# OSBR.CA

The Open Source Business Resource

JUNE 2008

# JUNE 2008

**PUBLISHER:**
The Open Source Business Resource is a monthly publication of the Talent First Network. Archives are available at the website: http://www.osbr.ca

**EDITOR:**
Dru Lavigne
dru@osbr.ca

**ISSN:**
1913-6102

**ADVERTISING:**
Rowland Few
rowland@osbr.ca

**GRAPHICS:**
Ryan May

***If you google the phrase*** "open source security", you'll find plenty of articles which debunk the "myth" of open source security, fuel the debate of Linus' law (http://en.wikipedia.org/wiki/Linus%27s_Law) vs. security through obscurity (http://en.wikipedia.org/wiki/Security_through_obscurity), or argue which type of software, proprietary or open source, is more secure. Yet, the question "which type of software is more secure?" is impossible to answer. Software security is highly dependent upon many variables: the programming language used, the practices implemented by the individual programmers, the processes imposed by the specific organization overseeing the programmers, and the configuration of the software by a particular end-user.

***This issue of the OSBR*** examines several facets of open source security. Jake Kouns from the Open Security Foundation introduces an open source project which manages a global collection of vulnerabilities, available for free use by the information security community. David Maxwell from the Coverity Scan project discusses their report on code defect trends from an analysis of several hundred open source projects, representing 55 million lines of code, through 14,000 build sessions over a two year period.

***Security research is led by*** the Government of Canada and Canadian universities. Robert Charpentier from Defence Research Establishment Valcartier and Mourad Debbabi, Azzam Mourad, and Marc-André Laverdière of Concordia University present key concepts related to security hardening and their applicability to the C programming language. Frederic Michaud and Frederic Painchaud from Defence Research and Development Canada discuss the results and recommendations from their analysis of automatic source code verifiers that search for program sanity and security bugs.

***In addition to the articles,*** Michael Geist, Canada Research Chair of Internet and E-commerce Law, discusses Bill C-61, which will amend the Canadian Copyright Act, and suggests actions for those who disagree with the proposed legislation. Alan Morewood from the security division of Bell Canada provides an example of a business reason for using open source to assess an organization's security risk. This month's conference report covers trends in technology marketing and how to build successful communities.

***As always, we look forward*** to your feedback. In particular, we're interested in your suggestions for editorial themes beyond the September issue. If you have a topic you would like to see discussed, send an email to the editor.

**Dru Lavigne**

**Editor-in-Chief**

**dru@osbr.ca**

*Dru Lavigne is a technical writer and IT consultant who has been active with open source communities since the mid-1990s. She writes regularly for O'Reilly and DNSStuff.com and is author of the books BSD Hacks and The Best of FreeBSD Basics.*

*"Secrecy prevents people from accurately assessing their own risk. Secrecy precludes public debate about security, and inhibits security education that leads to improvements. Secrecy doesn't improve security; it stifles it."*

Bruce Schneier
http://www.schneier.com/blog/archives/2007/01/debating_full_d.html

This article introduces the Open Source Vulnerability Database (OSVDB, http://osvdb.org/) project which manages a global collection of computer security vulnerabilities. It is freely available to the information security community. This collection contains information on known security weaknesses in operating systems, software products, protocols, hardware devices, and other infrastructure elements of information technology. The OSVDB project is intended to be the centralized global open source vulnerability collection on the Internet.

**Vulnerability Databases**

A vulnerability is an error or weakness in a component that allows it to be attacked, resulting in unauthorized use of the item or in damage to it and components connected to it. In an information technology network like the Internet, successful exploitation of vulnerabilities can result in operating system damage, illegal release of information, data destruction, disruption of service, and a galaxy of other tribulations.

Although we often discuss vulnerabilities in general terms like "open to man-in-the-middle attack" or "allows remote buffer overflow", attackers and defenders know that the essence of a security vulnerability is never the general description, but rather the vulnerability's specific details. There are very few generic attacks that will work against multiple targets. Similarly, there are few general vulnerabilities that simultaneously affect different network components. Instead, the classic vulnerability affects a single feature of one release of a software product installed under a single operating system, a feature that can be exploited in only one way.

Out of the trillions of lines of code running in networked systems, a vulnerability may exist in a single line. It is a unique grain of sand in a mile-long beach. How do those with systems containing that unique flawed line know they are potential victims? And how do they identify a solution?

As the number of network components grows every year, the number of vulnerabilities also grows. Annual vulnerability announcements now number in the thousands, well beyond the capacity for human memory to manage. Well-organized databases, with verified contents and flexible search abilities, are required if these vulnerabilities are to be controlled by the security community.

A vulnerability database serves many communities: businesses need to know whether elements of their current or planned computing environment are susceptible to security failures, system administrators want alerts to relevant security malfunctions and their cures, software developers need warning when their products have shown security flaws, and security practitioners depend on a comprehensive and standardized vulnerability list to build products and services.

Historically, it has been difficult to develop a comprehensive, unbiased, and timely resource that provides for these needs. One reason for the difficulty is that documenting and disseminating vulnerabilities has become an enormous task.

CERT, the security vulnerabilities research center at Carnegie Mellon University's Software Engineering Institute, identified just 171 vulnerabilities in 1995, but reported 7,236 in 2007: an increase of over 4,000 percent in twelve years (http://cert.org/stats/fullstats.html). CERT's counts are considered conservative and the actual number of vulnerabilities facing administrators, developers, and organizations may actually be higher.

The effort required to track vulnerabilities exceeds the resources of most organizations, and the volume of information appearing each year is unlikely to decrease. To meet the growing need for vulnerability management, OSVDB harnesses the efforts of the world's security practitioners and the power of the open source development model to locate, verify, and document this critical information.

OSVDB provides the necessary structure, technology, and content to support the security community's requirement for vulnerability management. OSVDB aims to be the leading open source project in its field by helping practitioners evolve and move beyond the current mainstream reactionary model. By maintaining a close connection with the security community, by remaining unaffiliated with commercial interests and open to community content development, and by actively promoting excellence in its operation, OSVDB will provide a stable, world-class resource for all security projects and practitioners.

## The OSVDB Project

The OSVDB project was launched in 2002 following a realization in the security community that no independent, community operated vulnerability database existed. There were, and still are, numerous vulnerability databases. Some of these databases are managed by private interests to meet their own requirements, while others contain a limited subset of vulnerabilities or have significant restrictions on their content. OSVDB's project leaders have set out to implement a vulnerability database that meets three requirements. The database must be: i) comprehensive; ii) open for use; and iii) answerable to the community.

OSVDB is currently an active web application available at http://www.OSVDB.org. The project was originally deployed in two major parts: a front end allowed vulnerabilities to be searched for and reported on, and a back end allowed contributors to add or edit vulnerabilities. In order to streamline the process, OSVDB has recently implemented a customizable portal that fully integrates the old back end interface and the front end website. In addition, the method for updating vulnerabilities has been changed to a wiki-like system that allows contributors to edit individual fields when needed. OSVDB is also available for download in multiple database export formats and as a very small Ruby on Rails application. This application utilizes our SQLite database export to give a user their own, albeit relatively featureless, local OSVDB instance.

OSVDB moderators identify new vulnerabilities and assign them a unique identifier. This allows contributors the ability to scour the web for information describing a vulnerability, then capture the details in a database record within OSVDB itself. A moderator checks each vulnerability entry before it is committed, to ensure that the OSVDB's standards for clarity and correctness are met. Once the update has been accepted, it is available to anyone requiring vulnerability information from the database.

The process is rapid, making new vulnerabilities available to the community quickly. It is also efficient, maximizing productivity for the contributors and moderators so that the team can keep above the rising tide of vulnerability data. The online process and the automation that supports it have been improved continuously since the project opened, and the OSVDB team will continue to add value to the basic database and associated services over time.

**Project Goals**

Many security endeavors benefit from a single source listing all vulnerabilities. This is in contrast to a federated approach where multiple vulnerability lists have to be queried and the results combined to get a comprehensive result. Developers creating vulnerability assessment tools, system administrators protecting servers and networks, business staff assessing risks and remedies, academic researchers documenting and analyzing the past and future of network security. All invest effort in identifying vulnerabilities, all work to document them consistently, and all can benefit from a single, comprehensive source of vulnerability data. The OSVDB project reduces duplication of effort and promotes data consistency.

Serious users of any database evaluate its sources and practices before placing trust in its contents. OSVDB is unbiased and neutral in its practices for accepting, reviewing, and publishing vulnerabilities. Its open acceptance of community input and internal review processes ensure that the vulnerability database is not colored by vendor biases. The OSVDB team works hard to ensure that the content evenly reflects the actual distribution of vulnerabilities, neither over-exposing nor under-exposing particular operating systems, products, or vendors.

Some experts have raised concerns that such a comprehensive security database may present potential dangers of its own. This is security's classic disclosure (http://en.wikipedia.org/wiki/Full_disclosure) problem. Can a vulnerability database help an attacker? It may do so, but it provides a far more significant benefit for defenders. Google can be considered the largest and most detailed vulnerability database in the universe. It operates whether or not other vulnerability lists exist, and provides the ultimate resource for the dedicated attacker.

Given the breadth of information security problems affecting businesses and individuals, it is easy to understand that subscribers to security information span a wide range of technical backgrounds and skills. At times, some software vendors have been criticized for releasing vulnerability information that lacks the details system administrators need. Others have drawn fire for complex vulnerability reports that confuse home users and non-technical staff. OSVDB includes both business-level descriptions and the technical details for the vulnerabilities in the database. Creating and supplying the proper type of information for the intended audience allows OSVDB to serve all consumers of vulnerability information.

Many security operations, whether stand-alone organizations or security departments within enterprises, operate under tight funding, and need to rely on the free efforts of others to be successful. OSVDB's features and services benefit all security practitioners because they are universally available, without distribution controls and without fees or charges. OSVDB deliverables can be freely used, whether as stand-alone components or integrated into other tools.

For example, an open-source web vulnerability scanner like Nikto (http://www.cirt.net/nikto2) or Nessus (http://www.nessus.org/) can use OSVDB data to populate reports from a vulnerability scan. Both development teams conserve effort in finding and documenting vulnerabilities, and the security community benefits from comprehensive and consistent reporting capabilities.

OSVDB organizers believe that more than one vulnerability database is needed to meet the full variety of community requirements. The 2nd Workshop on Research with Security Vulnerability Databases, stated that "no single proposition satisfies all parties involved" and that the parallel pursuit of different strategies would have the best opportunity for success. OSVDB intends to fulfill the recognized community requirements for an open, centralized resource.

While it references other vulnerability databases (http://www.osvdb.org/ext_references), it develops its own database entries to ensure that there are no restrictions on distribution and re-use of OSVDB vulnerability data. Its contents are free of cost and free of restrictions on use under the terms of the OSVDB Free License (http://osvdb.org/osvdb_license).

**Project Accomplishments**

Since March 31, 2004, when the OSVDB first opened for public use, the project has reached many milestones, including:

1. The formation of the Open Security Foundation (http://www.opensecurityfoundation.org/) a non-profit public foundation which provides independent, accurate, detailed, current, and unbiased security information to organizations, protects the OSVDB from commercial acquisition, and formalizes the tax status of contributors.

2. The creation of the OSVDB vendor dictionary, a free resource through which the security community is able to gather vendor contact information. The vendor dictionary is a list of vendors, indexed by name, which may be freely searched and utilized by all who wish to find both general and security contact information. The service also provides a way for vendors to keep their information current within the dictionary.

3. The OSVDB blog (http://osvdb.org/blog/) started as a way for the project to keep the public better informed on the project's status. Very quickly, the blog became a place to discuss and comment on various aspects of vulnerabilities, and has become a successful mechanism for communicating with the security industry.

4. A custom portal was implemented to allow users to define specific alerting of vulnerabilities with OSVDB's Watchlist service. This service allows users to track new vulnerabilities by vendor or products and also consolidates vendor security mailing lists.

5. The OSVDB displays relevant blogs for additional reading and has the ability for security practitioners to comment on specific vulnerabilities. While OSVDB has made every effort to include all references in some fashion, we have implemented a concise method for the community to add information about a vulnerability.

6. A detailed classification system allows OSVDB to track numerous fields for each vulnerability. The enhanced data allows users to find vulnerabilities based on criteria such as attack type, solution status, or whether or not the vulnerability has been confirmed or disputed by the vendor.

7. Integration and cross-referencing of OSVDB via the application programming interface (API) which can provide multiple result formats to fit various needs. Queries can be run against any number of correlation factors, including CVE ID (http://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures), Microsoft Bulletin ID, Bugtraq ID (http://en.wikipedia.org/wiki/Bugtraq), and a host of other common reference points.

8. The OSVDB supports multiple database export formats (XML, SQLite, MySQL and CSV) as well as a small Ruby on Rails application that utilizes a SQLite database export to give a user their own local OSVDB instance.

**Future Plans**

The OSVDB is working towards the following objectives:

1. The OSVDB Vulnerability Disclosure Framework, a service to help to improve, streamline and, more importantly, remove the mystery and breakdowns in the disclosure process. The framework will assist researchers and vendors to better coordinate disclosing vulnerabilities.

2. A policy on the release of vulnerability information which incorporates clear guidelines on the timing of notification to the product developer and of notification to the open security community. In addition, a formal statement of policy for handling previously-unknown (0-day, http://en.wikipedia.org/wiki/Zero_day_attack) security vulnerabilities and exploits, covering communications with affected vendors as well as with the security community.

3. Recruitment of more security professionals to maintain and extend the vulnerability database and formal recognition of contributors and identifying lead contributors to support organizations underwriting their time and effort.

4. Active integration with vulnerability tools to streamline the process of identifying and setting priorities for the identified vulnerabilities. OSVDB will assist tool developers to identify vulnerabilities that are not already represented in their products, and will provide a way to identify the high-priority vulnerabilities for immediate attention.

5. The creation of a Vulnerability and Patch Management Portal to create a flexible framework that can provide organizations with the ability to track and manage vulnerabilities and patches. OSVDB is looking to not only provide information on vulnerabilities, but also a service that can provide security professionals a way to track and ensure that vulnerabilities have been addressed at their organization.

6. The OSVDB Training Portal Framework will create a flexible framework that can provide training on security issues. The OSVDB aims to be a repository for training information that will help educate end users on how to avoid security risks and developers on how to avoid coding insecure applications.

7. The OSVDB Port Listing Project will be a central repository for all known ports and protocols. This will be the foundation for many new features such as referencing ports and protocols to OSVDB vulnerabilities. This will then allow the OSVDB to be better mapped to firewall rules, intrusion detection system (IDS) alerts, and potential integrations to other security projects.

8. Long term sponsorship to provide additional services and an improved dataset.

## Conclusion

The OSVDB provides an important service for the security community by maintaining and propagating an open, freely available database of security vulnerabilities. As Stewart Brand said, "information wants to be free". This is doubly true for security information, which can protect network users and organizations from harm. The project is already significant to the world security community, and it will increase in importance as its contents grow and as it adds features and services over time.

The OSF, a non-profit organization which oversees the operations of the OSVDB, was setup as an umbrella organization to support open source security projects. Another project that continues to provide value to the community is the DataLoss DB (DLDOS, http://attrition.org/dataloss/dldos.html), run by Attrition.org since July 2005. It will be formally maintained as an ongoing project under OSF. The DLDOS project's core mission is to track data loss and data theft incidents--whether confirmed, unconfirmed, or disputed--not just from the United States, but across the world. As of June 4, 2008, DLDOS contains information on over 1,000 breaches of personal identifying information covering over 330 million records.

*This article is based upon the whitepaper entitled OSVDB Aims. The original whitepaper is available in HTML and PDF at the OSVDB website (http://osvdb.org/documentation).*

*Jake Kouns is the co-founder and President of the Open Security Foundation (http://www.opensecurityfoundation.org/) which oversees the operations of the Open Source Vulnerability Database (OSVDB). Kouns' primary focus is to provide management oversight and define the strategic direction of the project. He holds a Bachelor of Business Administration with a concentration in Computer Information Systems and a Master of Business Administration with a concentration in Information Security from James Madison University.*

**Recommended Resources**

Requirements and Approaches for a Computer Vulnerability Data Archive
http://www.blackmagic.com/ses/bruceg/workshp.html

Sharing Vulnerability Information Usinga Taxonomically-Correct, Web-Based Cooperative Database
https://www.cerias.purdue.edu/papers/archive/2001-03.pdf

Data Mining in Vulnerability Databases
http://www.ito.tu-darmstadt.de/publs/pdf/sdb-dfn-cert-eng.pdf

*"In programming, as in everything else, to be in error is to be reborn."*
Alan J. Perlis, first recipient of Turing Award

On May 20, 2008, static analysis tool vendor Coverity released a report entitled "Open Source Report 2008" (http://scan.coverity.com/report/). The report includes information gathered over the first two years of the Coverity Scan project which was developed as part of a contract from the US Department of Homeland Security. Coverity provides its analysis tools to open source projects in order to identify quality and security flaws in the codebases. Once identified, the developers of the open source projects are given the information in order to facilitate hardening of the software.

The report includes information about the progress made by various projects using the Scan service. Additionally, the Scan databases constitute one of the largest and most diverse collections of source code to be built and analyzed while tracking changes to those code bases over a two-year period. This data provides a substantial set of samples for considering some questions about the nature of software. The report investigates relationships between codebase size, defect counts, defect density, function lengths, and code complexity metrics. This article highlights some of the results from the report.

**Data Used in the Report**

Software has become a larger part of our lives over the last few decades. Whether on a desktop computer, or in systems we use like bank machines and automobiles, there are few people left who don't interact with software on a daily basis. Flaws in software can lead systems to misbehave in ways that range from simply annoying to life-threatening.

Yet, although software plays such a ubiquitous and critical role in daily life, there are still many unanswered questions about how to develop good software and how to measure the quality of software.

Coverity is a software vendor that develops tools to automatically identify flaws in source code. While normally sold to commercial software developers, the US Department of Homeland Security contracted Coverity to analyze open source software (OSS) and provide the results to open source developers so that they could fix the defects that were identified.

Coverity's "Open Source Report 2008", includes a sampling of some of the data collected since the launch of the project in March of 2006. The information in the report falls into a number of different categories. There is data about the degree of improvement and regression in quality by the open source projects using the Scan site. There is data about the frequency of different types of defects identified by the analysis and information about the consequences of each type of defect. There are statistical correlations between various measurements of the software projects that are being tracked and statistics about the proportion of defects where the developers determined that the analysis tool was incorrect when it claimed there was a defect.

The data in the report is based on open source projects which add up to 55 million lines of code. In over 14,000 build and analysis sessions over two years, almost 10 billion lines of code were run through the analysis engine. In addition to looking for defects, the analysis retains information about the code itself, such as the names and numbers of functions and their lengths, the files that comprise the various projects, and the calculated complexity metrics.

Commercial software is not usually available for analysis on an ongoing basis such as that performed in the Scan project. Commercial developers often do not release their source code, and when they do, it is typically in the form of a specific release version which usually receives a thorough vetting before its public release.

In contrast, open source projects make their source code available in a public version control system. Anyone who is interested can track the changes that the developers make day by day. This provides a visibility into the software developer process that would not exist without open source principles.

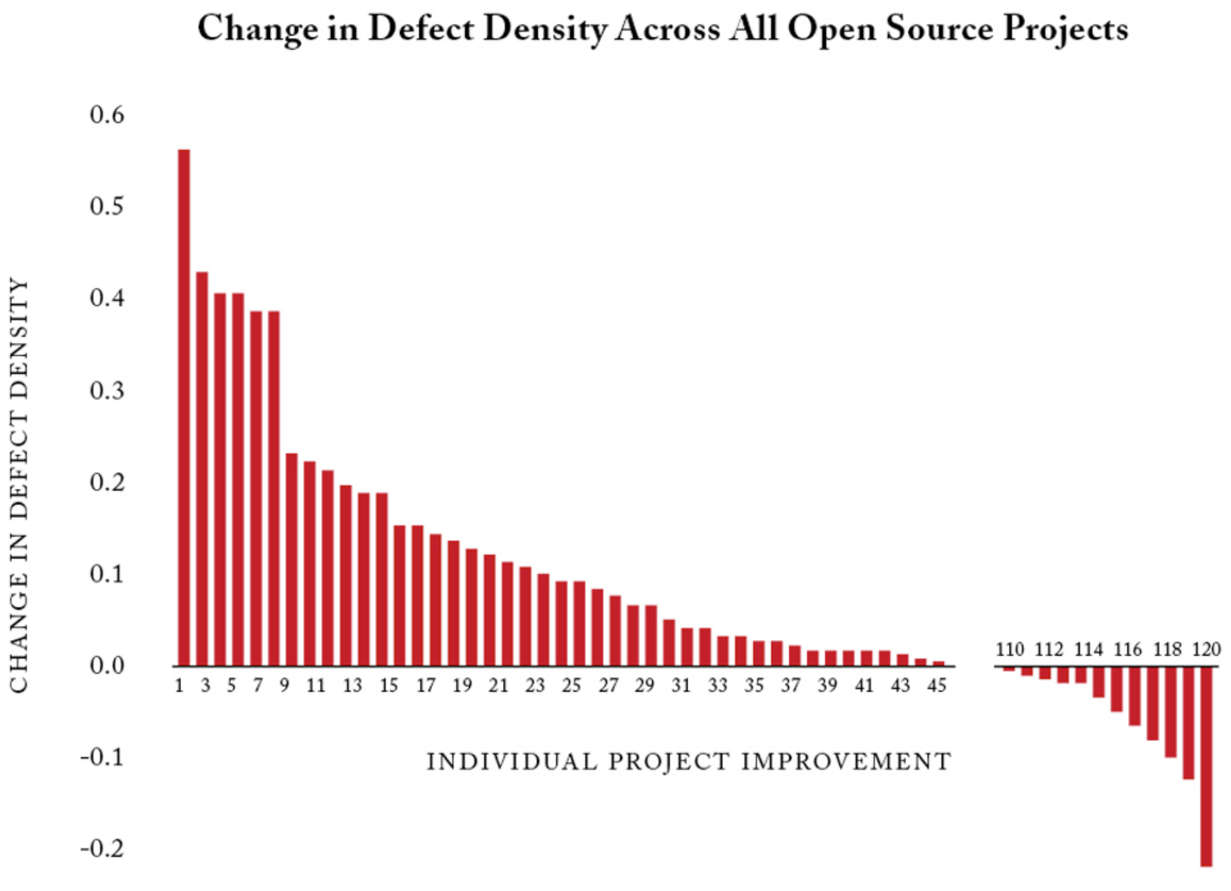
When a large number of projects are viewed together, the result is a sample set of data that can begin to answer many questions about the nature of software.

Knowing the answers to these initial questions allows us to begin to formulate more sophisticated questions for future efforts to pursue.

## Report Findings

Defect densities are measured in number of defects per 1,000 lines of code. Over the two years from March 2006 to March 2008, the average defect density in the projects being monitored dropped from 0.30 defects per thousand lines of code to 0.25, or from roughly 1 defect per 3,333 lines of code to one defect per 4,000 lines of code. This represents an overall improvement of 16%. Figure 1 shows the change in defect density.

**Figure 1: Change in Defect Density**

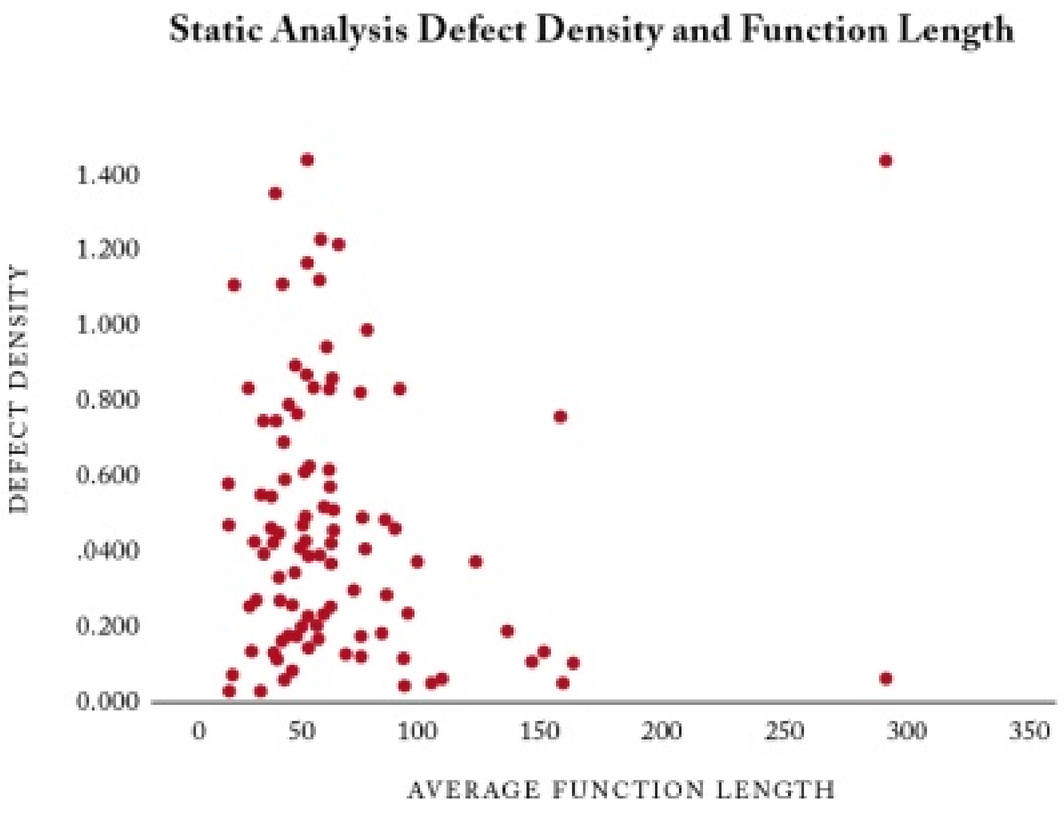


A statistical correlation was performed to compare defect densities and average function length in each project. The confidence factor found was 1.49%, where confidence factors can range from 0% to 100%. Data included in the report show no correlation between defect density and average function length. Since best practices often stipulate that long functions should be re-factored into multiple smaller functions, support for that practice would be demonstrated if the data showed a correlation of higher defect densities with longer average function lengths. While there may be advantages to re-factoring to assist in code maintainability, shorter functions do not seem to guarantee improvements in defect density. Figure 2 shows the relationship between defect density and function length.

An additional correlation was performed between codebase size in lines of code, and number of defects identified. A commonly repeated statement is that software complexity grows as project sizes grow, but to an exponential extent. That is, it is often asserted that adding additional code to a project adds complexity due to all of the interactions with the existing code.

The codebase size and defect count correlation was 71.9%, which indicates that the increase in defect count is largely linear with the growth in number of lines of code. If the increase in complexity leads to more defects, and more frequent defects, then a linear correlation ought to be much lower than the nearly 72% figure.

**Figure 2: Static Analysis Defect Density and Function Length**

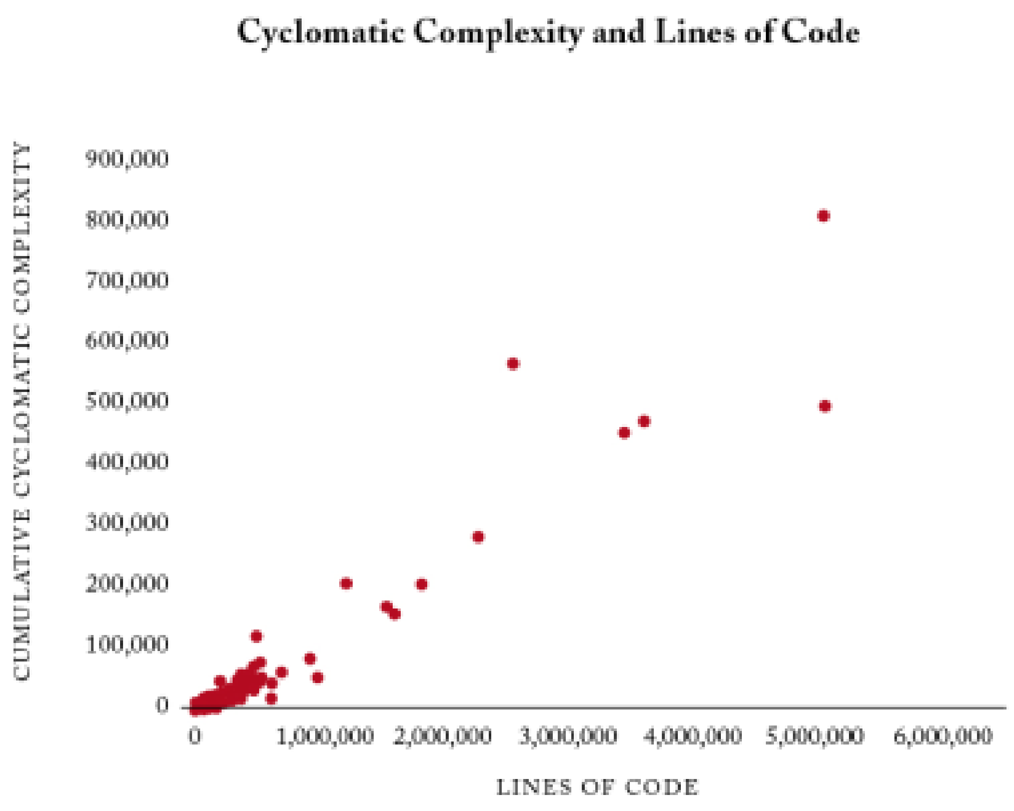


This appears to indicate that writing 1,000 additional lines of code to be added to a large codebase (say, over 1 million lines) is no more difficult or error prone than writing 1,000 lines of code to be added to a small codebase.

This finding has the potential benefit of alleviating a concern about software development. It has been speculated that software applications will become so large that they will become literally unmanageable. While there may be other aspects and limitations to the management of large projects, there does not appear to be an upper limit on project size causing defects to be created at an unmanageable rate.

Comparisons are made in the report between codebase size and the calculated complexity metrics for each codebase. Cyclomatic complexity is an algorithm for measuring the number of independent paths through a piece of source code (http://en.wikipedia.org/wiki/Cyclomatic_complexity). The total cyclomatic complexity of an application was found to correlate almost 92% to the number of lines of code in an application. This implies that calculating the complexity metric for a codebase may tell you more about how much code you have than about its complexity. Figure 3 shows the correlation between complexity and lines of code.

**Figure 3: Cyclomatic Complexity and Line of Code**



Since the complexity metric is so strongly related to codebase size, it may be important to double-check one's assumptions about the meaning of complexity metrics and determine whether the way in which they are being used is appropriate, given the information they convey.

When discussing the results of the report, there is a common desire to draw comparisons between open source code quality and commercial source code quality. While this issue is addressed in the report, it is not answered. The lack of availability of a wide sample set of commercial source code may make it impossible to ever perform an analysis similar to that done for open source code in the released document.

The report also includes information about the rate of false positives identified in the analysis by developers looking at the results for their codebases. The false positive rate is an important metric for any static analysis tool, because some portion of the analysis is constrained by information that will only be available at runtime. Any tool performing static analysis will identify some issues that cannot happen at runtime, and will fail to identify some issues that can. The critical factor is the degree to which a tool identifies code defects that are valuable to resolve, while not reporting so many false issues that developers become frustrated with the inaccuracy of the tool. To date, developers have identified only 13.32% of the results in the Scan project as false positives.

Finally, the report concludes with appendices covering the specific statistical methods applied to the data, and additional details about the consequences of the various types of defects identified in the code that was examined.

**Conclusion**

It is expected that feedback from readers of the report will drive deeper investigations into the available data, which may uncover further interesting trends in the nature of software development.

Coverity intends to issue updated reports on an annual basis, comparing year-over-year trends, and overall progress by the projects involved in the Scan. As updated tools are made available to the open source developers, the results will include new defect types and changes in the overall distribution of defect type frequencies.

*David Maxwell is Coverity's Open Source Strategist, and is tasked with the continuation and expansion of Coverity's DHS-sponsored open source scans. An open source security specialist, Maxwell has over 20 years of experience as an open source user and developer, and he is particularly active in the NetBSD community. He currently sits on the advisory board for the BSD Certification Group and the program committee for the annual BSDCan conference. He was also a NetBSD Security Officer from 2001-2005 and a contributor to the best-selling O'Reilly title "BSD Hacks." Maxwell has previously worked as a lead kernel developer for Nokia, and architected the Internet Service offering for Fundy Cable in New Brunswick.*

*"C is quirky, flawed, and an enormous success."*

Dennis Ritchie
http://cm.bell-labs.com/cm/cs/who/dmr/index.html

In today's computing world, security takes an increasingly predominant role. The industry is facing challenges in public confidence at the discovery of vulnerabilities and customers are expecting security to be delivered out of the box, even on programs that were not designed with security in mind. Software maintainers face the challenge to improve the security of their programs and are often under-equipped to do so. Some are taking advantage of open source software (OSS) as the availability of the source code facilitates their validation and answers their need for trustworthy programs. OSS are often implemented using the C programming language (26% according to SourceForge.net). This makes it necessary to investigate the security issues related to C.

This paper summarizes key concepts related to security hardening, and demonstrates its applicability on the C language. We also propose a progressive approach to integrate security services and protection measures into existing software to ultimately make it more resistant against cyber-attacks. Given our ever increasing dependability on information technologies, it becomes critically important to provide tools to maintainers that will facilitate and accelerate the security hardening process, increasing the effectiveness of the effort and lowering the resources required to do so.

**Software Security Hardening**

Security hardening of software is an informal term, but the technical community considers it to be an iterative process to progressively implement security services and protection measures.

The process starts with the basic software that has being designed and implemented to offer some functionality as typically defined by use cases. As a first step toward better protection of data, security services are introduced to implement features associated with authentication, access control, confidentiality, and integrity. These services are typically described via security use cases. However, this is not sufficient. It is often necessary to define misuse cases to protect the software against users' mistakes and other errors that could happen in any system operated by humans in a complex execution environment. Moreover, it is often required to test software against abuse cases that model deliberate attacks that could be encountered in a hostile environment. Depending on the criticality of the system being designed, it may be necessary to harden the key components to the highest level, including security services and protection measures against misuses or deliberate attacks. Some other less critical components can only be hardened to a lower level.

In practice, the risk analysis may lead to changes in the source code, the development process, the overall design, or even the operating environment itself as described in the following classification of security hardening methods:

*Code-level hardening* implies changes in the source code in a way that prevent vulnerabilities without altering the design. Some vulnerabilities are a direct result of the programming activities and code level hardening removes these vulnerabilities in a systematic way.

*Software process hardening* is the replacement of the development tools and compilers, the use of stronger implementations of libraries, and the execution of complementary test suites which implement security scenarios.

*Design-level hardening* consists of the re-engineering of the application in order to integrate security features that were absent or insufficient. Some security vulnerabilities cannot be resolved by a simple change in the code or by a better environment, but are due to a fundamentally flawed design. This category of hardening practices targets more high-level security such as access control, authentication and secure communication. In this context, best practices and security design patterns can be redirected from their original intent and used to guide the redesign effort.

*Operating environment hardening* stands for improvements to the security of the execution context (network, operating systems, libraries, etc.) that the software relies upon. Those changes typically make exploitation of vulnerabilities harder, although they do not remedy them.

The spectrum of changes that may be required is very broad and security analysts usually take two complementary perspectives to address key security issues. Typically, analysts prefer to start with the high-level perspective before engaging into code changes or other low-level security issues.

**High-Level Perspective**

From the high-level perspective, the attention will be put on design-level hardening and on the relationship that the system has with its operating environment. The goal is to identify more precisely the threats, to evaluate the real risks, and to propose countermeasures.

Identifying threats is an important task in security hardening since we need to determine which threats require mitigation and how to mitigate them, preferably by applying a structured and formal mechanism or process.

As such, the following is a brief description of the three main steps needed to identify and evaluate the risk of a threat:

1. Application decomposition divides the application into its key components in order to identify their trust boundaries. This decomposition helps to minimize the number of threats that need mitigation by excluding those that are outside the scope and beyond the control of the application.

2. Threat identification categorizes according to the six known categories presented by Howard and LeBlanc: spoofing identity, tampering with data, repudiation, information disclosure, denial of service and elevation of privilege (http://www.microsoft.com/mspress/books/5957.aspx).

3. Risk evaluation is needed to determine the priority of threats to be mitigated.

Once the previous steps are completed and the threat is well identified and categorized, it is possible to determine the appropriate mitigation technique(s). It is possible to find mappings between the categories of threats and known countermeasures. Howard and LeBlanc provide a list of mitigation techniques for each category of threats within their classification.

For example, against the threat of spoofing identity, they recommend using appropriate authentication and to protect secret data; against information disclosure, they recommend using authorization and encryption. Regarding the deployment of these techniques into applications and systems, security patterns are useful to choose the best techniques available, and guide their implementation.

### Low Level Perspective

From the low-level perspective, the attention will be on the source code itself and on the methodologies (tools and techniques) used to build software systems. Software analysts often use automated tools to find the software constructs that are problematic or exploitable in an attack scenario. Some tools use static code analysis (http://en.wikipedia.org/wiki/Static_code_analysis) to find potential implementation flaws. It may be necessary to complement the security analysis of the code with a run-time tester.

When code review is performed, it is important to evaluate the impact of a software defect because it may result in a real vulnerability that will represent an exploitable weakness. Even though many tools exist to help identify vulnerabilities, no tools are perfect. Some tools are good at certain types of defects while others may simply miss them. False positive diagnostics are often the most difficult problem software analysts encounter. [Editor's Note: readers interested in the findings of research into this subject will find details in the article Language Insecurity.]

### Notorious Vulnerabilities in the C Language

In this section, some major safety vulnerabilities of C programming are presented along with the hardening techniques used to remedy them at different levels. They are recognized as being among the most notorious source of problems in software security and reliability. They illustrate the multi-layer approach that is needed to cope with them in a rigorous manner.

1. *Buffer overflows* exploit common programming errors that arise mostly from weak or non-existent bounds checking of input being stored in memory buffers. Buffers on both the stack and the heap can be corrupted. Many APIs (application programming interfaces) and tools have been deployed to solve the problem of buffer overflow or to make its exploitation harder. Table 1 summarizes the security hardening solutions for buffer overflows.

**Table 1: Hardening for Buffer Overflows**

| Hardening Level | Product/Method |
|---|---|
| Code: | Bound-checking, memory manipulation functions with length parameter, ensuring proper loop bounds, format string specification, user’s input validation |
| Software Process: | Compile with canary words, inject bound-checking aspects |
| Design: | Input validation, input sanitization |
| Operating Environment: | Disable stack execution, use libsafe (http://directory.fsf.org/project/libsafe/), enable stack randomization |

2. *Integer security issues* are caused by converting between signed and unsigned, sign errors, truncation errors and overflow and underflow. Those vulnerabilities can be solved using sound coding practices and special features in some compilers such as replacing integer operations with safer calls. The security hardening solutions for such problems are summarized in Table 2.

3. *Hardening for memory management vulnerabilities.* The C programmer is in charge of pointer management, buffer dimensions, allocation and de-allocation of dynamic memory space, all of which may cause memory corruption, unauthorized access to memory space, and buffer overflows. Security hardening solutions against such problems are summarized in Table 3.

**Table 2: Hardening for Integer Vulnerabilities**

| **Hardening Level** | **Product/Method** |
|---|---|
| Code: | Use of functions detecting integer overflow/underflow, migration to unsigned integers, ensuring integer data size in assignments/casts |
| Software Process: | Compiler option to convert arithmetic operation to error condition-detecting |

**Table 3: Hardening for Memory Management Vulnerabilities**

| **Hardening Level** | **Product/Method** |
|---|---|
| Code: | NULL assignment on freeing and initialization, error handling on allocation, pointer initialization, avoid null dereferencing |
| Software Process: | Using aspects to inject error handling and assignments, compiler option to force detection of multiple-free errors |
| Operating Environment: | Use a hardened memory manager (e.g. dmalloc, phkmalloc) |

4. *File management errors* can lead to many security vulnerabilities such as data disclosure, data corruption, code injection and denial of service. Unsafe temporary files and improper file creation access control flags are two major sources of vulnerabilities in file management.

In some cases, we can redesign the application to use inter-process communication instead of temporary files. The security hardening solutions for such problems are summarized in Table 4.

**Table 4: Hardening for File Management Vulnerabilities**

| **Hardening Level** | **Product/Method** |
|---|---|
| Code: | Use proper temporary file functions, default use of restrictive file permissions, setting a restrictive file creation mask, use of ISO/IEC TR 24731 functions |
| Software Process: | Set a wrapper program changing file creation mask |
| Design: | Redesign to avoid temporary files |
| Operating Environment: | Restricting access rights to relevant directories |

### Learning More

We introduced the concept of software security hardening and a classification for hardening methods. It is hoped that it will guide developers and maintainers in deploying and hardening security features and to remedy vulnerabilities present in existing OSS. More high quality information is available on security vulnerabilities and on the techniques used to mitigate them. We recommend some key resources to address security concerns in existing software, including the US Department of Homeland Security portal that is the most comprehensive reference for software security issues.

As a general advice, the scientific community recommends to look for OSS implemented in modern languages such as Java, C# .NET, Ada, SPARK, and CAML. These offer much better security than old programming languages like C and C++ that are deficient in terms of type safety and rigorous memory management. In all cases, well-recognized and well-supported implementations provide better building blocks since they are constantly improved to match the ever increasing risk encountered in the modern cyber environment.

**Acknowledgments**

This research is the result of a fruitful collaboration between the computer security Laboratory of Concordia University, Defense Research and Development Canada at Valcartier, and Bell Canada thanks to a grant under the NSERC/DND Research Partnership Program.

*This article is based on work originally presented at the 2006 International Conference on Privacy, Security and Trust which was hosted by the University of Ontario Institute of Technology.*

**Recommended Resources**

Secure Coding in C and C++
http://www.cert.org/books/secure-coding/

Secure Programming for Linux and Unix HOWTO
http://www.dwheeler.com/secure-programs/

DHS Software Security Portal
https://buildsecurityin.us-cert.gov/daisy/bsi/home.html

Common Vulnerabilities and Exposures
http://www.cve.mitre.org

Common Attack Patterns
http://capec.mitre.org

Environnemental Issues
http://nob.cs.ucdavis.edu/bishop/secprog/sans2002/index.html

Web Application Security
http://www.webappsec.org

*Robert Charpentier completed his degree in engineering physics at l'Ecole Polytechnique de Montreal in 1979. After working at CAE Electronics on flight simulators, he joined Defence Research Establishment Valcartier, where he specialized in infrared imagery and space-based surveillance. His current research domain is software security design and attack resistance of information systems operated in hostile environment. He has been deeply involved in F/LOSS studies since 2003.*

*Mourad Debbabi is full professor and acting director at the Concordia Institute for Information Systems Engineering. He is Concordia University Research Chair Tier I and Specification Lead for four Java Specification Standards. He received his Ph.D. in Computer Science from Paris XI Orsay University and worked as senior scientist for PINTL Laboratory and General Electric Corporate Research before joining Concordia University in Montréal.*

*Azzam Mourad is a PhD Student at Concordia University. His research and development thesis is on the automation of security hardening of open source software based on the Aspect-oriented programming paradigm.*

*Marc-André Laverdière did his MScA at Concordia University on security design applicable to open source software. He currently works in India.*

*"There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies."*

Professor C. A. R. Hoare

Developing reliable and secure software has become a challenging task, mainly because of the unmanageable complexity of the software systems we build today. Software flaws have many causes, but our observations show that they mostly come from two broad sources: i) design, such as a malicious or unintentional backdoor; and ii) implementation, such as a buffer overflow (http://en.wikipedia.org/wiki/Buffer_overflow).

To address these problems, our research group at Defence Research and Development Canada (DRDC) Valcartier first worked on design issues. A prototype of a UML (http://en.wikipedia.org/wiki/Unified_Modeling_Language) design verifier was built. Our approach was successful, but we faced two difficulties: i) specifying interesting security properties at the design level; and ii) scalability of the verification process.

Building on this experience, we studied design patterns for the implementation of security mechanisms. The output was a security design pattern catalog, available from the authors, which can help software architects choose mature and proven designs instead of constantly trying to reinvent the wheel.

This paper addresses the implementation issues from our evaluation of currently available automatic source code verifiers that search for program sanity and security bugs.

From this evaluation, it becomes clear that the choice of programming language to use when starting an open source project can have many important consequences on security, maintainability, reliability, speed of development, and collaboration. As a corollary, software quality is largely dependent on the adequacy of the programming language with respect to the desired properties of the system developed. Therefore, the adoption of open source software (OSS) should consider the programming language that was used.

## Context & Terminology

The assurance level required for executing applications depends on their execution context. Our context is military, in which confidential data is processed by sensitive applications running on widespread operating systems, such as Windows and Linux, and mostly programmed in C/C++ and Java. Our primary goal was to get rid of common security problems using automated source code verification tools for C++ and Java. To do so, we first investigated errors and vulnerabilities emerging from software defects. This allowed us to create meaningful tests in order to evaluate the detection performance and usability of these tools.

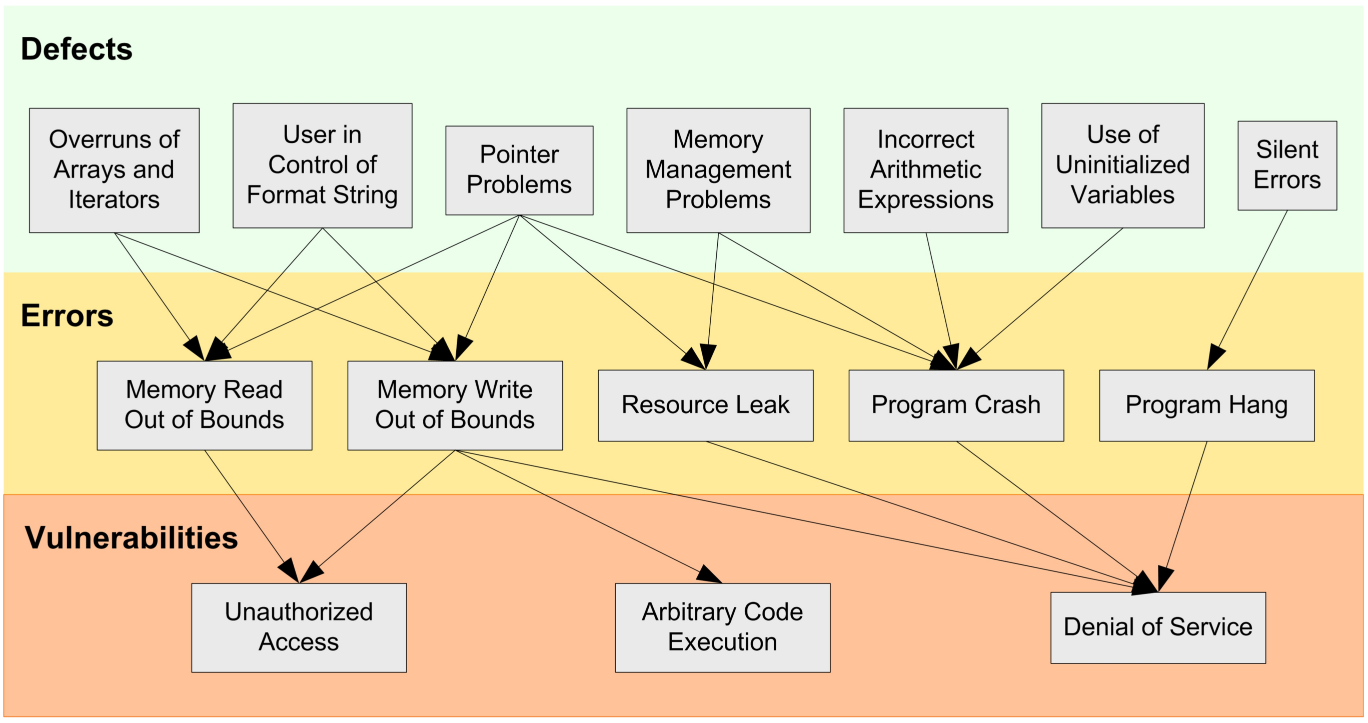
In our investigation of common software security problems, we observed that most do not come from the failure of security mechanisms. Rather, they occur from failures at a lower level, which we call program sanity problems. Security mechanisms ensure high level properties, such as confidentiality, integrity, and availability, and are mostly related to design. Access control frameworks, intrusion prevention systems, and firewalls are all examples of security mechanisms. Program sanity problems are related to protected memory, valid control and data flow, and correct management of resources like memory, files, and network connections.

Because these problems are many-sorted, a terminology is necessary to classify them. An error is closely related to the execution of a program and occurs when the behavior of a program diverges from "what it should be"; that is, from its specification. A defect lies in the code and is a set of program instructions that causes an error. A defect can also be the lack of something, such as the lack of data validation. Finally, a vulnerability is a defect that causes an error that can be voluntarily triggered by a malicious user to corrupt program execution.

We focused on errors, defects, and vulnerabilities that can have an impact on security. To be as general as possible, we wanted them to be application-independent. We defined five errors, twenty-five kinds of defects across six categories, and three vulnerabilities, as shown in Figure 1.

The list of possible low-level errors that can happen when a program is executed is very long. Since we had no interest in the correctness of computations with respect to specifications, we focused on general errors that can interfere with correct memory management, control flow, and resource allocation.

**Figure 1: Errors, Defects, and Vulnerabilities**


Defects
Overruns of Arrays and Iterators
User in Control of Format String
Pointer Problems
Memory Management Problems
Incorrect Arithmetic Expressions
Use of Uninitialized Variables
Silent Errors
Errors
Memory Read Out of Bounds
Memory Write Out of Bounds
Resource Leak
Program Crash
Program Hang
Vulnerabilities
Unauthorized Access
Arbitrary Code Execution
Denial of Service

Types of low-level errors include:

- memory write out of bounds: a valid region of memory is overwritten which results in serious vulnerabilities since it can allow an attacker to modify the program state

- memory read out of bounds: a region of invalid memory is read, resulting mostly in errors in computations, but sensitive values could be read

- resource leak: a discardable resource such as memory, a file handle, or a network connection is not returned to the available pool which will generally lead to a slowdown or crash of the resource-starved program

- program hang: the program is in an infinite loop or wait state, which generally leads to a denial of service

- program crash: an unrecoverable error condition happens and the execution of the program is stopped, leading to a denial of service

Most defects will not always generate errors for every execution of the program. Complex conditions have to be met for the error to happen and input values play an important role. Furthermore, many defects are composite and cannot be attributed to only one program instruction. The following is a list of the type of defects we used to create our tests:

- memory management faults: problems related to memory allocation, deallocation, and copy from one buffer to another

- overrun and underrun faults: problems related to the overrun or underrun of an array or a C++ iterator

- pointer faults: problems related to incorrect pointer usage

- cast faults: problems related to the incorrect cast of one type into another

- miscellaneous faults: problems that do not fit into any other category

Errors in general are undesirable, but the real problem is vulnerabilities, especially remotely-exploitable ones. We observed that almost all dangerous vulnerabilities are associated with memory reads or writes out of bounds. Vulnerabilities can be classified as:

- denial of service: allows an attacker to prevent user access to an appropriate service

- unauthorized access: allows an attacker to access functionalities or data without the required authorization

- arbitrary code execution: allows an attacker to take control of a process by redirecting its execution to a given instruction

**Problems with C/C++ Programs**

Many defects and errors are possible because of bad design choices when the C and C++ programming languages were created. These languages require micro-management of the program's behaviour through memory management, are error-prone due to pointer arithmetic, and induce serious consequences to seemingly benign errors such as buffer overflows. A short list of the major C/C++ design shortcomings follows:

- lack of type safety: type-safe programs are fail-fast as their execution is stopped immediately when an error occurs whereas non type-safe languages like C/C++ let the execution of erratic programs continue

- static buffers: buffers in C/C++ cannot grow to accommodate data, buffer accesses are not checked for bounds, and overflows can overwrite memory

- lack of robust string type: C has no native type for character strings, meaning static buffers with potential overflow problems are used instead; while C++ programs can use a string type, our observations show that this is rarely the case

Creators of modern languages, such as Java, had these problems in mind and addressed them. Indeed, Java is immune to C/C++ program sanity problems because runtime checks throw an exception if an error occurs. However, many program sanity checks throw unchecked exceptions and these are rarely caught by programmers. Many problems become denial-of-service vulnerabilities since uncaught exceptions crash the program.

**Tools Evaluation**

We evaluated 27 tools for C/C++ and 37 for Java. All these tools were categorized into three families: i) program conformance checkers; ii) runtime testers; and iii) advanced static analyzers.

*Program conformance checkers* perform a lightweight analysis based on syntax to find common defects. Because of this unsophisticated analysis, they perform poorly, except for a few defects that can be detected by simple syntax analysis. Many free tools were in this category.

*Runtime testers* look for errors while the program is running by instrumenting the code with various checks. This provides a fine-grained analysis with excellent scalability that can be very helpful when the program's behaviour cannot be computed statically because of values that are not known before runtime.

*Advanced static analyzers* work on program semantics instead of syntax. They generally use formal methods, such as abstract interpretation or model-checking, which often lead to scalability problems. The code must be compiled into a model and this is usually complex with C/C++ because of code portability problems between compilers.

Our results can be summarized as:

- for C/C++, commercial tools are by far the best

- for Java, there are many good free tools

- since Java is immune to most program sanity problems that plague C/C++, there are no exact equivalents to C/C++ tools

- the focus of Java tools is on good practices and high level design problems

Since our goal was to detect program sanity problems, we focused on tools for C/C++ during our evaluation. For our evaluation, our criteria were: i) precision in flaws detected vs. false positives; ii) scalability from small to large programs; iii) coverage or the inspection of every possible execution; iv) and the quality of the diagnostic report in its usefulness for problem correction.

Preliminary tests showed that only three tools for C/C++ had the potential to help us achieve our goal: Coverity Prevent (http://www.coverity.com) and PolySpace for C++ (http://www.mathworks.com) for detecting defects, and Parasoft Insure++ (http://www.parasoft.com) for detecting errors. We tested these tools in two ways: i) over real code in production that, to the best of our knowledge, worked well but was a bit buggy; and ii) over many small ad-hoc pieces of code (synthetic tests) containing specific programming defects.

To compare these tools, all results had to be converted to errors or defects. For synthetic tests, defects and the errors they caused were known in advance so it was easy to convert everything to defects. However, for code in production, nothing was known in advance, so we decided to use the best result as a baseline. Since Insure++ was the best performer, all results were converted to errors.

**Results**

The complete results of our synthetic tests are available in the original paper. The difficulties in testing C/C++ programs can be summarized as follows:

- no tool is able to detect every kind of defect or error
- static analysis tools need good quality code to perform well
- pointer arithmetic used to read from and write to complex data structures renders static analysis extremely difficult
- makefiles are often show-stoppers due to their lack of granularity, their number makes debugging a tedious task, and they are often complex and require many dependencies
- compiler-specific extensions to C/C++ make the parsing of non-standard extensions difficult
- the use of conditional compilation using preprocessor directives which come from a mix of environment variables, configuration files, and make parameters adds to the complexity of the verification process
- header files are often created or moved by the makefile while it is running
- there are often many different header files with the same name, but at different locations

We found that having the verification tool parse the program correctly is the most difficult part of the job, and this is often a show-stopper unless one has unlimited time. Java is not problematic because it has no preprocessor and no conditional compilation. It has been designed to be standard and portable.

**Tool Limitations and Best Usage Scenario**

We found that current static verification tools suffer from what we have called the “black box problem”. Indeed, for reactive applications and heterogeneous systems, execution does not always take place in available application code. For instance, in reaction to a mouse click, a reactive application can start executing in kernel code to pass the event over and around the operating system. This part of its execution can rarely be analyzed and, therefore, static analysis tools can hardly determine what type of data comes out of these calls. This prevents true interprocedural analysis.

Scalability is also a problem for static tools that have to consider (and abstract) all possible executions.

Dynamic tools have the opposite problem: they are very scalable but provide poor coverage with poor test cases. However, if you consider the number of tests needed to cover all possible executions with dynamic tools, scalability is still a problem.

The best usage scenario for Coverity Prevent is when the whole application needs to be analyzed and it is compiled using a working makefile. The application code size can be over 500K lines of C++ without problems.

Coverity has many good points: i) very good integration with makefiles; ii) uses the Edison compiler front-end that can read code containing compiler-specific extensions from almost every big compiler in the industry; iii) very scalable; iv) excellent diagnostics with execution traces that are easy to understand and very helpful to correct problems; and iv) uses an innovative, but proprietary, analysis based on statistical code analysis and heuristics.

The best usage scenario for PolySpace for C++ is to analyze small segments of critical code in applications where runtime exceptions should never happen. The application code size must stay under 20K lines of C++. It uses a very thorough analysis based on abstract interpretation, with which it can detect runtime errors statically. It has a nice graphical interface, especially the viewer module which is used to analyze the report and navigate the source code. However, it lacks a good diagnostic because sometimes it is impossible to understand the defect found.

The best usage scenario for Parasoft Insure++ is to test hybrid systems based on many heterogeneous components. To consider code coverage, it should always be integrated into test case harnesses that have been shown to provide good code coverage. Since Insure++ is a dynamic tool, there is no limit to the application code size and bad quality code has no effect on detection performance. Insure++ has a very good diagnostic with call stack and memory diagrams that show exactly what was overwritten. However, as already mentioned, test cases have to be carefully specified with a good coverage strategy.

**Discussion**

We have alluded to the importance of simple and unambiguously specified language constructs, standardized, portable, and type-checked language compilation, vigilant runtime monitoring, and available verification tools. We argue that it is simpler, though not simple, to produce better quality software with modern programming languages. We believe that modern programming languages should always be used over older ones, except when a convincing argument can be made against it.

Furthermore, programmers should use the verification tools that are available for their programming languages and should stay aware of the new ones. In the selection of open source products, the programming language used is, of course, not the only variable to consider in assessing software quality. But when evaluating two products that have been properly tested for appropriate and correct functionality for the task at hand, we would recommend to choose the one programmed with a modern language.

The computer industry tends to adopt new technologies very quickly. Setting human and financial resources aside, the adoption of new programming languages generally follows the laws of fashion: what is the new best thing this year? what should I use to stay cool and up-to-date? This is not necessarily a bad driver of progress. However, it covers a pernicious habit: we have rarely observed a programmer adopting a new programming language because he knew all the pitfalls of his current language and wanted to avoid them.

## Conclusion

The root of security problems are not the failure of security mechanisms. C/C++ programs are especially problematic because they enforce almost no restriction on the execution of programs and they are prone to vulnerabilities with serious consequences. However, modern languages, such as Java, are immune to C/C++ problems and are not prone to any serious vulnerability. Of course, just as with any language, design must be rigorously verified and implemented correctly. The use of Java is not a panacea and care should still be taken in the correct implementation of security mechanisms.

Verifying C/C++ programs is a huge challenge. These languages are very difficult to analyze because of many undefined or non-standard semantics, pointer arithmetic, and compiler-specific extensions to the language. We have found no currently available verification tool that can reduce the risk significantly enough for sensitive applications. We highly recommend the use of modern programming languages such as Java, which nullify program sanity problems. However, if the use of C/C++ is mandatory, we recommend restricting its usage and the use of serious test cases and verification tools.

*This article is based upon a paper originally published in the Proceedings of the Static Analysis Summit (http://samate.nist.gov/docs/NIST_Special_Publication_500-262.pdf)*

*Frederic Michaud is a researcher specialized in software security including verification and validation, defensive programming, and robust architectures for information systems operated in hostile environments.*

*Frederic Painchaud is a defence scientist at Defence Research and Development Canada, Valcartier. His research interests are language semantics, formal methods, program analysis, and IT security.*

*"Open, distributed innovation is attacking a major structure of the social division of labor. Many firms and industries must make fundamental changes to long-held business models in order to adapt."*
Eric Von Hippel, MIT

On May 23, 2008, Stoyan Tanev from Carleton University delivered a presentation entitled "Trends in Technology Marketing". This section provides the key messages from Dr. Tanev's lecture. Tanev's lecture discussed current trends in technology innovation and marketing by focusing on the evolution of traditional marketing concepts. The slides from the presentation are available for download (http://www.talentfirstnetwork.org/wiki/images/d/db/Trends_in_technology_marketing_May_23.pdf).

The TIM Lecture Series provides a forum that promotes the exchange of knowledge between university research and technology company executives and entrepreneurs. Readers outside the Ottawa area who are unable to attend the lectures in person are invited to view upcoming lectures in the series either through voice conferencing or webcast. Instructions for joining a lecture are available at http://tinyurl.com/5nhbc8.

**Core Marketing Concepts**

The first half of the lecture concentrated on current trends in technology innovation and marketing and described the evolution of traditional marketing concepts. Marketing strategy is changing in response to higher customer information access and increased global connectivity.

Current marketing strategies and their associated business model implications are a result of the emerging paradigm of value co-creation networks. The design of offerings based on value co-creation is considered as the next practice in value creation.

Aspects of the evolving marketing trends include:

- product support and company reputation are important for customers making buying decisions
- product leadership and customer intimacy are important for crossing the chasm (http://en.wikipedia.org/wiki/Crossing_the_Chasm)
- the ambiguity in technology markets requires different marketing strategies
- advertising should not communicate complex messages
- a strategy that works in one stage of the product life cycle may not work in another stage; using the same strategy may result in failure
- value chains can be parallel as well as serial

**Co-Creation Paradigm**

The second half of the lecture further explored the value that can be created through co-creation with one's user base. As in any value proposition, how to appropriate value isn't obvious. While technology plays a critical role in value creation, value is created not through a product's features but by its ability to get a job done. It is difficult for some marketers to grasp the new reality that value creation is made possible by cooperating with complementors and is enabled by transparency. It should also be noted that some vertical value chains have many layers and how you differentiate depends on the complexity of the value chain.

Traditional marketing techniques focus on planning whereas co-creation allows others to take advantage of opportunities.

In other words, the new marketing trend is all about creating a path to your product via pull methods instead of push. This section also offered insights into customers:

- customer satisfaction is multidimensional and based on experience rather than price
- customer involvement builds trust in your brand
- marketing can't satisfy a customer's need (promise vs. product)

There was some discussion regarding the phenomena that user experience does not necessarily mean co-creation. An example is a user that merely consumes Expedia's services as opposed to leaving comments--which creates value--for others to use. Even though there isn't direct value creation from a passive consumer, these types of consumers also provide value in that they represent traffic and exposure to one's services.

A final key message is that co-opetition (http://en.wikipedia.org/wiki/Co-opetition) can offer end-to-end services, something that is rarely achievable by one company.

*Stoyan Tanev is an Assistant Professor in the Department of Systems and Computer Engineering at Carleton University. He received a joint Ph.D. from the University of Sofia and the Université Pierre and Marie Curie. His research interests include open source innovation strategies in non-software sectors, management of innovation in new, emerging and cross-disciplinary technology areas, and biomedical optics and nanophotonics design and simulation tools.*

**Recommended Reading**

Marketing in the 21st century
http://myphliputil.pearsoncmg.com/student/kotler6/REV01.PDF

Democratizing Innovation
http://web.mit.edu/evhippel/www/democ1.htm

*"...communities are living organisms that are most analogous to gardens; they must be tended to, cultivated, and fertilized if they are to take root, grow and thrive."*

Mark Sigal
http://thenetworkgarden.com/weblog/2008/03/online-communit.html

On June 4, 2008, Ian Skerrett from the Eclipse Foundation delivered a presentation entitled " Building Technical Communities". This section provides the key mssages from Ian's lecture. Ian used his observations of working in the Eclipse community to explain why community building is important, its critical elements, and how the traditional roles within an organization relate. The slides from the presentation are available for download (http://www.talentfirstnetwork.org/wiki/images/6/69/Building_technical_communities_June_4.pdf).

**Community Defined**

A community refers to people who share a common interest or passion and interact with each other about the given passion, regardless of their geographic location. Participation in a community can be compelling and sticky in that people return frequently and remain for extended periods. A community is important to your company because, it:

• provides closer contact with your customers and users

• facilitates the development and delivery of a whole product or solution

• supplements technical support

• provides word of mouth marketing

• accelerates technology adoption

• enables a small number of individuals to have significant impact worldwide

In a technical community: i) peers, not vendors, determine the message; ii) developers talk to other developers, not through intermediaries or press releases, and marketers produce content such as case studies that help developers sell up to their managers; iii) people speak to people, not a market or a demographic attribute; iv) employees interact with people who are saying good and bad things about their companies; iv) interactions first build trust and then build value; and iv) you learn to live with your competitors being part of the same community.

It is a myth that committers are volunteers. The committers for the established open source projects are nearly all paid by companies to commit code to open source projects. Non-monetary motives for an individual to contribute to a technical community include:

• satisfy a passion for doing something interesting while receiving immediate feedback

• satisfaction from seeing individual's code being used and talked about

• satisfaction from being able to fix code immediately

• self branding that leads to consulting work and/or increase in the number and quality of job opportunities

To successfully interact with a technical community, you should: i) be authentic; and ii) respond and react respectfully, accurately and quickly. Other insights from this portion of the lecture include:

• customers and venture capitalists wish to know how healthy a technical community is using metrics that include size, diversification and talent

• metrics, beyond counting numbers of downloads, are needed to assess the health of a community

• monetizing a community is a delicate skill

• venture capital firms see a community as a mechanism that lowers their risks as well as lowers their ventures' sales and marketing costs

• old marketing is either broken or changing and new marketing is community-based

• top down marketing is still needed because technical people must sell up to their managers

• old school marketers usually don’t have the technical skills required to join a community and gain the interest of developers without insulting them or being insulted by them

## Building Communities

In order to build a community, you must produce good code that solves a compelling problem and/or decreases development costs. You must also ensure that the conversation about the code is worthwhile. Make it easy to contribute to the community by providing: i) documentation that lowers the barrier to understanding the code; ii) tutorials, white papers and books; and iii) experts who monitor newsgroups and bug databases to provide prompt and accurate feedback.

Community building also requires transparency. Signs of transparency include: i) maintaining open bug databases; ii) publishing project meetings; and iii) publishing project plans and incorporating community's feedback into plans. Be part of the community, do not attempt to control it. Provide a governance structure that fits with the aims of the community.

An architecture of participation (http://www.oreillynet.com/pub/wlg/3017) includes low barriers to entry for newcomers and some mechanism for isolating the cathedral from the bazaar (http://en.wikipedia.org/wiki/The_Cathedral_and_the_Bazaar). An architecture of participation allows for a free market of ideas, in which anyone can put forward a proposed solution to a problem. From experience, an architecture of participation that is comprised of a very large run-time system as the platform (cathedral) with plug-ins on top (bazaar) does not work.

A good architecture of participation supports a small cathedral which enables a bazaar where it is easy for individuals to add their ideas to the platform. For example, Eclipse is a platform that includes a small run-time system which enables add-ons and other components to run on top of the platform.

It is important that providers of new add-ons are on the same footing as those who provided the original system. A successful architecture of participation:

- empowers individuals and small groups to make decisions
- provides open APIs and commercial friendly licenses
- enables suppliers to compete on implementations and the users, not the platform, to decide who wins
- is easy to integrate and extend
- spurs innovation
- seeds a broader ecosystem
- promotes a culture of openness, transparency and meritocracy

Some successful communities have strong, visible technical leaders. Examples include Linus Torvalds for the Linux community, David Heinemeier Hansson (DHH) for Rails, and Mark Shuttleworth for Ubuntu. Other successful communities have various community leaders. In these communities, the quality of the code is more important than the visibility of the leaders.

To overcome challenges found in communities, it is often better to first obtain feedback and then make an executive decision. Communities fail because:

- they don't produce good code and/or the conversation about the code is not worthwhile
- there are no visible leaders and code is of poor quality
- no or little effort is invested into nurturing the community

**Eclipse**

Ian finished the presentation with a quick overview of the Eclipse Foundation (http://www.eclipse.org) which has 180+ members globally. The approximate breakdown is: US (50%), Europe (30%) and the rest of the world (20%). Eclipse's main marketing objective is to grow the community and spread the adoption of Eclipse into vertical enterprise markets. Eclipse does not compete with member companies' products. IBM open sourced Eclipse when it recognized that nobody wants to build modules for a locked-in system.

*Ian Skerrett is the Director of Marketing at the Eclipse Foundation, a not-for-profit corporation supporting the Eclipse open source community and commercial ecosystem. He is responsible for implementing programs that raise awareness of the Eclipse open source project and grow the overall Eclipse community. Ian has been working in the software industry for over 20 years. He has held a variety of product management and product marketing positions with Cognos, Object Technology International, IBM, Entrust and Klocwork. He graduated from Carleton University with a Bachelor of Computer Science and has an MBA from McGill.*

**Recommended Reading**

Brand Hijack: Marketing Without Marketing
http://www.amazon.com/Brand-Hijack-Marketing-Without/dp/1591840783

Developing an Architecture of Participation
http://project.bazaar.org/wp-content/stall_project_uploads//2007/09/111_final_paper.pdf

Does Code Architecture Mitigate Free Riding in the Open Source Development Model?
http://www.people.hbs.edu/cbaldwin/DR2/BaldwinArchPartAll.pdf

Having had a few more hours to think about Industry Minister Jim Prentice's Canadian DMCA (www2.parl.gc.ca/HousePublications/Publication.aspx?Docid=3570473&file=4), I am left with one dominant feeling--betrayal. I have already highlighted the key provisions (http://www.michaelgeist.ca/content/view/3025/125/) and coverage (and note that it will take some time to fully assess the implications of this bill) but it is immediately apparent that the concerns of thousands of Canadians--now over 45,000 on the Fair Copyright for Canada Facebook group alone--have been realized. If enacted, the Canadian DMCA would strongly encourage the use of technological locks and lawsuits. While Prentice has given a handful of new rights to Canadian consumers, each is subject to many limitations and undermined by the digital locks provisions that may effectively render the new rights meaningless.

**So Why is it a Betrayal?**

Because in a country whose Supreme Court of Canada has emphasized the importance of balance between creators rights and user rights, the Canadian DMCA eviscerates user rights in the digital environment by virtually eliminating fair dealing. Under this bill, the right to copy for the purposes of research, private study, criticism, and news reporting virtually disappears if the underlying content is digitally locked.

Because in a country that rightly promotes the importance of education, the Canadian DMCA erects new barriers for teachers, students, and schools at every level who now face the prospect of infringement claims if they want to teach using digital media.

Because in a country that prioritizes privacy, the Canadian DMCA will render it virtually impossible to protect against the invasion of privacy by digital media companies. The bill includes an exemption for those that circumvent digital locks to protect their privacy, yet renders the tools needed to circumvent illegal. In other words, the bill gives Canadians the right to protect their privacy but prohibits the tools needed to do so.

Because in a country that values consumer rights, the Canadian DMCA means that consumers no longer control their own personal property. That CD or DVD or e-book or cellphone you just bought? The bill says you now have the right to engage in "private use copying" but not if it contains digital locks.

Because the Conservative Party of Canada promised to Stand Up for Canada, yet the Canadian DMCA is quite clearly U.S.-inspired legislation, the result of intense pressure from U.S. officials and lobby groups.

Because the government pledged to table treaties for House of Commons debate before introducing implementing legislation and failed to do so. Claims that this legislation does not ratify the treaties violates the spirit of that commitment.

Because ratification of the World Intellectual Property Organization's Internet Treaties can be accomplished in a far more balanced manner.

Because countries such as New Zealand and Israel have demonstrated that there is no need to blindly follow U.S. demands on the copyright file.

Because the interests of individual Canadians--including those calling for more flexible fair dealing--is completely ignored.

Because the Canadian DMCA was introduced without consulting consumer groups, education groups, civil society groups, or the Canadian public.

Because Jim Prentice knows better. He saw first-hand the passion of Canadians calling for balanced copyright and has received thousands of calls and letters on the issue. Yet rather than genuinely working to craft a balanced solution, he opted to release a fatally flawed bill.

Despite all that, I still also harbour some optimism. The events of the past six months have demonstrated conclusively that Canadians care about balanced copyright even if the Industry Minister does not. Over the coming months, I firmly believe that we will see the fair copyright movement expand well beyond what has been just built. We will see Canadian musicians, songwriters, artists, and filmmakers speak out against this legislation. We will see companies of all sizes and all sectors speak out against this legislation. We will see the privacy groups, education groups, and consumer rights groups speak out against this legislation.

We will see the NDP speak out against this legislation. We will see the Liberals--who are already focusing on the lack of consultation and the prospect of a police state--ultimately identify their Bill C-60 as a better approach and speak out against this legislation. We will see Conservative MPs from coast to coast (including the Conservative candidate from the forthcoming Guelph by-election) wonder why their party has introduced a bill that runs counter to their own policies and (quietly) speak out against this legislation.

We will see thousands of Canadians speak out against this legislation again and again and again until it is changed.

The Canadian DMCA is a kick in the gut to Canadians everywhere. But I believe we will get back up and demand better. Start now:

1. Write to your MP, the Industry Minister, the Canadian Heritage Minister, and the Prime Minister. If you send an email, print it out and drop a copy in the mail. If you are looking for a sample letter, visit http://www.copyrightforcanadians.ca/action/firstlook/.

2. Take 30 minutes to meet directly with your MP. From late June through much of the summer, your MP will be back in your community. Every MP in the country should return to Ottawa in the fall having heard from their constituents on this issue.

3. If you are not a member of the Fair Copyright for Canada Facebook group (http://www.facebook.com/group.php?gid=6315846683), join. If you are, consider joining or starting a local chapter and be sure to educate your friends and colleagues about the issue.

*This article originally appeared as a blog entry (http://www.michaelgeist.ca/content/view/3029/125/). You can learn more about the Canadian DMCA and other copyright issues at Michael Geist's blog (http://www.michaelgeist.ca/).*

*Michael Geist is a law professor at the University of Ottawa where he holds the Canada Research Chair of Internet and E-commerce Law. He has obtained a Bachelor of Laws degree from Osgoode Hall Law School in Toronto, Master of Laws degrees from Cambridge University and Columbia Law School, and a Doctorate in Law from Columbia Law School. Dr. Geist serves on the Privacy Commissioner of Canada's Expert Advisory Board and maintains http://privacyinfo.ca, a leading privacy law resource.*

**Q. I can understand how small businesses and startups can benefit from the no-licensing costs associated with open source software (OSS). Can you provide an example of a business reason for using OSS other than the licensing cost of the software?**

**A.** Here is an example that occurred during a time when worm outbreaks were just starting to hit organizations hard. Some organizations were already receiving feedback as traffic caused by computer worms was causing networks to fail, and failing is the worst form of feedback.

At Bell Canada, our large internal network was running well. Being a carrier, the internal network was very well connected. Having many experts within the company to design and build the network, it was configured to withstand very heavy usage and to handle some types of failure modes gracefully. Considering the many worms of the time, the network did very well at isolating the mischievous traffic to the periphery of the network.

Bell has systems to monitor many different aspects of the network and its attached computers. With such a robust network and high degree of network isolation, however, there was not enough information to track worm propagation in detail. At least not in the detail expected by the executives who wished to identify and track the extent of worm propagation and incident management effectiveness using metrics and reporting that business people could understand.

There were vendors who had monitoring solutions which could provide the metrics, but they would be expensive. In a large organization like Bell Canada, the expense itself was not an issue and the procurement process started almost immediately.

In a large organization, however, the procurement process does not get completed overnight. The investment needed for the additional monitoring was going to be significant as this large network has tens of thousands of nodes and hundreds of physical locations. Accordingly, procedures had to be followed, including the need for technology studies for specifications, assessments, a proper bidding process, architecture, design and operational reviews.

Enter open source software (OSS), an old computer, a motivated systems administrator, and some teamwork. A plan was developed to build a darknet (http://www.team-cymru.org/Services/darknets.html). The Bell incident response process had already been initiated due to the industry recognition of this particular worm outbreak, even though Bell had no indications of problems internally. A subcommittee was established, network configuration changes were made, operational processes outlined, and a single server was configured with some OSS.

Within a day, statistically significant reporting was established on a near real time basis which met the immediate incident management metric and reporting needs. The executives and incident management groups were confident that the magnitude of impact was now understood, and that the effectiveness of the remediation efforts could be tracked.

The darknet was successful at the general reporting required, and actually became the primary source of detailed information needed for the remediation team to identify and correct the individual network nodes. Further, the darknet's detailed reporting produced the information needed to identify that, in some instances, the vendor provided patch was unsuccessful in removing the vulnerabilities.

For some situations Bell has significant influence, so when Bell discovers a failed patch it may actually expedite a proper solution for a much larger community.

The OSS had the functionality necessary to meet 90% of the immediate goals "out-of-the-box". The open logging formats and well documented utilities allowed for quick adaptation to the business need. The secure-by-default, transparent and understandable configuration, and ecological diversity from the main devices being monitored, gave confidence that the system would not be affected by the malware traffic it was monitoring.

For the technical reader, the tracking server was setup using an OpenBSD (http://www.openbsd.org) server for collecting network information, native tcpdump and Perl for extracting and reporting on logs, native syslog's ability to launch programs and nbtstat to collect near real time information about hostnames and userids. O'Reilly's Perl Cookbook (http://www.oreilly.com/catalog/9780596003135/) facilitated the creation of much of the glue. These tools, plus the co-ordination with the network operations group, were all that was needed to setup the darknet, a blackhole where packets go in but nothing leaves but information.

*Alan Morewood has been involved with open source since discovering Linux in 1992. In 1993, he started as a systems administrator for Bell Sygma, bringing the standard GNU tools to the attention of the sysadmins familiar only with expensive commercial tools. While at Bell Sygma, he learned about security by managing the main corporate Internet gateway and establishing the first bell.ca platform. Bell Canada's corporate security department solicited his participation in 1996 where he continues today by coaching employees in the relationship between networks, system administration, security, and business needs. Alan has a B.A.Sc in Systems Design Engineering from Waterloo, a P.Eng, and is CISSP certified.*

The goal of the Talent First Network Proof of Principle (TFN-POP) is to establish an ecosystem anchored around the commercialization of open source technology developed at academic institutions in Ontario.

The priority areas are the commercialization of open source in:

- Mapping and geospatial applications
- Simulation, modeling, games, and animation
- Conferencing
- Publishing and archiving
- Open educational resources
- Social innovation
- Business intelligence
- Ecosystem management
- Requirements management

**Expected Results**

The TFN-POP is expected to:

- Establish a healthy ecosystem anchored around the commercialization of open source assets
- Maximize the benefits of the investment in the Talent First Network by the Ministry of Research and Innovation
- Accelerate the growth of businesses in Ontario that use open source assets to compete

**Eligibility to Receive Funds**

Individuals eligible to receive funds are faculty, staff, and students of universities and colleges in Ontario.

**Budget and Size of Grants**

A total of $300,000 is available. Applicants' requests should not exceed $30,000.

The TFN-POP may provide up to 50 percent of total project costs.

**Criteria**

Proposals will be judged against the following five criteria:

- Strength and novelty of open source technology proposed
- Extent of market advantage due to open source
- Project deliverables, likelihood that the proposed activities will lead to deliverable completion on time, and effectiveness of the plan to manage the project
- Track record and potential of applicants
- Extent of support from private sector

**Application**

The electronic version of the application received by email at the following address: TFNCompetition@sce.carleton.ca will be accepted as the official application. The email must contain three documents: a letter of support, project's vitals, and a project proposal.

**Letter of support**: (maximum 2 pages) a letter, signed by the person responsible for the Technology Transfer Office or Applied Research Office of the academic institution that proposes to host the project and the faculty member or student who will lead the project, must be included. This letter should describe the nature of the support for the project from the academic institutions, companies and other external organizations.

**Project's vitals**: (maximum 1 page) The project's vitals must include:

- Person responsible for applied research or technology transfer at the college submitting the proposal: name, mailing address, telephone number, and email address
- Project leader: name, mailing address, telephone number, and email address
- Team members: names, mailing addresses, telephone numbers, and email addresses
- Budget: Total budget, with TFN's contribution and that of other organizations
- TFN investment: TFN contribution broken down by payments to students, payments to faculty, and payments to project awareness activities

**Project proposal:** (maximum 5 pages) Project proposal must include the following:

- Benefits: (maximum 1/2 page) Description of the benefits of the proposed project, and an overview of the context within which the project is positioned
- Advantage: (1/2 page) Market advantage provided by open source assets used in the project
- Information on applicants: (maximum 1.5 pages) Background information to help assess the track record and potential of the people who are key to the project and the college
- Project plan: (maximum 2.5 pages) Description of the deliverables (what will be delivered and when); key project activities; nature of the involvement from companies, and other external organizations; and plan to manage the project

**Evaluation & Deadline**

Proposals will undergo review by the Expert Panel established by the TFN-POP. The Chair of the Panel may contact the applicants if required. A final decision will be communicated to the applicants within 30 days after the email with the official application is received.

There is no deadline. Applications will be evaluated on a first-come basis until the $300,000 available is committed.

**Contacts**

Luc Lalande: Luc_Lalande@carleton.ca

Rowland Few: rfew@sce.carleton.ca

**About the Talent First Network**

*The Talent First Network (TFN) is an Ontario-wide, industry driven initiative launched in July 2006 with the support of the Ministry of Research and Innovation and Carleton University. The objective is to transfer to Ontario companies and Open source communities: (i) Open source technology, (ii) knowledge about competing in Open source environments and (iii) talented university and college students with the skills in the commercialization of Open source assets.*

**Evaluation of Ten Standard Setting Organizations with Regard to Open Standards**

**Copyright:** IDC

**From the Introduction:**

This document has been prepared by IDC for the Danish National IT and Telecom Agency (NITA). It describes a methodology to evaluate SSOs (standard setting organizations) with regard to the degree of openness of the organization and thereby the degree of openness in their deliverables. The NITAs assessment of degree of openness is part of an overall assessment of standards where, beside the aspect of openness, public value and market support are also assessed.

http://www.itst.dk/arkitektur-og-standarder/Standardisering/Aabnestandarder/baggrundsrapporter

---

**Undocumented Open Source Leaves a Gap in Your Application Security Strategy**

**Copyright:** Palamida

**From the Executive Overview:**

Application security is more susceptible than ever in today's dynamic application development landscape. Most applications, internal and external, developed within the last five years, include at least 30% open source (OSS) and third-party components. And by 2010, open source products will be well established in 75% or more of mainstream enterprises. While important to a company's bottom line, this increase in OSS usage presents a huge security challenge to organizations industry-wide. The root cause of many application security vulnerabilities lies in the application source code. The problem is that the sheer size of a code base coupled with the number of contributing developers makes it nearly impossible for companies to get an accurate assessment of their software assets, much less a clear understanding of the vulnerabilities associated with the adopted code.

http://www.palamida.com/themes/resources/Palamida_WhitePaper_ImportanceofAppSource.pdf

## Open Source Curriculum Expanded at Seneca

**May 21, Toronto, ON**

Red Hat, Inc. today announced that Seneca College will expand the use of open source software in its curriculum through the Fedora Project, a Red Hat sponsored and community-supported open source collaboration. Seneca College students in the School of Computer Studies will work within the Fedora Project while learning open source development and administration. As the principles and methodologies of open source are changing the software industry, Red Hat and Seneca College are committed to driving change through new models of computer science education.

http://lwn.net/Articles/283274/

---

## Launching of the CIPP Wiki

**June 2, Montreal, QC**

A great deal has been written recently about copyright reform in Canada. Canadians have all been reflecting on a variety of copyright-related issues, as these questions and controversies jump out at us in both the traditional and digital media. Now that the various constituents in this debate have put forward their best cases on various points for potential reform, we have decided at the CIPP to try to provide all participants to have a direct hand in creating a Copyright Act through a wiki platform. Help us draft a reformed Copyright Act by logging on the CIPP wiki page and giving us your suggested changes and providing good reasons. he wiki platform will be opened until July 15.

http://www.cipp.mcgill.ca/en/news/newsletter/172/

## Copyright Bill will Hurt Innovators

**June 3, Ottawa, ON**

A new coalition of Canadian software businesses and supporters is concerned about how reforms to Canadian copyright laws might affect the open source business model. The Canadian Software Innovation Alliance (CSIA) represents over 20 businesses that specialize in open source software. The CSIA is particularly concerned about potential changes to copyright law, such as making it illegal to tamper with technological protection measures. In everyday language the proposed legislation is similar to making the use and ownership of screw-drivers and pliers illegal because they can be used to commit crimes such as burglary. Similar laws in other countries, such as the Digital Millennium Copyright Act in the United States, have caused problems for consumers and businesses alike.

http://www.softwareinnovation.ca/wp-content/uploads/2008/06/csia_media_release-final-3june2008.pdf

**June 25**

Symposia On Eclipse Open Source Software

**Ottawa, ON**

Eclipse and OMG are jointly organising symposia to promote and build on the partnership between Eclipse's open source software and OMG's open standards during the OMG Technical Meeting in Ottawa. The symposia is a unique opportunity to participate in shaping the joint future of the Eclipse Open Source community and the OMG Open Standards community. Please join us for a day of stimulating technical planning and discussion.

http://www.omg.org/news/meetings/eclipse-omg-2008/index.htm

---

**June 25-27**

Open Scholarship

**Toronto, ON**

The objective of the conference is to bring together researchers, lecturers, librarians, developers, business executive, entrepreneurs, managers, users and all those interested in issues regarding electronic publishing in widely differing contexts. This year's presentations include the topic of Open Access.

http://www.elpub.net/

**July 15-16**

2008 Wireless & Mobile Expo and Conference

**Toronto, ON**

The 2008 Wireless & Mobile Expo and Conference and Expo is a value-packed IT conference and exhibition that will educate senior executives, IT managers, program managers and technical specialists on the future adoption and integration of wireless enterprise architecture, mobile networks and user-defined products, in demand today. It serves to focus and align an organization’s IT investments with business goals and substantially improve its business performance and productivity.

http://wirelessandmobile.wowgao.com

---

**July 21-25**

GeoWeb 2008

**Vancouver, BC**

The GeoWeb 2008 conference reflects the breadth, evolution and growing maturity of the ability to locally/globally integrate and share geospatial information via the Internet. It is one of the only conferences focusing exclusively on the convergence of GIS and the Internet, and the economic potential associated with the convergence of XML, web services and GIS.

http://geowebconference.org

**July 23-26**

Linux Symposium

**Ottawa, ON**

The Linux Symposium is one of the foremost Linux and Open Source conferences in the world, bringing together software developers, industry professionals and enthusiasts to discuss the latest emerging technologies and ways to improve the functionality and integration of Linux and Open Source software. 2008 marks the 10th anniversary of the conference and we are very excited about celebrating the progress the Linux community has made in the past decade. The conference will also focus on looking towards the future of Linux and Open Source Development through groundbreaking presentations and discussions.

http://www.linuxsymposium.org/

---

**August 4-8**

Agile 2008

**Toronto, ON**

Agile 2008 presents the latest techniques, technologies from both a management and development perspective, for successful Agile software development. Agile 2008 puts attendees in contact with the latest thinking in the agile domain, bringing together executives, managers, software development practitioners and researchers from labs and academia. The conference is not about a single methodology or approach, but rather provides a forum for the exchange of information regarding all agile development technologies.

http://www.agile2008.org/

**August 10-14**

74th IFLA General Conference and Council

**Quebec, QC**

Open Source, distributed services delivery, web services and smart clients provide new paradigms for delivery of library services technology to small and special libraries. The theme of the 74th World Library and Information Congress is "Libraries without borders: navigating toward global understanding".

http://www.ifla.org/IV

The goal of the Open Source Business Resource is to provide quality and insightful content regarding the issues relevant to the development and commercialization of open source assets. We believe the best way to achieve this goal is through the contributions and feedback from experts within the business and open source communities.

OSBR readers are looking for practical ideas they can apply within their own organizations. They also appreciate a thorough exploration of the issues and emerging trends surrounding the business of open source. If you are considering contributing an article, start by asking yourself:

1. Does my research or experience provide any new insights or perspectives?

2. Do I often find myself having to explain this topic when I meet people as they are unaware of its relevance?

3. Do I believe that I could have saved myself time, money, and frustration if someone had explained to me the issues surrounding this topic?

4. Am I constantly correcting misconceptions regarding this topic?

5. Am I considered to be an expert in this field? For example, do I present my research or experience at conferences?

If your answer is "yes" to any of these questions, your topic is probably of interest to OSBR readers.

When writing your article, keep the following points in mind:

1. Thoroughly examine the topic; don't leave the reader wishing for more.

2. Know your central theme and stick to it.

3. Demonstrate your depth of understanding for the topic, and that you have considered its benefits, possible outcomes, and applicability.

4. Write in third-person formal style.

These guidelines should assist in the process of translating your expertise into a focused article which adds to the knowledgable resources available through the OSBR.

## Upcoming Editorial Themes

| | |
|---|---|
| **July 2008** | Accessibility |
| **August 2008** | Education |
| **September 2008** | Social Innovation |

**Formatting Guidelines**:

All contributions are to be submitted in .txt or .rtf format and match the following length guidelines. Formatting should be limited to bolded and italicized text. Formatting is optional and may be edited to match the rest of the publication. Include your email address and daytime phone number should the editor need to contact you regarding your submission. Indicate if your submission has been previously published elsewhere.

**Articles:** Do not submit articles shorter than 1500 words or longer than 3000 words. If this is your first article, include a 50-75 word biography introducing yourself. Articles should begin with a thought-provoking quotation that matches the spirit of the article. Research the source of your quotation in order to provide proper attribution.

**Interviews:** Interviews tend to be between 1-2 pages long or 500-1000 words. Include a 50-75 word biography for both the interviewer and each of the interviewee(s).

**Newsbytes:** Newsbytes should be short and pithy--providing enough information to gain the reader's interest as well as a reference to additional information such as a press release or website. 100-300 words is usually sufficient.

**Events:** Events should include the date, location, a short description, and the URL for further information. Due to the monthly publication schedule, events should be sent at least 6-8 weeks in advance.

**Questions and Feedback:** These can range anywhere between a one sentence question up to a 500 word letter to the editor style of feedback. Include a sentence or two introducing yourself.

**Copyright:**

You retain copyright to your work and grant the Talent First Network permission to publish your submission under a Creative Commons license. The Talent First Network owns the copyright to the collection of works comprising each edition of the OSBR. All content on the OSBR and Talent First Network websites is under the Creative Commons attribution (http://creativecommons.org/licenses/by/3.0/) license which allows for commercial and non-commercial redistribution as well as modifications of the work as long as the copyright holder is attributed.

The Talent First Network program is funded in part by the Government of Ontario.

The Technology Innovation Management (TIM) program is a master's program for experienced engineers. It is offered by Carleton University's Department of Systems and Computer Engineering. The TIM program offers both a thesis based degree (M.A.Sc.) and a project based degree (M.Eng.). The M.Eng is offered real-time worldwide. To apply, please go to: http://www.carleton.ca/tim/sub/apply.html.