" according to a severity ranking such as the CVSS. While there is inevitably some judgment involved in assigning CVSS scores, they nonetheless enable some "apples to apples" comparisons between comparable products to the extent all product vulnerabilities are assigned a CVSS score by vendors.

Non-comparability of product factoring or product size is also a consideration. For example, comparing numbers of security vulnerabilities in competing products is not always useful to the extent that the products are either significantly different as regards features and functions, the size of code base, or in the way the product is factored (that is, extra-cost or additional product functionality may be bundled in one product where it is factored as a separate product by another vendor). It is, in a way, like comparing termite inspection reports from a 20,000 square foot house and a doghouse. You might know how many absolute numbers of bugs there are in each structure, but it doesn't tell you whether the McMansion can truly be considered "buggier" than the doghouse, or vice versa.

Another difficulty with comparing number of published security vulnerabilities in products is that there is no standard (or objective) way of actually counting security vulnerabilities. If a change to one function addresses the root cause of several reported security bugs (involving, say, failure to validate input correctly), is that to be counted as one security bug or several security bugs? A common criticism of many vendors is that they do not, upon addressing externally reporting vulnerabilities, address "nearby" vulnerabilities. For example, a reported vulnerability in an Apache mod should lead one to analyze other Apache mods for similar problems. Should a vendor who does so report "related" vulnerabilities in the same mod as part of a security advisory, the disclosure of what else was found (if the vendor discloses it) is likely to be held against the vendor. Ironically, "correct" behavior ("look for other issues and fix them") is likely to rebound more negatively on the vendor (at least in the short run) than would be the case if the vendor either doesn't look for related issues, or looks for and fixes additional issues without disclosing what else was found and fixed. (One can argue about "silent fixing" vs. full disclosure, but the bottom line is that finding and fixing more issues earlier is generally in customers' best interests.)

To that point, using published numbers of security vulnerabilities as a competitive comparison is inherently flawed as it relies to a large extent on self-disclosure by the vendor and thus is "rigged to be gamed." Absent knowing what a vendor's vulnerability disclosure policies are (for example, do they self-disclose issues they find and fix themselves?), it is not possible to make any meaningful comparison based on "number of publicly disclosed vulnerabilities." One can assume that vendors find some portion of security vulnerabilities themselves: do they disclose all of these? Likely not. Oracle's own metrics substantiate that (taking all product families together) more than 90% of security vulnerabilities are found internally (a percentage that has increased since we began keeping metrics).

Furthermore, as more vendors move to a bundling strategy as regards security bugs (that is, security patches are released in regular patch bundles either monthly or quarterly or on another published schedule), does it really matter how many vulnerabilities are contained in the bundle (and isn't that the point, really, to address a number of critical security issues in one patch rather than piecemeal?)

Granted, knowing how many security issues are fixed in a patch may be cause for customer concern. Of course, absent knowing how many issues are fixed-but-not-public in a bundle or whether the trend of "new vulnerabilities introduced in code" vs. "old vulnerabilities found and fixed" is positive (that is, that newer versions are more secure), it's still not necessarily possible to draw security-worthiness inferences from *published* security vulnerability information. It's like drawing inferences about the size of an iceberg from looking at the amount above water (except that in the case of icebergs, we at least have specific gravity information to work with whereas with vulnerabilities, there is no way to estimate what's below the water line).

The nail in the coffin in using "number of published security vulnerabilities" as a quality metric or for security comparison purposes is that it is not only gameable, but the metric itself invites gaming. In other words, if a vendor thinks that numbers of published vulnerabilities will be used against it competitively, the incentive is to not self-report significant vulnerabilities so as to make the numbers look better. Note: while no vendor publicly reports every issue they fix for a multiplicity of reasons (e.g., inability to fix all issue on all old versions argues for some discretion in reporting to the extent those versions are at a risk that can't be mitigated); on balance, creating positive incentives to protect customers (instead of reasons to "game") is in industry's interests.

In a competitive market, businesses will use what competitive advantage they can to gain leverage over a competitor. Security-worthiness, to the extent that it can be verifiably a differentiator, can be a competitive advantage. However, number of publicly disclosed security vulnerabilities is not objectively a fairly competitive metric.

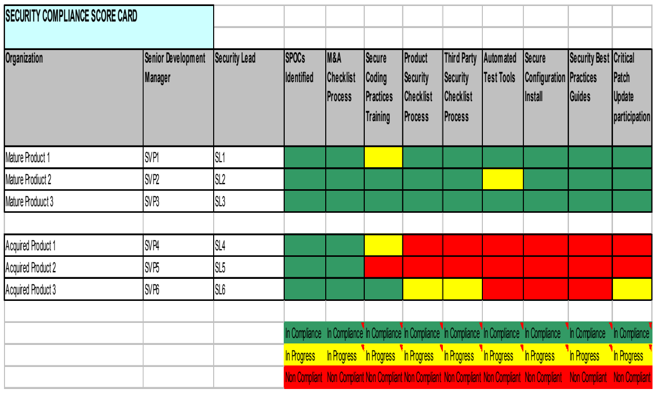
V. Mine's Bigger Than Yours

The corollary to the – potentially destructive – aspects of using externalized (public) security metrics competitively is the benefit of harnessing competitive metrics internal to an organization. Specifically, nobody likes to be seen as the person or group doing the worst at something. Peer pressure can be a force for good in a security program.

For example, Oracle has made a number of significant acquisitions in the past several years. As part of post-acquisition integration, we align development organizations from acquired companies with Oracle secure development practices. One of the difficulties of this process is that different acquired entities are at different points in the continuum of secure development practice, in regards to both "where they started" and "how they are aligning with the corporate secure development practices" (the umbrella term for which is Oracle Software Security Assurance (OSSA)). Organizations that, post-acquisition, find themselves under the umbrella of a mature development organization tend to “inherit” a number of development practices as they are integrated into the development group. On the other hand, some acquired entities – mostly in vertical or industry-specific segments – are incorporated as largely standalone entities into organizations that are not primarily development organizations. There is thus, in these organizations, no long-established development practice to inherit.

For reasons including the disparity of “inherited development practice” as well as the sheer number of acquisitions, Oracle has instituted a more structured governance program around software assurance that is geared to looking at consistency of secure development practice across the entire organization. The primary purpose of the collected metrics – in this case, measurement of which elements of OSSA have been adopted by each organization and to what degree – is simply to measure “How are we doing in the uptake of OSSA software assurance programs?” The security compliance scorecard is reported to the Oracle Security Oversight Committee (OSOC) as well as the Chief Executive Officer (CEO).

The secondary purpose of metrics around OSSA adoption is to be able to track organizations that are slow in adoption or appeared to be running into difficulties in adopting standard Oracle secure development practices. Groups that are slow, or running into difficulties, are targets for concentrated assistance. Specifically, rather than using Oracle Global Product Security (GPS) (the group responsible for monitoring OSSA compliance) resources thinly across the entire company, the resources are applied in a more concentrated form to the teams that are “slow learners” or “late adopters.” This may include a product assessment conducted by the ethical hacking team, accelerated uptake of automated tools (e.g., static analysis) and other fast-tracked OSSA requirements.



SECURITY COMPLIANCE SCORE CARD

Organization	Senior Development Manager	Security Lead	SPOCs Identified	M&A Checklist Process	Secure Coding Practices Training	Product Security Checklist Process	Third Party Security Checklist Process	Automated Test Tools	Secure Configuration Install	Security Best Practices Guides	Critical Patch Update participation
Mature Product 1	SVP1	SL1	🟩	🟩	🟨	🟩	🟩	🟩	🟩	🟩	🟩
Mature Prodiuct 2	SVP2	SL2	🟩	🟩	🟩	🟩	🟩	🟨	🟩	🟩	🟩
Mature Produuct 3	SVP3	SL3	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Acquired Product 1	SVP4	SL4	🟩	🟩	🟨	🟥	🟥	🟥	🟥	🟥	🟥
Acquired Product 2	SVP5	SL5	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥
Acquired Product 3	SVP6	SL6	🟩	🟩	🟩	🟨	🟨	🟥	🟥	🟥	🟨
			In Compliance	In Compliance	In Compliance	In Compliance	In Compliance	In Compliance	In Compliance	In Compliance	In Compliance
			In Progress	In Progress	In Progress	In Progress	In Progress	In Progress	In Progress	In Progress	In Progress
			Non Compliant	Non Compliant	Non Compliant	Non Compliant	Non Compliant	Non Compliant	Non Compliant	Non Compliant	Non Compliant

Figure 1. Oracle Security Compliance Scorecard

The third reason for a governance structure is in anticipation of greater customer interest in secure development practices and a requirement for transparency – perhaps driven by regulation. Anecdotally, more customer requests for proposals (RFPs) are requesting information on secure development practice, and being able to track, by development group, what is being done and not done enables more transparency and more accurate RFP responses. In at least one case, an increased customer concern in the security of a particular product led to a “full court press” in increasing the uptake of OSSA activities. This would not have been possible without the baseline reporting that the security compliance scorecard (metrics) enables.

Without making compliance an impossible goal to meet, the intention of the compliance scorecard is that what constitutes “compliance” will change over time. Specifically, as the state of the art improves in software assurance, Oracle will redefine what “compliance” means for a number of cells in the chart. For example, one compliance area is the use of automated tools, which is one of the best and cheapest ways to get avoidable, preventable defects out of one’s code base. “Use of tools” as a compliance metric encompasses everything from identification of the appropriate and applicable tools by development group (e.g., static analysis, web vulnerability analysis, fuzzers, etc.) to deployment and degree of code coverage. Over time, we may identify (or develop) additional tools that would also be adopted, which means “compliance” will include use of new tools in the groups to which they are applicable.

An expected benefit of the security compliance scorecard is the – for lack of a better expression – shaming effect it has on groups that have had OSSA adoption issues. Specifically, one of the baseline elements of OSSA is security training – more specifically, completion of an online class on Oracle Secure Coding Standards (OSCS), that is mandatory for most of development (e.g., developers, quality assurance (QA), release management, product management, and so on).

Despite the OSCS class being online and modular (which facilitates people being able to take the class at their own pace, completing portions of it a bit at a time), adoption had been slow in some groups. The single factor in rapidly increasing class adoption was creation of a bar chart showing, for each organization, their percentage completion. Since the bar chart showed all organizations’ training status as compared with each other, it also highlighted who the “worst offenders” were. Reporting numbers by group as a percentage had never “grabbed” management attention: the bar chart showed not only who was compliant, but how far out of compliance some groups were relative to others.

Publishing the bar chart that showed who the outliers were had a dramatic and immediate effect upon compliance rates. Specifically, once the word got out that the charts had gone to executive management and the Oracle Security Oversight Committee – and that the report was being rerun and the results re-reported in a month’s time – the compliance rate went straight up. “Compliance” here meant not merely that someone had attempted the class, but more specifically, “of the people in a development group required to take the class, who has taken it and passed it?” Ninety five percent or better is considered “compliant” for tracking purposes.

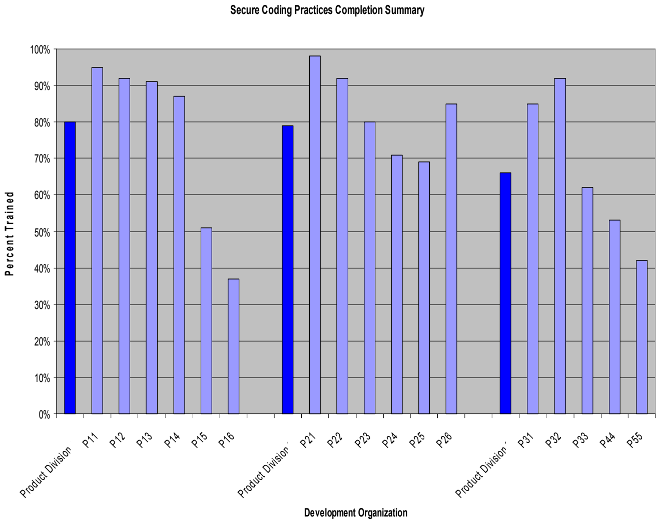


Figure 2. OSCS Training Completion Status – Pre-OSOC Meeting

To reiterate that this was not intended to be a "gotcha" exercise, Oracle Global Product Security:

- Published the then-current scorecard about two months in advance of the semi-annual Oracle security oversight committee meeting at which the numbers would be reported
- With a notation that the final results would be compiled and reported in two months' time
- And a reminder that the class was mandatory for most of development (individual managers were reminded of the specific names of those who worked for them who had to take the class)
- And a request to comply with the training mandate as soon as feasible

In other words, this was geared as a reminder – with a long lead-time – of the requirement to take the class and a notation that the results would be reported to executive management. One could argue that it should not require "reporting to the boss" to ensure compliance but the fact remains that compliance rates dramatically increased when the results clearly showed the outliers. The results also showed groups that had made a dramatic improvement, and the written analysis also credited the security leads for those groups who had used the opportunity to exhort their teams to compliance. In other words, the report taken in toto was geared to show "who has done very well" as well as "who is falling behind."

Figures 2 and 3 show, respectively, compliance rates before the numbers were reported to the Oracle Security Oversight Committee, and compliance rates four months later (in reality, the most dramatic compliance increase occurred within the first two months).

When all else fails, public "naming and shaming" works, and numbers don't lie.

It's not clear that there is a direct correlation between expanded secure coding training and security bug reduction, in particular because automated tools have also become more broadly deployed and thus more security bugs are found earlier. That said, the foundation for OSSA is the Oracle Secure Coding Standards. The training class is intended to be familiarization for developers that they do indeed have responsibilities and standards of practice, and pointers to sources of more information. Getting developers to "get it" is the main purpose of the class, and other processes, tools, checks and balances and compliance constitute the framework to make it easier for developers to write secure code than not.

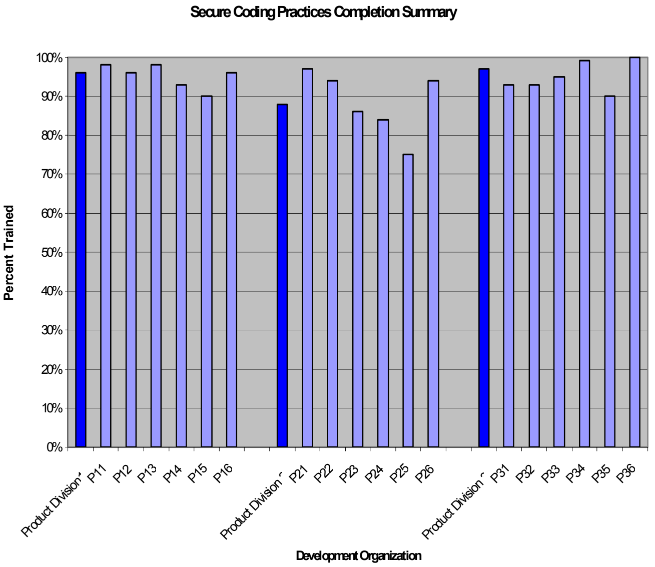


Figure 3. OSCS Training Completion Status – Four Months Post-OSOC Meeting

VI. Tastes Great *and* Less Filling, Too?

One of the benefits of a thriving metrics program is that in many cases information you collect for one purpose may have a secondary use. Of course, this is what privacy experts worry about vis-à-vis personally identifiable information (PII). Metrics you develop for your own use do not have the same concerns: in fact, one hopes that a metric developed for a good business reason proves to have secondary benefits.

That is not the primary reason to collect data – "I might need that some day" – but a secondary use can help validate the metrics function and in some cases can provide business justification "with teeth" that would not otherwise be available. For example, one of the challenges for Oracle related to acquisitions is – as noted in the earlier discussions of OSSA – getting acquired entities to adopt standard secure development practices, including Oracle vulnerability handling processes. In many cases, an acquired entity may *not* have a pre-existing security vulnerability handling team or processes; also, an acquired entity may become a target of the research community *after* the product becomes an Oracle branded product, post-acquisition.

Therefore, the challenge is both the “how” to integrate the entity with Oracle vulnerability handling processes and determining the correct number of “whos” to do the work. Even instituting a lot of automation around security vulnerability handling processes, as Oracle has done, does not mean that no additional bodies are needed to handle an increased workload.

Fortunately, the area of vulnerability handling is one in which Oracle’s security metrics include:

- Number of open security bugs being tracked, aged, and growth rate
- Number of security bulletins or alerts issued per year
- Volume of email correspondence to the security vulnerability handling team (i.e., from external reporters who are not customers)

Managers made the business case for the number of people needed to resource the security vulnerability handling function (at the yearly budgeting cycle) based upon what was known about the “security bug load” of recently acquired entities. (Similar analysis was used to justify additional headcount for the oversight function.) More specifically, looking at workload items such as volumes of emails to the vulnerability group, number of security bulletins/advisories issues, backlogs of security vulnerabilities that required tracking and resolution was used as raw data to justify headcount increases.

At Oracle, the tracking system for security vulnerabilities tracks not only the reporter but also “buckets” the reporters for metrics purposes. Specifically, we differentiate among:

- Customer reported bugs
- Security researcher- reported bugs
- Internally found bugs (general)
- Internally found bugs (found by the ethical hacking team)

One of the main reasons to differentiate among the above groups is that – all things being equal – we believe it is more important to speedily resolve researcher-reported bugs than comparably severe bugs reported by other sources. The reasons include the concern that researchers may become impatient if their bugs are not speedily addressed (and thus they may “out” the bug before there is a fix available, thus increasing risk to customers). Also, researchers whose bugs are not fixed promptly may complain publicly, which – in addition to generating bad PR – generates work to respond to the bad PR and thereby address customer concerns. (Note: one of the best ways to avoid negative PR generated by security researchers is to give them nothing that is both negative and accurate to say. To the extent bad PR is based on incorrect information, it can be refuted; to the extent it is accurate, the root cause needs to be addressed by the vendor.)

At Oracle, the purpose of having an ethical hacking team focused on product assessments (rather than, for example, doing penetration tests of production systems) is twofold. The first is to find vulnerabilities in our own product suites before third party researchers or actual hackers do; the second (though, in a way, the first) is to effect knowledge transfer to development (in the form of in-house “hacking tools,” secure coding standards, “case law” on hacker-resistant development practices and the like). It is a general goal of our vulnerability handling processes to address internal, ethical-hacking team found vulnerabilities quickly since the entire goal of turning the hackers loose on a product is to beat third party researchers or actual hackers to the punch. (There’s nothing worse than having a third party researcher report the exact same bug that was found internally, has not been fixed and is still languishing in development.)

There was a concern that ethical hacking bugs were not being fixed quickly enough. The ramifications of that would include a higher cost to the company (because, to the extent the fixes for those vulnerabilities are ultimately released on multiple platforms and versions, the longer they languish, the greater the cost to remediate as we miss a version going out the door which could contain the fix and may thus need a separate patch later).

An analysis of the metrics of how many issues had been reported to-date in each “reporter” category (shown in Figure 4) plus the percentage of issues that were still open showed that “gut feeling” was not accurate. In fact, next to researcher reported bugs, “ethical hacking bugs” had the highest degree of bug closure (see below chart). (Note: “external, no credit” category captures third party researchers who do not care about receiving credit in security advisories.) A quick review of “the facts” helped alleviate the concern over what was, in fact, anecdotal but not accurate. More importantly, it helped Oracle avoid an expensive fire drill to solve a non-existing problem.

Reporter	**Percentage of Reported Bugs That Are Open**
Customer	9.9
External, No Credit	3.3
Researchers	3.6
Internal	11.7
Internal (ethical hacking team)	5.2

Figure 4. Percentage of Open Bugs by Reporter

VII. SUMMARY

Metrics are a useful tool in many areas of management, including security and more particularly, in the area of security vulnerability handling and security assurance. Oracle’s security metrics program has moved from a gleam in a manager’s eye to an active program used to change the way we allocate security resources. Giving access to security metrics to developers themselves has helped highlight successes (as well as “needs work” areas) to help development organizations better manage their own workload. Providing assurance compliance metrics to senior management has helped increase the visibility of the assurance function within Oracle and harnessed peer pressure in pursuit of more secure code.

ACKNOWLEDGMENT

Thanks to Darius Wiles of Oracle Global Product Security for development of the Oracle security metrics wiki and to John Heimann of Oracle Global Product Security for the Oracle Security Compliance Scorecard.

REFERENCES

[1] Rudolph W. Giuliani, Leadership, Miramax, 2005.