Approved for Public Release: 11-5334.

The Software Industry's "Clean Water Act" Alternative

Robert A. Martin
MITRE Corporation
Bedford, MA

Steven M. Christey
MITRE Corporation
Bedford, MA

ABSTRACT

With water we have trust that qualities harmful to its intended use are not present. In order to avoid a regulatory "solution" to problems with "contaminants" that endanger software's intended use, the industry needs to put in place processes and technical methods for examining software for the contaminants that are most dangerous given the intended use of specific software.

The Common Weakness Enumeration (CWE™) [1] offers the industry a list of potentially dangerous contaminants to software. Common Weakness Scoring System (CWSS™)[2] and Common Weakness Risk Analysis Framework (CWRAF™)[3] provide a standard method for identifying which of these dangerous contaminants would be most harmful to a particular organization, given the intended use of a specific piece of software within that organization.

By finding systematic and verifiable ways of identifying, removing, and gaining assurance that contaminated software has been addressed, software providers can improve customers' confidence in systems and possibly avoid regulatory solutions.

INTRODUCTION

In the late 1990s the sharing, discussing, measuring, and reporting activities surrounding publicly known vulnerabilities in software products were very unorganized and cumbersome to manage. MITRE, with the help and engagement of industry, academia, and government, changed that through the introduction and worldwide industry adoption of the Common Vulnerabilities and Exposures (CVE®)[4] effort so that today everyone can easily correlate and coordinate their activities and information with respect to publicly known vulnerabilities in software products. Unfortunately, the same problems that lead to these publicly known vulnerabilities occur in everyone's software applications and are hard to detect. This is especially problematic if you aren't aware of their possible impact to your business, given what your software is doing for your business.

The Common Weakness Enumeration, or CWE, is a community effort to catalog and define all of those things in software that can lead to an exploitable vulnerability. CWE currently contains over 870 weaknesses (contaminants) that could be in the software's architecture, design, code, or deployment. Examples include buffer overflows, format strings, etc.; structure and validity problems; common special element manipulations; channel and path errors; handler errors; user interface errors; pathname traversal and equivalence errors; authentication errors; resource management errors; insufficient verification of data; code evaluation and injection; and randomness and predictability. In some situations, such weaknesses can make the software vulnerable to exploitation, causing a risk to your business. Many of these

© 2012 IEEE. All Rights Reserved.

weaknesses are the target of static analysis tools and represent standard identifiers for many of the “issues” that these tools report as findings.

Much like the water we use in widely diverse sets of daily activities across all aspects of our world’s ecosystem, the actual sources and manner in which most of us receive the software we use in every aspect of our cyber ecosystem is unknown and possibly unknowable. With water we have, over time, developed measurements and methods that give us trust in the fact that the qualities that are harmful to water and our intended use of it are not present. This is due to both our collective technical ability to specify and measure for those quality characteristics of water that are harmful, like temperature, hardness, volume, presence of pollutants, sediments, and other contaminants (depending on what use the water is intended for), as well as a regulatory framework that mandates that these quality characteristics of water be checked for when they could be dangerous to the intended use of the water. With knowledge of the dangerous qualities for a given use of water there are mitigation and controls that can be applied and verified, such as water softeners, filtration, settling ponds, cooling towers, etc., depending on the quality issues of concern for a particular use.

In order to avoid a regulatory “solution” to the contaminated software problem facing consumers of software, the industry needs to put in place its own processes and technical methods for examining software for contaminants and pollutants that are harmful or dangerous given the intended use that software is being asked to perform, and ensuring that appropriate mitigations and controls are in place to remove the harmful characteristics. The static analysis field is the most direct approach to measuring for many of these harmful items.

The CWE effort offers us, as an industry, the list of potentially harmful contaminants and pollutants. Identifying which items are most dangerous given the intended use of a specific piece of software is the focus of the risk model underlying the Common Weakness Scoring System, or CWSS, and the Common Weakness Risk Analysis Framework, or CWRAF.

In this paper we will define an approach for organizations and development teams to document the security-relevant capabilities of a piece of software in a manner that can be used to rank the various impacts that could result from exploitation of any particular CWE that ended up in the fielded software. Thus, the weaknesses that would be most dangerous to the software’s intended use by an organization can be identified, and the elimination or mitigation of those weaknesses can be prioritized over the others. By addressing contaminated and dangerous software now, and finding systematic and verifiable ways of removing these issues, software providers can improve their customers’ assurance in their systems and possibly avoid a regulatory solution to this issue.

BACKGROUND

Over the last two decades software has emerged from a novelty item used in specialized scientific, academic, and business practices to underlying almost every aspect of modern commerce, government, and recreation activity. Software is in our homes, vehicles, communications, and entertainment. Unfortunately, unlike the rest of our world, software, being man-made, doesn’t follow natural laws (as water does) and can be spectacularly unpredictable, especially given the variety of training and skills used to create and update it.

© 2012 IEEE. All Rights Reserved.

Starting with the vagueness and looseness of many of the computer languages that are used to write software, to the variety of compilers and other tools used to build and deploy our software, and the wide diversity and patchwork of skills and training of those that manage, specify, create, test, and field this software, it is actually amazing that most of it works for most of us. However, that same set of software is also being used and abused by others who are making that same software do things that were never intended by its creators. While traditionally this has led to calls for more security functionality and more rigorous creation/testing and management of software, that approach will take a long time and may not help you and your business in time to avoid your own disaster from software exploitation.

We believe that correcting all of this vulnerable software is necessary, and the creation of CVE and CWE are meant to help manage and facilitate those efforts. But we also believe that organizations need to be able to focus their efforts so that the exploits that are truly dangerous to their organizations are addressed first and foremost, and that was our motivation for creating the annual "CWE/SANS Top 25 Most Dangerous Software Errors List"[5]. But those prioritization lists, while a useful approach for sorting through and targeting the subset of CWE that is most harmful, do not reflect the specific uses a particular organization has for its software nor the types of failures that would be most detrimental to the organization.

Used together, CWE's Common Weakness Scoring System, or CWSS, and Common Weakness Risk Analysis Framework, or CWRAF, allow organizations to rank classes of weaknesses independent of any particular software package or project, in order to prioritize them relative to each other (e.g., "buffer overflows are higher priority than memory leaks").

CWSS provides the mechanism for scoring weaknesses in a particular piece of software in a consistent, flexible, open manner while incorporating knowledge of the context of the software's use in the particular business and reflecting the impacts of weaknesses against that context. CWSS, which is the part of the CWE project aimed at providing a consistent approach for tools and services prioritizing their static and dynamic analysis findings, is being developed as a collaborative, community-based effort so that it addresses the needs of stakeholders across government, academia, and industry. .

CWRAF uses the core scoring mechanisms from CWSS to provide a means for software developers and consumers to prioritize their own target list of software weaknesses across their unique portfolio of software applications and projects. The central focus is to do this prioritization so that the weaknesses that represent the greatest potential for harm to their business, mission, and deployed technologies can instead be used to reduce that risk. For example, to help select appropriate tools and techniques, focus training of staff, and define contracting details to make sure an outsourced effort can also address those issues prioritized as being the most dangerous.

Using a prioritized list, sometimes referred to as a "Top-N list," is a common approach used by the CWE/SANS Top 25 Most Dangerous Software Errors, the OWASP Top Ten, and similar efforts to help the community focus their risk reduction activities. CWRAF and CWSS allow users to create their own custom Top-N lists of the weaknesses that are the most critical for the particular software that is used in their own relevant business domains, missions, and technology groups. In conjunction with other activities, CWSS and CWRAF ultimately help developers and consumers to introduce more robust and resilient software into their operational environments.

© 2012 IEEE. All Rights Reserved.

VULNERABLE SOFTWARE IS EVERYWHERE YOU LOOK, SO…

While some lament the sorry state of developer knowledge and skill at producing invulnerable software, the truth is that most of the issues that can go "wrong" and lead to exploitable vulnerabilities haven't been well known or understood. For context we can look at our analogy of software as water. Many of the quality issues related to water's safe use have been clear to all. For instance, as shown in the top of Figure 1, having too much water in a river makes its use for recreation dangerous, as does too little, or having obstacles near the surface of the water. Likewise, as shown in the bottom of Figure 1, but less obvious to the naked eye, is the hazard of hard water, lead water pipes, or ground water contamination from industrial waste when water is used for consumption, cleaning, or other similar processes.

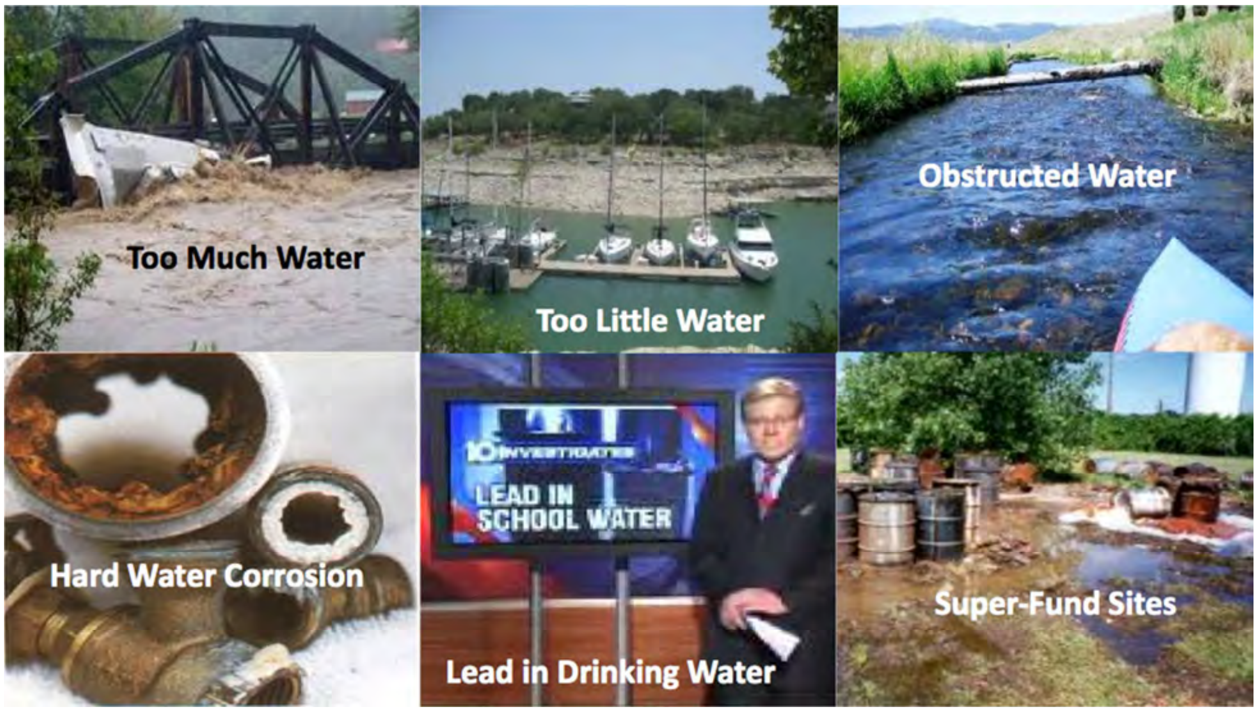


Figure 1: Issues with Water

Like many of the hazards in our software, there are a lot of water problems in the world and remediating these hazards, whether by replacing and renovating plumbing systems or adding active controls and protections to control the dangerous issues, addressing them is still an ongoing challenge. For instance, it was a long time before we collectively understood that hard water leads to residue build-up and corrosion in pipes. Similarly, lead pipes used to be the standard for plumbing until the harm from lead was identified and the sources of it were proscribed and targeted for removal or mitigation[6]. Likewise, the impact to ground water aquifers from polluted industrial sites has led to the Environmental Protection Agency's huge super-fund clean-up efforts[7] that are already decades long with no end in sight for both the ongoing efforts as well as new efforts to address known problems that haven't even begun yet.

But in our ecosystem, recreational use and human consumption are only two of the "uses" we put to water. We use it for transportation, creation of power, irrigation, and cooling , and in many

© 2012 IEEE. All Rights Reserved.

different aspects of manufacturing; we fish and harvest from it; and we use it as an ingredient in food, medicine, and beverages.

As shown in Figure 2, water is used in a great number of ways by us, wildlife, and the earth itself. For each of those uses there are qualities of the water that must be understood and checked to make sure that the water isn't dangerous for the ways it will be used.

Similarly, we use software for almost every type of activity in our cyber ecosystem, but in very few instances have we actually studied and understood what the qualities of the software that must be understood are, and then made it the norm to ensure that the software isn't dangerous given the ways it will be used.

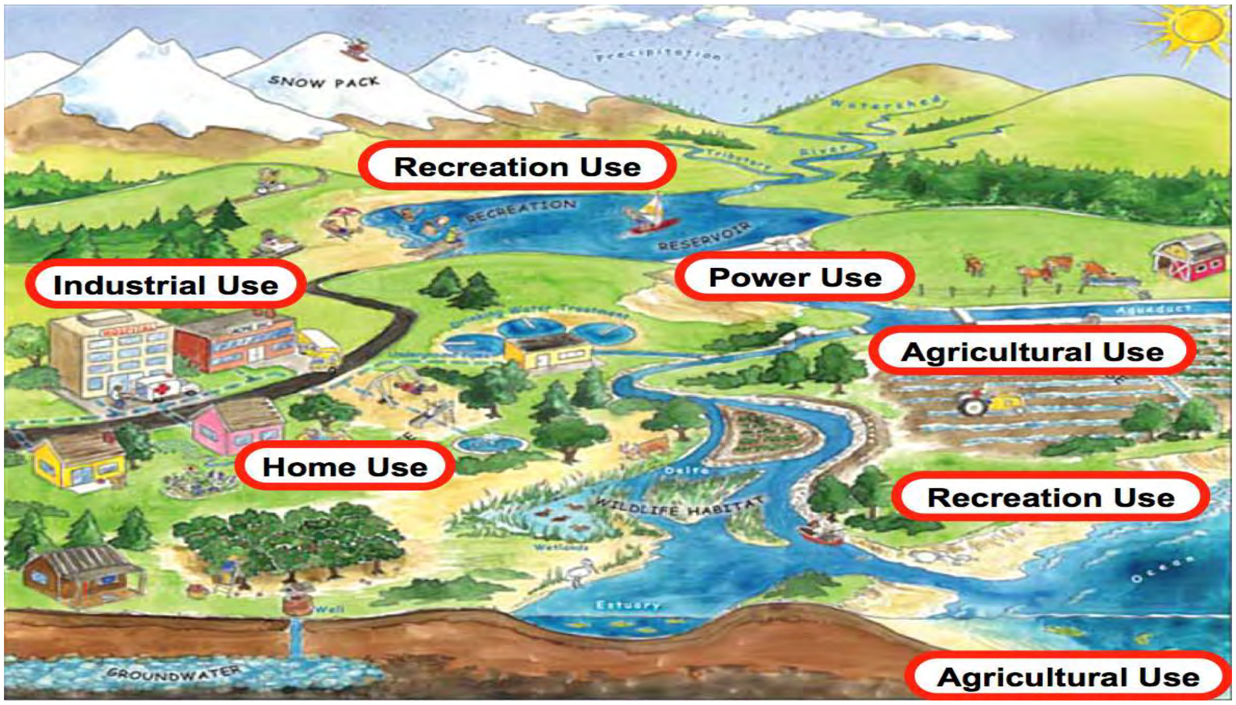


Figure 2: Water Uses in Our Ecosystem

Even in areas where we have been using and managing water use for our entire history, like in agriculture, we still end up with problems like e-coli contaminated water being used to irrigate spinach that ends up killing people around the world. Similarly, in our high-technology world, we are still susceptible to stagnant water in cooling towers leading to Legionnaires' outbreaks even today. So while the use of water has had a long time to mature and grow as a "safe and secure" discipline, clearly we still have areas for improvement. Our use of software is infantile in comparison and has many areas for maturity and growth, one of which is the recognition of what makes software dangerous for its intended use, and the cataloging and prioritizing of those contaminants that can make software exploitable so we can target them, prevent or remove them, and have assurance that they have been addressed appropriately.

WHAT ARE THE DANGERS TO OUR SOFTWARE?

© 2012 IEEE. All Rights Reserved.

During the past decade, as the CVE Initiative has correlated and documented over 47,000+ publicly known vulnerabilities in the commercial and open source software used around the globe, there have been some vulnerabilities that were very harmful to pretty much all of us, as well as many that were harmful only to specific types of businesses and business practices.

In the CWE Initiative we have collected and documented over 680 contaminants, or :weaknesses," that can lead to exploitable vulnerabilities, along with over 160 categories and views for collecting and reviewing subsets of the overall CWE content. Within most of the weaknesses in CWE we have included a discussion of the "Common Consequences," trying to describe what the typical consequences could be if a weakness ended up being exploitable when the software is deployed.

We also describe the specific "Technical Impacts" as part of that material, which can be abstracted into eight specific impact categories for all of the CWEs found so far: modify data; read data; DoS: unreliable execution; DoS: resource consumption; execute unauthorized code or commands; gain privileges/assume identity; bypass protection mechanism; and hide activities. What this means is that when a specific CWE is in your software and that weakness ends up being exploitable when the software is performing its intended function in support of your organization, and someone actually exploits it, the outcome will be one or more of these eight technical impacts.

Thus, by establishing a method for reasoning how bad or dangerous these different impacts are to a specific piece of software, given what that software is doing for your enterprise, we can then use this as a key factor in understanding which CWEs we should prioritize above others. We do this by examining the consequences they can lead to, that is, their impacts, and use this to establish part of a standardized mechanism that can be used by CWSS and CWRAF.

DIFFERING BUSINESS CONTEXTS

As you have certain requirements for managing water depending upon your particular needs for it, it's almost a given that each organization will have its own individual set of business priorities, threat environment, and risk tolerance. This makes the development and application of a standardized scoring mechanism difficult, since any assumptions in these areas by the scoring mechanism may not match those of the organization that is applying the scoring. We minimize this difficulty by allowing those working to score the risks a method to model their own organization-specific considerations on business impacts within our risk scoring mechanism/framework. These considerations can then be reflected in the formulas that produce customized risk scores, which are then used to prioritize the various weaknesses and allow an organization to identify which weaknesses are most important to their business, given what their software is doing for their organization.

VIGNETTES

To help capture these business impacts, we introduced a concept called "Vignettes." Basically, a Vignette is a shareable story that describes a particular piece of software's environment within the context and priorities of a business domain such as health, finance, and manufacturing. A Vignette includes the role that the software's archetypes play within that environment, where

© 2012 IEEE. All Rights Reserved.

archetypes are the notional architectural constructs of the system that the software runs within and interacts with, and an organization's priorities with respect to software security for that software. A Vignette identifies the essential resources and capabilities, as well as their importance relative to security principles such as confidentiality, integrity, and availability. For example, in an e-commerce context, 99.999% uptime may be a strong business requirement that drives the interpretation of the severity of the weaknesses that could lead to interruptions in the software's execution.

Vignettes allow the scoring system to support diverse audiences with a wide variety of different requirements for how to prioritize weaknesses and their impact on their businesses. The scoring done with the CWSS occurs within the context of a Vignette.

BUSINESS VALUE CONTEXT (BVC)

An important part of a Vignette is the Business Value Context, or BVC. The BVC contains two main parts:

1. A general description of the security-relevant archetypes, assets, and interfaces that are of concern to the software supporting the business domain.
2. The security priorities of the business domain with respect to the potential outcomes that could occur if the software is successfully attacked.

LINKING BUSINESS VALUE WITH WEAKNESSES

Much like a list of the consequences that could occur if a person ingests water that includes dangerous contaminants, a Technical Impact Scorecard in CWSS and CWRAF is the concept that connects the business concerns in the BVC with the possible technical impacts that could happen if an attacker successfully exploits the weakness. Examples of what could occur if a weakness is exploited include code execution, reading of sensitive application data, or a denial of service. For each potential technical impact, the scorecard assigns a subscore and the rationale for the score. See Table 1 for examples.

When the CWSS score is calculated for a weakness, the values created from the Technical Impact Scorecard are used to drive the score so they reflect the organization's business priorities and requirements as described in the BVC. See Table 1 for examples.

Finally, a Vignette provides the mechanism for calculating CWSS scores so that they reflect the context of the organization for prioritization, as encoded within the Technical Impact Scorecard of the Vignette. See Table 2 for examples.

ENUMERATION OF TECHNICAL IMPACTS

As discussed previously, each weakness, if successfully exploited, can lead to one or more eight possible Technical Impacts:

- Modify data

© 2012 IEEE. All Rights Reserved.

- Read data
- DoS: unreliable execution
- DoS: resource consumption
- Execute unauthorized code or commands
- Gain privileges / assume identity
- Bypass protection mechanism
- Hide activities

Within CWRAF and CWSS, the successful exploitation of a weakness could have varying impacts that occur at four different "layers":

- System - The entity has access to, or control of, a system or physical host.
- Application - The entity has access to an affected application.
- Network - The entity has access to/from the network.
- Enterprise - The entity has access to a critical piece of enterprise infrastructure, such as a router, DNS, etc.

When completing a Technical Impact Scorecard we go through all of the possible combinations of Technical Impact and Impact Layer (32 possibilities), and capture the analysis within the Technical Impact Scorecard by documenting the following information:

- Impact - The kind of Technical Impact under consideration (from the list of eight, currently).
- Layer - The layer at which the Technical Impact could reside. Four impact layers are defined: System, Application, Network, and Enterprise. These layers are used by other factors in CWSS.
- Importance - A value is assigned between 0 and 10 that quantifies the impact of any weakness that can be exploited to have the given Impact at the specified Layer. This is also referred to as a "Subscore."
- Notes - Explanations and rationales for the score, describing the associated business impact if the given weakness could be successfully exploited.

Example - Technical Impact Scorecard

Table 1 shows a subset of Technical Impacts, along with hypothetical subscore evaluation for a notional Vignette, for a piece of software that supports a Web-based e-commerce site.

Table 1: Example Impact Scores

Technical Impact	Layer	Importance	Explanation
Hide activities	Network	3	Inability to identify source of attack. Cannot obtain sufficient evidence for criminal prosecution.
DoS: resource consumption	System	3	Customers experience delays in reaching site; reductions in order placement and resulting financial loss.
DoS: unreliable execution	System	4	Customers cannot reach site due to frequent application crashes; financial loss due to downtime.
Modify data	Application	8	Modify or delete customer order status and pricing, contact

© 2012 IEEE. All Rights Reserved.

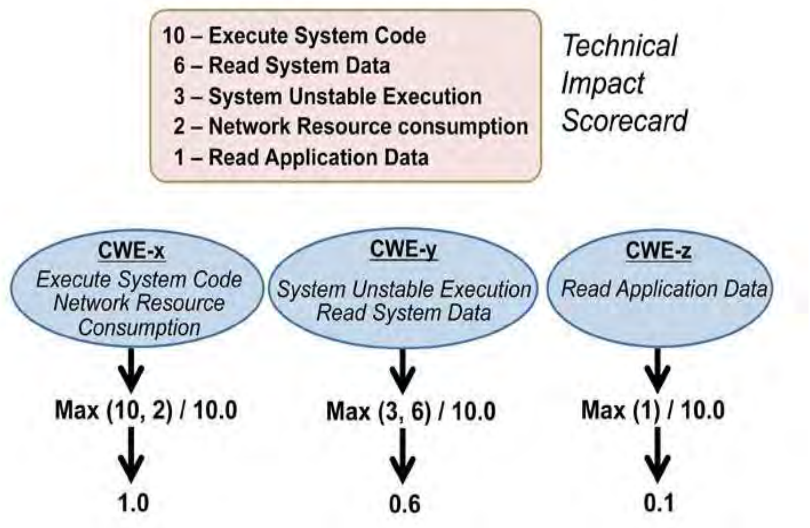
			information, inventory tracking, customer credit card numbers, cryptographic keys, and passwords (hopefully encrypted).
Modify data	Enterprise	10	Modify DNS records to redirect targeted employees to a drive-by-download site that automatically installs malware.
Read data	Application	8	Read customer credit card numbers, contact information, order status, cryptographic keys, and passwords (hopefully encrypted). Read application configuration.
Execute unauthorized code or commands	System	10	Read or modify customer credit card numbers, contact information, order status and pricing, inventory tracking, cryptographic keys, and passwords (plaintext and encrypted). Cause denial of service. Modify web site to deface or install malware to deliver to customers; uninstall critical software.
Bypass protection mechanism	Application	7	Avoid detection of attacks; possibly steal data; pose as other users.

CALCULATING TECHNICAL IMPACT WEIGHTS FROM THE SCORECARD

For each weakness of interest, the weakness is scored as follows:

1) For each relevant CWE entry, extract its potential Technical Impacts. These are specified within the Common_Consequences element of a CWE.
2) For each Technical Impact that is listed in the relevant CWE:
   a) If the Layer is known (e.g., it is a specific weakness finding in a particular software package by a static analysis capability), then look for the line item that has the same Layer and Technical Impact, and use its Importance rating.
   b) If the Layer is unknown (e.g., if a weakness is being given a general score), then search all line items that have the same Technical Impact, and use the maximum subscore of all the items, regardless of the layer being used.
3) Calculate the CWSS Impact factor using the subscore from step 2.a or 2.b.

The approach is roughly as shown in Figure 3:


10 – Execute System Code
6 – Read System Data
3 – System Unstable Execution
2 – Network Resource consumption
1 – Read Application Data
Technical Impact Scorecard
CWE-x
Execute System Code Network Resource Consumption
CWE-y
System Unstable Execution Read System Data
CWE-z
Read Application Data
Max (10, 2) / 10.0
Max (3, 6) / 10.0
Max (1) / 10.0
1.0
0.6
0.1

© 2012 IEEE. All Rights Reserved.

Figure 3: Using the Technical Impact Scorcard for Specific CWEs

Using this method, a Vignette defines the criteria for establishing the relative importance of weaknesses relative to their Technical Impact, e.g., whether the weakness can allow an attacker to read application data, execute code, or cause unstable execution.

Since there are more than 680 weaknesses in CWE, it would be resource-intensive for analysts to evaluate each CWE for a Vignette. The list of technical impacts is much smaller, so it is easier and faster for a human analyst to evaluate. In addition, by Vignettes detailed technical understanding of each weakness is not required to do the analysis.

Note that the importance ratings are allowed to be 0. In many cases, the presence of a 0 rating in a Technical Impact Scorecard is probably an error. However, there may be some BVCs in which a particular impact has no security relevance at all. For example, a product might be single-user only, so the concept of "gaining privileges" at the System layer may not be relevant. Alternately, all data on the product may be intended to be readable by any user or outsider, rendering a 0 subscore for "read data" at the Application layer. While these scenarios may be rare, it seems reasonable to support them in the risk framework.

CALCULATING THE CWE-SPECIFIC TECHNICAL IMPACT SUBSCORE

Once the technical impact scorecard is filled in for a particular Vignette, each CWE entry can be described in light of the entry's technical impacts, as obtained from the Common_Consequences element, as shown in Table 2. Note that this process can be automated.

For example, SQL Injection (CWE-89) will have a subscore of 8, which is the high score from the Technical Impacts of: “Read data” with a subscore of 8; “Modify data” also with a subscore of 8; and “Bypass protection mechanism”; with a subscore of 7.

Similary, Cross Site Scripting (CWE-79) will have a subscore of 10, which is the high score from the Technical Impacts of: “Bypass protection mechanism” with a subscore of 7 and “Execute unauthorized code or commands”; with a subscore of 10.

Finally, Classic Buffer Overflow (CWE-120) will have a subscore of 10, which is the high score from the Technical Impacts of: “Execute unauthorized code or commands” with a subscore of 10 and “DoS: unreliable execution”; with a subscore of 4.

Note that our current approach is that the highest subscore is used as the Impact subscore for the CWSS score of any finding from a static or dynamic analysis capability for the given CWE entry. However, further study and discussion with practitioners and researchers may lead to a more tailored approach.

For more examples of a detailed breakdown of technical impacts and how they fold into the calculation of a CWSS score, all 2011 CWE/SANS Top 25 entries are available on the CWE web site at https://cwe.mitre.org/top25/.

VIGNETTE EXAMPLES

© 2012 IEEE. All Rights Reserved.

Earlier, we referenced a “Web-based e-commerce site” in our Technical Impact Scorecard example. A fuller description of the Vignette for that software, along with a selection of example Vignettes for seven diverse types of software, is presented in Table 2. These Vignettes were selected to represent a diverse set of software from different communities and use cases. A more extensive, up-to-date list of example Vignettes is available on the CWE web site at https://cwe.mitre.org/cwraf/vignettes.html.

Table 2: Example Vignettes

Vignette	**Details**
Web-based Retail Provider	Internet-facing, E-commerce provider of retail goods or services. Data-centric - Database containing PII, credit card numbers, and inventory. Archetypes: Database, Web client/server, General-purpose OS BVC: Confidentiality essential from a financial PII perspective, identity PII usually less important. PCI compliance a factor. Security incidents might have organizational impacts including financial loss, legal liability, compliance/regulatory concerns, and reputation/brand damage.
Financial Trading / Transactional	Financial trading system supporting high-volume, high-speed transactions. Archetypes: N-tier distributed, J2EE and supporting frameworks, Transactional engine BVC: High on integrity - transactions should not be modified. Availability also very high - if system goes down, financial trading can stop and critical transactions are not processed.
Human Medical Devices	Medical devices - "implantable" or "partially embedded" in humans, as well as usage in clinic or hospital environments ("patient care" devices.) Includes items such as pacemakers, automatic drug delivery, activity monitors. Control or monitoring of the device might be performed by smartphones. The devices are not in a physically secured environment. Archetypes: Web-based monitoring and control, General-purpose OS, Smartphone, Embedded Device BVC: Power consumption and privacy a concern. Key management important. Must balance ease-of-access during emergency care with patient privacy and day-to-day security. Availability is essential - failure of the device could lead to illness or death. Devices are not in a physically secured environment.
Smart Meters	Meter within the Smart Grid that records electrical consumption and communicates this information to the supplier on a regular basis. Archetypes: Web Applications, Real-Time Embedded System, Process Control System, End-point Computing Device BVC: Confidentiality of customer energy usage statistics is important - could be used for marketing or illegal purposes. For example, hourly usage statistics could be useful for monitoring activities. Integrity of metering data is important because of the financial impact on stakeholders (consumers manipulating energy costs). Availability typically is not needed for real-time; other avenues exist if communications are disrupted (e.g., site visit).
First Responder	First responder (such as fire, police, and emergency medical personnel) for a disaster or catastrophe. Archetypes: End-point Computing Device

© 2012 IEEE. All Rights Reserved.

	BVC: Communications and Continuity of Operations (COOP) are essential, so availability is extremely important. Integrity is needed to ensure that the correct data is being used for decision-making and communications, such as status updates and contact lists. Confidentiality is, relatively speaking, less important.
State Election Administration Using DRE	State Election Administration using DRE (Direct Recording Election) machines. Archetypes: Embedded Device, Endpoint System, Removable Storage Media, Proprietary Firmware BVC: Integrity essential to election terminals as well as endpoint systems used in pre-election device programming. Protecting PII less important than ensuring accurate vote tabulation and audit trails. Physical security of devices also essential. Help America Vote Act (HAVA) requirements mandate paper audit logs for use by election officials. Security incidents might facilitate fraud via malicious influence of election process or outcomes as well as incur Federal regulatory concerns, and erosion of voter confidence.
Weapon System Sensor	Sensor for a weapons system that is connected to the Global Information Grid (GIG). Archetypes: Real-time Embedded System BVC: Integrity is essential to prevent reporting of false data and faulty decision-making. Lack of availability could cause mission failure. Confidentiality may be slightly less important.

SCORING WEAKNESS FINDINGS USING VIGNETTES – APPLYING CWSS

One of the most important uses of CWSS is to support the automatic scoring of findings that are generated from an automated static analysis code scanner or other tool. By having the various tool and service providers that analyze systems for CWEs offer a consistent approach for prioritizing their findings according to the software's role in the organization, is our collective way out of having a "Clean Water Act" for software contaminants placed upon us all.

CWSS can make use of the Vignette-driven Technical Scorecards or the Technical Impact factor can be assigned through the selection of the discrete values it offers, such as "High" and "Low."

However, as shown in Figure 4, CWRAF Vignettes allows customization of CWSS so that the scores that are generated for tool findings reflect the particular organization's business priorities and risk tolerance. The Technical Impact Scorecard and quantified CWE weightings, constructed using previously-described methods, can be brought into any tool that supports CWSS. This means any tool that uses CWSS can then provide a prioritized list of the findings that is consistent with what other tools that also support CWSS will provide. Organizations may also combine various tools and services and come up with standardized approaches to describing a priori clip-levels about what severities of weaknesses they will tolerate.

Figure 4: CWSS Tailoring Of Static Analysis Tool Findings

Automatically Building Custom Top-N Lists

© 2012 IEEE. All Rights Reserved.

With CWSS and the tailoring that Vignette-base Technical Impact Scorecards allows, an organization can use multiple tools and services for assessing individual pieces of software for the weaknesses that are most dangerous to that software, given what the software is doing for the organization.

In addition, by using CWRAF, an organization can pre-select which CWE entries are of greatest interest to one of its development groups by creating a custom Top-N list for that team. For example, a set of Vignettes can be created that describe the critical types of software that the team typically is asked to develop. Each Vignette would be drafted and the Technical Impact Scorecard filled out and applied to the CWEs to create a ranked list. That list would then be combined to provide an integrated ranking across the Vignettes so that the composite scoring of each CWE by each scorecard is used for the ranking.

The ranked list of CWEs could then be used by that development group to identify which CWEs they need to cover when training their developers and testers. It can also be used to identify which CWEs they need to ask their tool vendors and assessment service suppliers to provide coverage for, as well as in negotiations for outsourcing development and the kinds and types of testing and assessment that the contracted work will be done under.

The process of creating a custom Top-N list involves several steps:

1) Select or manually define an appropriate set of Vignettes, including their Technical Impact Scorecards, and for each Vignette:
    a) Identify which CWSS factors should be treated as Not Applicable (e.g., Remediation Cost).
    b) For each relevant CWE entry, extract its potential Technical Impacts.
    c) Use the Vignettes' Technical Impact Scorecards to evaluate each Technical Impact that is part of the relevant CWE entry. Select the maximum available subscore, regardless of the affected layer.
    d) Calculate the CWSS Impact factor using the maximum available subscore (i.e, use Quantified weighting instead of the pre-defined values for the Impact).
    e) Perform the full CWSS calculation to obtain a general score for the CWE entry (using the "Not Applicable" factors from step 1.a).
    f) Rank all relevant CWE entries according to their CWSS scores.
2) Combine the Vignette ranked lists of CWE entries and combine the weights, either treating each Vignette as equal or by weighting them.
3) Use the top of the combined ranked list as the group's Top-N list.

ARCHETYPES FOR CREATING YOUR OWN VIGNETTES

In the CWRAF section of the CWE web site we have a growing number of Vignettes (https://cwe.mitre.org/cwraf/vignettes.html) and Technical Impact Scorecards (https://cwe.mitre.org/cwraf/groups-domains.html), however anyone can create their own Vignette and its accompanying Technical Impact Scorecard so they can identify which CWEs are most dangerous to their own business and software.

© 2012 IEEE. All Rights Reserved.

One of the items found in these example Vignettes are "Archetypes," which are a general type of technical capability, component, system, system-of-systems, or architecture that is commonly used to support the mission of a particular organization. A list of the currently defined Archetypes available for use in describing Vignettes is provided below in Table 3. If you need to define new Archetypes to cover the environment of your software in your Vignette, we ask that you please send them to us at cwe@mitre.org so we can also add them to our list.

Table 3 – Archetypes for Vignettes

Anti-Virus Program	Authentication Server	B2B Communications
Custom applications	Database	Development Framework
Digital certificate	Distributed Control System (DCS)	Document Processing
Embedded Device	Endpoint System	Firewall
General-purpose OS	Infrastructure as a Service (IaaS)	Internet Communications
J2EE and supporting frameworks	Laptop	Modem Communications
N-tier distributed	PDA	PKI
Platform-as-a-Service (PaaS)	Privacy management	Process Control Systems
Programmable Logic Controller (PLC)	Proprietary Firmware	Remote Terminal Unit (RTU)
Removable Storage Media	Router	SCADA
SOA-based web service	Service-oriented architecture	Smartphone
Software-as-a-Service (SaaS)	Transactional engine	VPN
Virtualized OS	Web application	Web browser
Web browser plugin	Web client	Web proxy
Web server	Web service	Wireless Communications

The Archetypes listed above are used as the context for describing the technical elements utilized by the application described in the Vignette, and can be used to help identify which CWEs are possible. For instance a Vignette without the Database Archetype probably won't have any SQL-related CWEs (CWE-89 and CWE-564).

Community Adoption and Engagement

In conclusion, as the various systems for managing water have been developed and become standardized over the years, the effort to create a standardized common scoring system for weaknesses in software started many years ago and has finally come to the point where the CWE, CWSS, and CWRAF efforts—all of which are Software Assurance strategic initiatives co-sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security —are being adopted by the community and integrated into static analysis offerings and other solutions in order to help organizations properly deal with these dangerous contaminants to software.

As an example, when the most recent versions of CWSS and CWRAF were launched in June 2011, Cenzic, GrammaTech, Veracode, Coverity, CAST, Klockwork, Fortify, and SANS all publicly declared their support and plans to incorporate CWSS, while Trustwave, DTCC, SAIC, EC-Council, CISQ, and OWASP all declared their plans to work on using and evolving CWRAF to meet their customers' and the community's needs for a scoring system for software errors.

These standardization efforts certainly help the community to address the contaminated software problems that still faces consumers of software today, as well as to avoid regulatory solutions

© 2012 IEEE. All Rights Reserved.

such as those that exist for water's various uses, but additional community participation and adoption of these efforts is still needed. If your static analysis vendor or assessment service supplier isn't on the list above please ask them to use CWSS and CWRAF to reduce your organization's risk and ensure that those software weaknesses that are most dangerous to your organization are identified and dealt with first. We welcome your input at cwe@mitre.org.

REFERENCES

[1] "The Common Weakness Enumeration (CWE™) Initiative", MITRE Corporation, (https://cwe.mitre.org/)

[2] "The Common Weakness Scoring System (CWSS™)", MITRE Corporation, (https://cwe.mitre.org/cwss/)

[3] "The Common Weakness Risk Analysis Framework (CWRAF™)", MITRE Corporation, (https://cwe.mitre.org/cwraf/)

[4] "The Common Vulnerabilities and Exposures (CVE®) Initiative", MITRE Corporation, (https://cve.mitre.org/)

[5] "The CWE/SANS Top 25 Most Dangerous Software Errors List", MITRE Corporation, (https://cwe.mitre.org/top25/)

[6] "Lead Poisoning", Wikipedia Article, (http://en.wikipedia.org/wiki/Lead_poisoning/)

[7] "Cleaning up the Nation's Hazardous Waste Sites", EPA Superfund Website, (http://www.epa.gov/superfund/)

© 2012 IEEE. All Rights Reserved.