"Opening the Black Box"

A Source Code Security Analysis Case Study

Jeff Williams
CEO, Aspect Security
Chairman, OWASP

Jack Danahy
CTO and Founder
Ounce Labs

ASPECT SECURITY
Application Security Specialists
www.aspectsecurity.com

OUNCE LABS
SOFTWARE SECURITY ASSURANCE
www.ouncelabs.com

Executive Summary

While businesses often understand the importance of maintaining secure applications, most companies have no idea whether their code is vulnerable or not. Applications have generally been accepted and deployed with no insight into their potential security impact, opening the floodgates for billions of dollars spent on patching systems, preventive technologies, and security services designed to protect against the compromise of flawed software.

We respectfully argue that the first step in assuring applications are secure is to open the black box; to look deep into the source code and identify the security vulnerabilities, design flaws, and policy violations that expose systems to attack. Peering even deeper, this process leads to the organizational root causes of the vulnerabilities, which can be addressed with an application security initiative to improve people and teams, policies and processes, and the technology supporting better software security.

This paper presents a case study of how companies can reliably verify and remediate the security of an application using source code during development or before deployment. Our primary research subject is the open source BitTorrent client Azureus, analyzed and discussed publicly with support from the Azureus development team[1] (http://azureus.sourceforge.net/). Additional trend data and general assessments were also made using a number of other popular open source applications, especially other file sharing applications such as eMule and LimeWire.

We used these applications as examples to answer the following questions:

- How do specific decisions or mistakes during the application lifecycle impact overall security?
- What are the essential steps of a thorough software security review?
- How do you set up an application security initiative for your organization?

The case study presents Aspect Security's "sweep and probe" method for verifying the security in an application, using Ounce Labs' source code security analysis technology to generate a set of findings covering critical security areas.

Azureus' application design was found to be generally not vulnerable to attack, and security mechanisms have been well-implemented within the code. Our analysis identified only two flaws that could possibly be exploited with serious impacts, as well as a small number of minor weaknesses. Our evaluation of the open source process based on other applications also generated positive results. Almost all projects demonstrated long-term reduction in vulnerabilities, primarily focusing on eliminating high-severity vulnerabilities first.

The process detailed in the case study below is meant to serve as a model for executives and managers to develop or enhance their own organization's application security process. With the current tools and knowledge available, it is no longer acceptable to trust applications based on a surface-level view of their security implications. Only by peering into the black box at the source of vulnerabilities and design flaws can organizations truly understand and eliminate the risks presented by their applications. The recommendations below are based on real-world techniques, proven successful by enterprises that have implemented them to measurably reduce software risk.

---

1 Paul Gardner and the Azureus Team voluntarily allowed us to use their application as a test subject for our security verification project. Their willingness to be put under the "application security microscope" publicly was extremely generous as was their support during the process. This paper would not have been possible without their support, assistance, and cooperation throughout the course of our study.

About the Authors

JEFF WILLIAMS is the founder and CEO of Aspect Security (http://www.aspectsecurity.com) and Chair of the non-profit OWASP Foundation (http://www.owasp.org). Jeff speaks frequently on application security and has published numerous papers on practical risk and assurance techniques. Jeff has advanced degrees in psychology, computer science, human factors, and a law degree from Georgetown University Law Center.

ASPECT SECURITY specializes in web application and web services security. Aspect's expert staff is responsible for the security of financial, healthcare, biotechnology, e-commerce, Fortune 500, and government web applications. Aspect provides code review, penetration testing, policy development, and developer security training services to find, diagnose, and eliminate vulnerabilities in custom web application code. Aspect is privately held and headquartered in Columbia, Maryland. To contact Aspect Security call 301-604-4882, visit us on the Web at www.aspectsecurity.com, or write to info@aspectsecurity.com.

JACK DANAHY, CTO and founder of Ounce Labs, is one of the industry's premier software security assurance advocates. He frequently contributes his expertise in national trade and business press, industry conferences, and organizations including the DHS Task Force on Security across the SDLC and DOD Homeland Security Information Sharing Initiative. He holds patents or has patents pending in kernel security, secure remote communications, systems management and distributed computing.

OUNCE LABS, the leader in software security assurance, delivers products that allow customers to verify that software meets their defined security requirements. Ounce Labs' enterprise-level automated source code analysis provides reliable vulnerability metrics necessary to manage software risk, enforce security policies, enhance audit capabilities, and track compliance efforts. Based on patents-pending Contextual Analysis technology, Ounce Labs' products also pinpoint specific software design errors and coding flaws to simplify remediation during any phase of the development lifecycle. Founded in 2002, Ounce Labs is located in Waltham, Massachusetts. For more information, please visit www.ouncelabs.com

TABLE OF CONTENTS

1 INTRODUCTION

Businesses are driving tremendous advances in software functionality to keep up with worldwide growth and demands for better connection to critical data and assets. Applications are typically built under strict time pressures and often represent a collection of outsourced, open source, and in-house development to fulfill specific business needs. Organizations need to be able to trust that this software has appropriate security mechanisms to thwart attacks and that the source code does not contain vulnerabilities that might expose networked resources.

It is irresponsible for organizations to subject themselves to unknown or unacceptable levels of risk in order to gain technical advantages. Vast improvements in workforce mobility, amount of data transferred, or number of transactions taking place can all be negated by a single security breach. This has become a demonstrable problem with business applications, as targeted application attacks have led to massive fraud and data theft incidents. Cyber criminals continue to find increased financial incentive in targeting business applications, suggesting that we are unlikely to see this trend slow down any time soon.

> *The best network, host and data security can't effectively protect a weak application. Security must be considered first in the application."*
>
> **Theresa Lanowitz, Gartner**
> Now is the Time for Security at the Application Layer (12/05)

2 WHY DO WE NEED TO UNDERSTAND SOFTWARE RISK?

Ongoing security breaches have demonstrated how devastating attacks can be when targeting applications with direct access to sensitive customer and business data. In many cases, decisions were made to purchase and deploy software that did not meet specific security requirements. Considering the potential for severe negative publicity, loss of customers, interruptions to business operations, and settlements with the Federal Trade Commission, the most likely assumption is that these decisions were made without sufficient assessment or knowledge of the software's security shortcomings.

Software is absolutely critical to every aspect of business, connecting employees, partners, and customers to critical data and resources. Many of these applications are internet-facing, though corporate intranets and extranets are providing additional channels of access to sensitive data. Further complexities arise in the integration of legacy software with networked services and the growing use of custom, outsourced, and open source applications. When software becomes critical to a business – in some cases even becoming the whole business – understanding the risk it presents to the organization is of paramount importance.

Despite this criticality however, many businesses are currently operating without any insight into the security levels of their applications. Even sophisticated development teams focus resources on verifying applications do what they are supposed to do (functionality) instead of making sure they do not do what they are not supposed to do (security). Additionally, the explosion of web-enabled applications and web services in the past few years has outpaced most organizations' ability to keep up with new threats and security challenges in order to properly protect against them.

To truly be informed about the security of an application requires an understanding not only of the technical guts of the application, but also of the people, processes, and tools used to create the application. Only with this level of information can appropriate decisions be made to capitalize on applications without exposing critical resources to unacceptable risk.

2.1 APPLICATION SECURITY IS NOT NETWORK SECURITY

Securing applications is quite different than securing networks. In the network security world, there are thousands of researchers who test and reverse-engineer products to find security holes. Once vulnerabilities are uncovered, scanning and intrusion detection products include the signatures for these exploits and help install the latest patches to maintain security.

However, this approach does not work for applications. Because applications are frequently custom-built or heavily customized, there may only be one installation of the application anywhere, so there is not a community to work together identifying and fixing vulnerabilities. Without signatures, vulnerability scanners and intrusion detection systems are blind to the unique flaws in these custom applications. Human attackers however, can find vulnerabilities in custom applications quite easily and exploit them either manually or with specially-designed tools.

Most existing network security teams are ill-prepared to handle application security. Typically, these teams are trained to search for known network security issues and respond. Achieving application security requires the ability to search applications for issues that are unique and previously unknown. Team members must be able to read code with a deep understanding of how software architectures work. Also, responding to vulnerabilities generally involves the ability to change code and redeploy applications.

2.2 Finding Vulnerabilities is Not Enough

No one can make good decisions about risk without clearly understanding the business context to which that risk applies. Finding vulnerabilities is only a small part of this understanding. Even thorough security scanners that present a list of vulnerabilities in an application may not give the complete picture of the risk it presents. Further analysis is required to verify that the results make sense according to the function of that application.

There are also a number of design elements that affect the security of an application that may not specifically be vulnerabilities. Access control, authentication, encryption, logging, and proper input validation and encoding are common security mechanisms that should be verified before deployment. Less-obvious design elements that have an impact on security include calls that invoke unsupported applications interfaces, native or dynamic code, databases, or network communications.

Companies with portfolios of many applications face an additional challenge. In order to invest time and effort in programs to improve the organization’s overall application security, it should be known which applications represent the greatest risk. Many organizations struggle to even maintain a list of their applications, so managing them according to risk levels may require significant changes in process.

The first step is to determine which application characteristics truly affect the risk of applications; the key is to eliminate those characteristics that don’t make a difference. Once a core set of characteristics has been identified, an organization can establish the degree to which security efforts are valuable. With this approach applied to each critical area of security, application assessments will enable informed risk management decisions.

2.3 Tracking Down Root Causes

Application security issues result from errors during development that can be categorized into three key areas: insufficient processes or practices, inadequate skills or teams, and incomplete supporting technology. While security technologies are critical to an organization’s application security efforts, they must be paired with the right set of team and process improvements.

The root causes of application security issues are deeply ingrained in many organizations. Much of the current software development culture evolved in a time when the threats to applications were not nearly as critical as they are today. Almost all software now in development is either currently networked or will be soon. Connecting applications to networks dramatically increases risk, yet our software development techniques have not appropriately adjusted to this change.

For example, many recent, high-profile security breaches occurred because basic security design elements, such as use of encryption, were either not included in the software requirements or were overlooked during development. While they do not always garner the same amount of attention as coding errors like unchecked buffers or unvalidated inputs, oversights such as lack of encryption for financial transaction software seem to be obvious mistakes that would have been caught ahead of time had proper software security activities been in place.

The most common issues in the process area are failures to define clear and detailed security requirements, conduct threat modeling activities, or perform security testing and analysis. In the category of skills and teams, many developers are not trained in secure coding, and very few organizations have an application security team to support development projects. Finally, most organizations have not put in place supporting tools and technologies to identify and diagnose vulnerabilities and have not established coding guidelines or standard libraries that then result in uniform and consistent security functions and calls.

Addressing root causes as early in the lifecycle as possible has clear benefits. As a project moves from a concept to requirements, design, and implementation, ostensibly minor flaws tend to become entrenched and increasingly difficult to fix. Finding and addressing the organizational root causes of these vulnerabilities helps to prevent them before they present actual risk.

2.4 APPLICATION SECURITY INITIATIVES

Several key decisions are required to determine the methodology that best suits your organization's goals for improved application security. The most effective approach largely depends on the assessment of risk, security resources available, and metrics that will determine success. These factors will impact which tools are used, what efforts of the project are outsourced, and the scope of the initiative.

If implemented correctly, a focused initiative to improve software security can transform a company's ability to reliably produce and deploy secure applications. These programs involve training, team-building, software lifecycle process improvements, and technology to support securing applications. Several vendors, including Microsoft, RedHat, and Compuware have publicly committed to these initiatives and are starting to report concrete benefits. Microsoft, for example, has noted a 60% drop in application security issues on projects where their secure development lifecycle is followed.

Similar gains continue to be made by internal development organizations in major companies as well as those that generate software as outsourced providers. More importantly, application security initiatives allow security audit and review teams to assess applications in production across large enterprises, including legacy applications, which often represent the largest source of software risk.

3 How to Establish Trust in an Application

This section describes a process for verifying the security of an application. The goal of this process is to evaluate all the potential security risks in an application and verify that they have been adequately addressed. Performing this process at each stage in the software development lifecycle is the most effective way to assure applications meet security standards.

3.1 WHAT TO MEASURE

When setting out to measure the security of an application, there are a number of elements to consider. Direct measurements include the type of vulnerabilities as well as the presence of important security features, although information can also be gathered with an appraisal of the teams, policies, processes, and additional technologies that support the development effort. These indirect measures of an organization are highly predictive of whether an organization's products are secure or not.

Attributes that should be measured at the project level include security procedures in the software development lifecycle, development and testing technology, project personnel, and management structure. This information can be gathered by interviewing project participants, reviewing project documentation, evaluating training classes, or even by asking developers during security verifications.

Attributes that should be measured at the organizational level include training programs, procedures for setting policy and process standards, and investments in technology. This information can be gathered through formal appraisals or informally during interaction with security and development groups.

Evaluating these indirect measures of application security is especially helpful when working with an outsourced or commercial vendor. With outsourced projects, the buyer typically has access to the source code and can run their own assessments, but in both cases, it is important to verify that the organization and processes are conducive to secure development.

3.2 APPLICATION SECURITY VERIFICATION

We use the term “application security verification” to describe efforts to gain trust in an application. Some of the techniques frequently used to verify applications include vulnerability scanning, static analysis, penetration testing, manual code review, threat modeling, and architecture review. All of these techniques are useful and important, but should be used strategically, where they are the most effective.

Most importantly, organizations should be able to clearly and completely articulate vulnerability levels and explain how the expected risks have been adequately addressed. Verification establishes a level of assurance in the application that can be communicated to the application users as well as internal teams.

Rather than guaranteeing perfect confidence that there are no security flaws in an application, verification is about gathering enough information to make informed decisions to manage or accept software risk. The goal is to find a balance between the level of unacceptable risk and the level of affordable analysis and remediation. Achieving this goal requires the discipline to focus first on the areas most likely to contain critical vulnerabilities. There is rarely enough time to apply a comprehensive application security checklist and apply it to an entire software baseline.

Flexibility is required in this decision-making process, because changes in what issues an organization deems ‘critical’ will greatly affect a verification strategy. Also, the variety of applications, environments, and situations that assessment teams come across may require specialized tools, changes in approach, or alternate metrics to assure the most worthwhile analysis.

4 Verifying Azureus: The Sweep and Probe Method

The application security specialists at Aspect evolved the “sweep and probe” method over years of verification efforts for a variety of clients. The idea behind this approach is to direct and control the level of effort at each stage to get maximum efficiency out of an application review. In brief, this process includes:

1. **Preparation:** Evaluation of the application’s business context
2. **Sweep & Probe – Pass I:** Prioritize areas for examination
3. **Sweep & Probe – Pass II:** Explore details of prioritized areas
4. **Sweep & Probe – Pass III:** Analyze the most critical attack vectors
5. **Creating and Managing Findings:** Document results for business decisions
6. **Root Cause Analysis:** Identify and eliminate organizational sources of persistent problems

For this paper, we chose to demonstrate our unique method using Azureus, a file-sharing application deployed by millions of people around the world. A description of the application from the Azureus web site (http://azureus.sourceforge.net/doc/Azureus_User_Guide.htm) explains:

> “Azureus is a BitTorrent Java client. The BitTorrent protocol is a new way of exchanging or distributing data over the internet (see http://bitconjurer.org/BitTorrent/introduction.html). Downloading also means uploading, and the amounts of each are linked, to ensure fairness and rapidity in the spread of the file at hand. To be able to download a file, you first need to get the associated .torrent file. This file, usually a dozen KB in size, is the “signature” of the much bigger file to be downloaded, and it needs a software to be read properly. Azureus is such a software. If you want to host files yourself, you need a tracker, which is basically a central server coordinating the connections between peers. Azureus can provide a tracker too.”

We did not perform a line-by-line review of the Azureus, and this review should not be considered comprehensive. Our goal for this review was to verify the application well enough to confirm that critical threats against the majority of Azureus users are unlikely.

4.1 PREPARATION

The first step in any security verification effort is to truly understand the application and the business operations it will be supporting. Without understanding the context of threats, assets, functions, and related business processes, it is impossible to set priorities by which to identify and evaluate the security strengths and weaknesses in an application. The preparation step involves a review of available documentation and meetings with appropriate project personnel. This is the most efficient way to identify what is most important in terms of potential attacks on the application and its assets and functions and correspondingly the impacts and risks on the business from such attacks.

For Azureus, we considered the typical users of BitTorrent and the potential attacks against users of the client. We identified a number of attack vectors, which are listed below. Note that Azureus is written in Java, so the most common client application attacks, buffer overflows, are not an issue.

Priority	Verification Priority	Attack Type
High	Self update feature	Code Injection
High	Modifying download/upload reporting	Data Integrity
Medium	Changing the configured download limit/rate	Access Control
Medium	Removing, modifying, bypassing configured blocked IP list.	Access Control
Medium	Protocol errors	Input Validation
Medium	Error handling and logging	Monitoring
Low	Trying to download/upload malicious files	Data Corruption
Low	Flooding the tracker server with requests	Denial of Service
Low	Flooding peers with connection attempts	Denial of Service

Figure 1: Priorities Set During Preparation Phase

4.2 Sweep & Probe – Pass I

The first pass of the sweep and probe method is focused on quick information gathering to prioritize the areas to examine. Vulnerability scanning and static analysis tools create a set of preliminary findings that can indicate areas that deserve additional scrutiny. They may provide a reliable overview of application vulnerability levels and security design features, but in a detailed source code review, they function as a first pass for the manual review team.
For example, to verify that a Java application is protected against SQL injection attacks, the static analysis tools can verify that the application does not use the susceptible java.sql.Statement interface, but instead uses the java.sql.PreparedStatement interface, which protects against injection. Static analysis is quite good at identifying security-relevant areas of code, the invocation of dangerous methods, and programming patterns that lead to vulnerabilities.

During Pass I, we attempt to accomplish two things.

1. Identify any obvious security issues and eliminate any results that can clearly be ignored.
2. Based on the results of the preparation and scanning, identify the critical security areas for this application and create a preliminary plan for the remainder of the verification effort.

The Eclipse Metrics plug-in generated a number of helpful pieces of information for our initial review, including:

- 1,920 java source files
- 194,538 lines of code
- 6 libraries from various sources

We then ran the Azureus source code through Ounce Labs' software security assurance product. This tool uses compiler technology to analyze software source code in the context of how information is entered and executed by the application. It generates reports by comparing the assessment against a knowledgebase of over 100,000 types of coding errors, design flaws, and policy violations.

The engine took just under one hour to scan the entire application, identifying a total of 4925 items for further review. Ounce Labs' reports separate confirmed vulnerabilities from results categorized as 'exceptions', which require more detailed analysis. From the reports

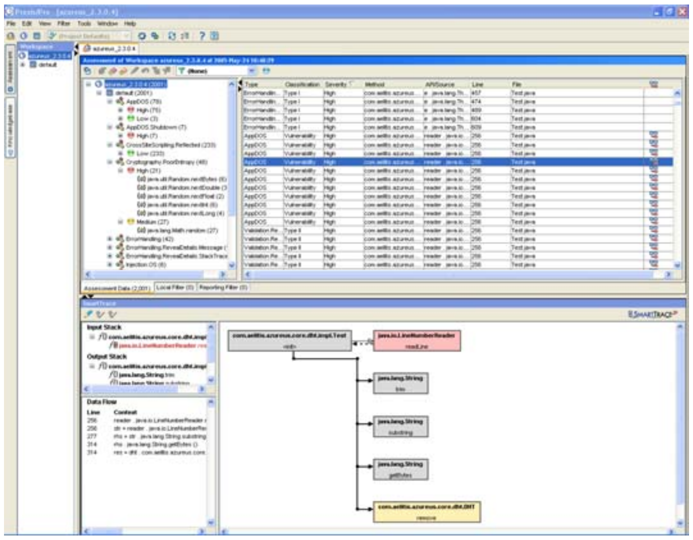


Figure 2: Ounce Labs Analysis of Azureus

generated, we were able to eliminate cross-site scripting and encoding required issues, which are not relevant for this type of BitTorrent client because it does not utilize a browser interface. It is often important to note these findings however, since such vulnerabilities may become relevant if the code were later reused to build an internet-facing application with similar functionality.

The Ounce Labs product also helped us confirm that the application makes heavy use of System.out.println() and Exception.printStackTrace(), which are not appropriate error handling and logging mechanisms. We were also able to see that Azureus includes 9 calls to the operating system and some native code. These calls deserve additional verification, as they can be quite dangerous from a security perspective.

Based on the Ounce Labs assessment results, we found a plugin architecture that allows code to be loaded at runtime, 11 calls to unsupported libraries in the JDK, and 61 places where a weak random number generator is used, and lots of socket code. Azureus relies on several libraries, including the BouncyCastle cryptographic provider, the Programmers-Friend library at http://www.programmers-friend.org, and the log4j library from Apache.

This first pass also points out several interesting facts about what is not in the Azureus application. The Ounce Labs Knowledgebase covers many of the common libraries used in Java applications, so we can tell that either the application is not using a mechanism at all, or they have implemented their own. In this case, we can see that Azureus does not use a standard mechanism for authentication, access control, or SSL. We can also see that the application doesn't use the standard database, CORBA, DNS, Email, RPC, LDAP, XML, or validation APIs. This doesn't necessarily represent a problem, but helps to prioritize the issues to look for.

One confusing factor at this stage is that Azureus, like many applications, contains many main programs and a large amount of test code. This makes it difficult to differentiate between real problems and issues that are confined to test code. Based on the data generated by our scans and confirmation of details, the elements we categorized for further investigation were:

Priority	Verification Priority	Attack Type
High	Self-update feature	Code Injection
High	Operating system calls	Command Injection
Medium	Plugin architecture	Access Control
Medium	Cryptographic code	Data Integrity
Medium	Error handling and logging	Monitoring
Low	Trying to download/upload malicious files	Data Corruption
Low	Flooding the tracker server with requests	Denial of Service
Low	Flooding peers with connection attempts	Denial of Service

Figure 3: Priorities Set After Pass I

4.3 SWEEP & PROBE – PASS II

The second pass focuses on exploring the priorities established by the initial exploration. During this pass, each of the security critical areas is analyzed using a combination of targeted code review and penetration testing. During this phase, the items flagged as vulnerabilities by the scanning and static analysis tools are investigated and confirmed.

The first step is to identify the mapping between the source code baseline and the running application. This requires understanding the basic architecture of the software, its attack surface, and the conventions used by the developers on the project. The best source for this information is actually using the application in conjunction with analyzing the structure of the software baseline. Project documentation can be useful, but is often misleading, so it is important to be cautious in spending time analyzing documentation.

To verify a security mechanism in an application requires discipline and focus. Wandering through the code without a purpose is not productive. Generally, the first step is to use the mechanism to see how it works. For example, when verifying access control, the best place to start is to find a function that one role can invoke and another role cannot and experiment with it.

Once a general idea of how the mechanism is supposed to work is established, a strategy for validating the mechanism can be identified. In many cases, tracing the execution of the mechanism through the source code is a good place to start. However, in some cases it is easier and more productive to start with penetration testing techniques, actually exercising the mechanism in the running application.

This phase requires tools that allow navigation, searching, sorting, visualization, and organization of the source code baseline through a security lens. Penetration testing tools, such as the free WebScarab tool from OWASP, are also important in order to verify potential security problems identified by static analysis or noticed in examining the code. The more these tools are "tuned" to the particular application being analyzed, the better. The challenge with application penetration testing tools is addressing the broad variations found among different applications. They often have difficulty identifying the security mechanisms and understanding how they work, though in many cases they can be 'taught' about the application's business logic in order to generate more accurate results. The more the tools are taught about the specific application, the better they will be at identifying real problems and eliminating false positives.

For Azureus, this pass focused on authentication, input validation, error handling, logging, encryption, secure communication, and other common security mechanisms. Pass II resulted in the discovery of a number of minor issues such as the use of System.out.println() and Throwable.printStackTrace(). However, the automatic update feature, use of environment variables, and searching for malicious code were identified as the priorities for Pass III because they represented the highest degree and likelihood of risk.

Priority	Verification Priority	Attack Type
High	Self-update feature	Code Injection
High	Operating system calls	Command Injection
Medium	Cryptographic code	Data Integrity

Figure 4: Priorities Set After Pass II

4.4 SWEEP & PROBE – PASS III

After identifying clear priorities in the first and second passes, the third pass focuses on deep analysis of the most serious attack vectors. This pass uses manual code review along with targeted penetration testing in order to examine specific attack vectors closely. The use of penetration testing in combination with static analysis is critical to successful deep analysis of applications.

In some development environments, not all of the developers who contributed to the software are fully trusted by the business, which increases the potential for malicious code. Examples include Trojan horses, backdoors, timebombs, and logic bombs, all of which are particularly difficult to find, because attackers usually go to great lengths to obfuscate the attack. In fact, skilled attackers may make malicious code look indistinguishable from an inadvertent mistake by a programmer.

Verifying that an application does not contain any malicious code is a significant undertaking. However, there are many applications that are so critical that this level of analysis is justified. Static analysis tools are essential to this level of review, as they can help to locate atypical calls and other anomalies that may indicate malicious code.
In our verification of Azureus, we found code related to the automatic update function, but neither the static analysis tools nor automated vulnerability scans confirmed this code as a definite source of potential vulnerability. Only by following the verification process were we able to identify this as the most serious attack vector on the Azureus client.

4.5 CREATING AND MANAGING FINDINGS

During the verification of an application, it is important to capture each finding accurately so that it can be communicated to the developers, project management, security and audit teams, and senior management. Each of these levels requires a different view of the results and will use it for different purposes. These findings have been refined through the sweep and probe process and are now highly focused, detailed, and accurate descriptions of each potential vulnerability.

A good security finding is immediately useful to the developers on the software project. In the case of the Ounce Labs tools we were using, the reports provided details about the line-of-code location of the vulnerability, a description of what it was and how it could be exploited, the severity ranking of the vulnerability, and suggestions on ways to remediate it. This information is critical for analysts that are responsible for improving the security of an application, which we could have done in this case by launching a code editor directly from the results report to the vulnerable line of code.

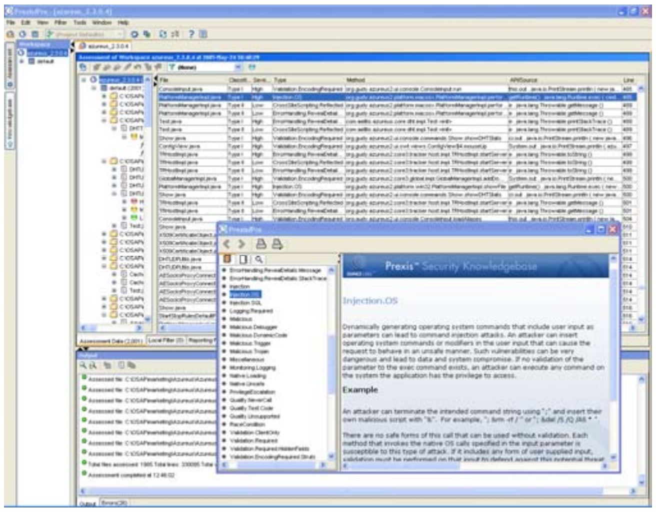


Figure 5: Ounce Labs Analysis with Remediation Help and Link to Code Editor

These findings should also be tracked through resolution, as there is much more to learn than just the vulnerability itself. These findings are a window into the organization's application security ability and can enable insight into security patterns across projects. This will allow strategic organizational improvements based on clear and substantiated trends.

From a management perspective, the results can also allow comparisons among different applications, projects, or development teams. Ounce Labs' products gave us the ability to identify the projects and files in Azureus that represented the highest criticality, which can be used to guide work flow strategy. For managers keeping track of multiple projects over time, trend reports are also available.

4.6 Root Cause Analysis

Identifying the root causes of application vulnerabilities may require only the information from the software itself, but it is far more successful when additional information about the development process and organization is available. This extra information can be obtained during interviews, appraisals, training efforts, and other sources.

Root cause analysis is the process of looking for the people, process, and technology issues that either caused a vulnerability to exist or allowed it to continue undiscovered. Many problems have multiple root causes that extend from the initial introduction of a problem all the way through its discovery (or lack thereof).

For example, there are several potential causes of an access control vulnerability discovered in an intranet-facing web application. Working our way back through the application development process, the appropriate questions would be:

5) Why wasn't this vulnerability caught during testing?
4) Why wasn't this vulnerability caught during detailed code review?
3) Why wasn't a standard library for authorization used?
2) Why wasn't this vulnerability caught during design review?
1) Why wasn't authorization properly defined as a requirement?

Having all the applicable root causes, one can then ascertain which are most probable, the degree of confidence that each is the root cause, and potential remediation steps that would work for the organization's specific people, processes, and technologies.

5 Azureus Findings Overview

Overall, the Azureus baseline is typical for a relatively large open source project that has been evolving for several years. Our analysis found two weaknesses, neither of which would be easy to exploit. However, the impact of these two issues is serious enough across the millions of Azureus users to warrant remediation. We also made a few findings that do not represent significant security issues, but provide additional insight into the organization that created the software.

5.1 STRENGTHS

By its design, Azureus is resistant to most of the attacks that we considered for this verification effort. Since it is written in Java, the threat from buffer overflows and related attacks is eliminated. Azureus only allows configuration changes via the client GUI, which is not remotely accessible. Therefore, an attacker would have a difficult time trying to exploit or change these settings without access to the client.

Also, modifying the files being uploaded and downloaded is not possible due to the use of the .torrent file which includes piece hashes of the original file. Attempts to trick the BitTorrent client into serving pieces of other files would seem improbable since the original hash would not match and clients verify this original .torrent hash value for each piece before uploading content. Similarly, trying to upload malicious file contents is unlikely because the client verifies the hash of each piece before writing to disk.

5.2 WEAKNESSES

The analysis identified several weaknesses, most of which can be easily remedied. Several of our findings are instances of pervasive issues that we have consolidated into a single finding. Two of these issues are more serious, and, although an exploit is unlikely, a successful attack could allow a complete takeover of the client machine.

NOTE: The Azureus team is fully aware of these results and gave permission to have them published.

High	**Unverified software updates**			
Location	org.gudy.azureus2.update.CoreUpdateChecker.java			
Risk Description	Upon initial start of the application, Azureus checks specific server(s) to see if it needs to update itself, if it determines an update is needed, it will use the BitTorrent strategy to download the update/patch, integrate the update into its jarfile and then restart itself. It appears that software updates are automatically downloaded from sourceforge, patched into the jar file, and executed. URL mirrors_url = new URL("http://prdownloads.sourceforge.net/azureus/" + latest_file_name + "?download"); One possible attack would be to watch the sourceforge upload directory, grab the Azureus update, and post a Trojaned replacement with the same name. The Azureus team might then grab the Trojaned file and post it as an official update. This would allow an attacker to compromise a large number of hosts very quickly. This attack applies to plugins as well. If successful, the attacker would be able to take full control of any machines compromised by the malicious patch.			
Risk Factors	**Likelihood:**	**Medium**	**Impact:**	**High**
Recommended Solution(s)	Consider a more trustworthy source for handling software updates. You could also use jar signing to sign updates and verify them in the client. This can be done with tools provided with Java, as follows: jarsigner -keystore .keystore -storepass password myjar.jar alias jarsigner -verify myjar.jar			

Medium	Use of unvalidated environment variables in Runtime.exec()			
Location	Runtime.exec (cmdargs) org.gudy.azureus2.platform.macosx.PlatformManagerImpl.performRuntimeExec (485) Runtime.exec (command_line) org.gudy.azureus2.pluginsimpl.local.utils.UtilitiesImpl.createProcess (241) Runtime.exec (exec) com.aelitis.azureus.core.update.impl.AzureusRestarterImpl.restartAzureus_win32 (287) Runtime.exec (exec) org.gudy.azureus2.ui.swt.updater.snippets.Main.spawnStarted (64) Runtime.exec (execStr) com.aelitis.azureus.core.update.impl.AzureusRestarterImpl.chMod (451) Runtime.exec (“explorer.exe /e,/select\" + file_name+ "\") org.gudy.azureus2.platform.win32.PlatformManagerImpl.showFile (500) Runtime.exec (new java.lang.String[][2]) org.gudy.azureus2.ui.common.util.UserAlerts$3.runSupport (85) Runtime.exec (new java.lang.String[][3]) org.gudy.azureus2.ui.common.util.UserAlerts$3.runSupport (88) Runtime.exec (new java.lang.String[][3]) org.gudy.azureus2.ui.swt.osx.CarbonUIEnhancer.stopSidekick (185)			
Risk Description	An attacker might be able to append a command (like “& format c:”) onto one of the environmental variables (both shell environment and System.properties) and cause his malicious command to be executed via Azureus code. This command would run with the same privilege Azureus has. To be successful, the attacker would need to be able to manipulate environment variables in the user’s environment, but this could be a way to escalate privileges. The impact here is high because this flaw could enable an attacker to run arbitrary programs on a victim’s machine. The likelihood is only low because the attack requires a fair amount of technical skill and would be difficult to launch.			
Risk Factors	**Likelihood:**	**Low**	**Impact:**	**High**
Recommended Solution(s)	The application should validate all input from untrusted sources, even environment variables. This will eliminate the possibility that an attacker could use Azureus to escalate privilege and take over a client’s machine.			

We also found quite a number of smaller issues that would be difficult or impossible to exploit, but that are not consistent with best practice.

Low	Use of custom cryptographic code for SHA-1			
Location	org.gudy.azureus2.core3.util.SHA1.java			
Risk Description	The application uses SHA-1 to generate hashes for pieces of a given file. This hash info is contained in the .torrent file and is verified on the actual file content pieces being downloaded/uploaded. Azureus does not use the standard Java encryption libraries to implement the SHA-1 algorithm. Instead the project has implemented its own custom class to implement the algorithm. The likelihood of a problem here is minimal as Azureus would not interoperate with other BitTorrent clients if it was not implemented properly, though the use of the standard libraries is recommended. Also, in theory, an attacker could serve peers with corrupt data that happens to have the same hash values of the original file. SHA-1 has been proven to be insecure by today’s standards and is known to have collision problems; SHA-256 or SHA-512 are recommended. This is a part of the torrent specification, which will have to be updated to address this finding.			
Risk Factors	**Likelihood:**	**Low**	**Impact:**	**Low**

Recommended Solution(s)	Update the specification with a more secure hash algorithm, then use the standard Java encryption libraries to implement the hash verification.

Low	**Use of telnet instead of SSH**
Location	package org.gudy.azureus2.ui.telnet.UI.java
Risk Description	The application provides a capability to use telnet for certain management related tasks. This is not “ON” by default but must be intentionally started via command execution of " java -Dazureus.console.multiuser=1 ..... org.gudy.azureus.ui.common.Main --ui=console".
Risk Factors	**Likelihood:** **Low** **Impact:** **Low**
Recommended Solution(s)	The assumption by the Azureus team is that users understand the weaknesses of using this method of doing management. It is recommended that if remote administration functionality is used, SSH should be used instead of telnet.

Low	**Use of BASIC authentication over insecure connection**
Location	org.gudy.azureus2.core3.tracker.server.impl.tcp.TRTrackerServerProcessorTCP.java
Risk Description	Azureus can use BASIC authentication over ordinary http in its Internal tracker web server. This feature is provided to support non-SSL enabled clients. Basic authentication is known to pass credential information on each request with password values being only weakly obfuscated (base64 encoded) and easily recovered. Usage of http basic authentication increases the potential of credential leakage when not transferred over an encrypted line.
Risk Factors	**Likelihood:** **Low** **Impact:** **Low**
Recommended Solution(s)	The assumption by the Azureus team is that users understand the weaknesses of using this method of authentication. It is recommended that if remote administration functionality is used, a stronger authentication scheme or SSL should be used.

Low	**Use of java.util.Random instead of java.util.SecureRandom**
Location	122 places throughout the baseline, including: java.util.Random.nextInt () com.aelitis.azureus.core.dht.transport.udp.impl.DHTTransportUDPImpl$3.runSupport (531) java.util.Random.nextInt (optimistics . java.util.ArrayList.size()) com.aelitis.azureus.core.peermanager.unchoker.UnchokerUtil.getNextOptimisticPeer (130) java.util.Random.nextBytes (buffer) org.gudy.azureus2.core3.util.test.SHA1Verification.createTestFiles (55) java.lang.Math.random () com.aelitis.azureus.core.dht.vivaldi.maths.impl.HeightCoordinatesImpl.unity (75) java.lang.Math.random () com.aelitis.azureus.plugins.clientid.ClientIDPlugin.createPeerID (117) java.lang.Math.random () org.gudy.azureus2.core3.util.MD5.main (345) java.lang.Math.random () org.gudy.azureus2.core3.tracker.server.test.Main$4.run (306) …
Risk Description	The use of an insecure random number generator for security purposes creates the possibility that an attacker could guess the sequence and subvert a security mechanism.
Risk Factors	**Likelihood:** **Low** **Impact:** **Low**
Recommended Solution(s)	These calls should be replaced with SecureRandom wherever there is a security implication associated with the quality of the pseudorandom number generatation. The calls to Random that are a part of security mechanisms or used in some security decision should be examined closely and possibly replace with SecureRandom.

Low	Use of System.out.println and e.printStackTrace instead of logging			
Location	398 calls to Throwable.printStackTrace() 492 calls to Throwable.toString() 38 calls to System.err.println() 518 calls to System.out.println();			
Risk Description	While Azureus uses a popular logging library, log4j from Apache, it is not used consistently. If a security event were to occur, it would most likely be difficult to integrate the messages from System.err and System.out with the events in log4j. As they are global variables for the entire JVM, it is possible for System.err and System.out to be reset by a plugin and then all these events would sent to an unknown location.			
Risk Factors	**Likelihood:**	Low	**Impact:**	Low
Recommended Solution(s)	Remove test and debug output from the production system, and change all the other messaging systems over to use log4j.			

Note	Use of unsupported Sun methods in Java runtime
Location	sun.misc.CharacterDecoder.decodeBuffer (body) org.gudy.azureus2.core3.tracker.server.impl.tcp.TRTrackerServerProcessorTCP.doAuthentication (474) sun.misc.CharacterDecoder.decodeBuffer (password) com.aelitis.net.udp.impl.PRUDPPacketHandlerImpl.sendAndReceive (564) sun.misc.CharacterDecoder.decodeBuffer (pw) org.gudy.azureus2.core3.tracker.server.impl.tcp.TRTrackerServerProcessorTCP.doAuthentication (527) sun.misc.CharacterEncoder.encode (pw) org.gudy.azureus2.ui.swt.auth.AuthenticatorWindow.getAuthentication (163) sun.misc.Cleaner.clean () com.aelitis.azureus.core.diskmanager.MemoryMappedFile$2.run (224) sun.misc.CharacterDecoder.decodeBuffer (body) org.gudy.azureus2.core3.tracker.server.impl.tcp.TRTrackerServerProcessorTCP.doAuthentication (474) sun.misc.CharacterDecoder.decodeBuffer (password) com.aelitis.net.udp.impl.PRUDPPacketHandlerImpl.sendAndReceive (564) sun.misc.CharacterDecoder.decodeBuffer (pw) org.gudy.azureus2.core3.tracker.server.impl.tcp.TRTrackerServerProcessorTCP.doAuthentication (527) sun.misc.CharacterEncoder.encode (pw) org.gudy.azureus2.ui.swt.auth.AuthenticatorWindow.getAuthentication (163) sun.misc.Signal.getName () org.gudy.azureus2.core3.util.ShutdownHook$1.handle (50) sun.misc.Signal.getName () org.gudy.azureus2.core3.util.ShutdownHook$2.handle (66) sun.misc.Signal.handle (new sun.misc.Signal, new org.gudy.azureus2.core3.util.ShutdownHook$1) org.gudy.azureus2.core3.util.ShutdownHook.install (47) sun.misc.Signal.handle (new sun.misc.Signal, new org.gudy.azureus2.core3.util.ShutdownHook$2) org.gudy.azureus2.core3.util.ShutdownHook.install (63) sun.misc.SignalHandler.handle (sig) org.gudy.azureus2.core3.util.ShutdownHook$1.handle (53) sun.misc.SignalHandler.handle (sig) org.gudy.azureus2.core3.util.ShutdownHook$2.handle (69)
Risk Description	The use of these unsupported calls is not best practice as there is no guarantee that these calls will work as advertised in future versions of the Java environment.
Recommended Solution(s)	Replace these calls with an equivalent library or call that is within the control of the project. Most of these have already been removed by the Azureus team.

5.3 POSSIBLE ROOT CAUSES

Most of the issues we identified in Azureus are not particularly serious. But like all application security issues, they point to possible root causes in the organization that lead to these findings. In this section we consider the types of organizational problems that could have led to security issues like the ones that we found in Azureus.
We know that Azureus is an open source project with many contributors and a large software baseline. Since we did not perform any kind of appraisal of their project or organization, we have only the application itself from which to draw conclusions. The best way to proceed would be to perform an appraisal or have some other interaction with the development team and project governance to try to identify possible improvements.

1) The update flaw is really a design-level flaw as opposed to a simple implementation error. This suggests that either security was not captured in the requirements or use cases for this feature, or it was not properly implemented in the design. The solution for this type of issue is to ensure that security is a part of the requirements and architecture process. Threat modeling has been adopted by many organizations as a way to make sure that threats to an application have been identified and controlled.
2) Several of the other issues, such as the use of unsupported methods and the widespread use of System.out.println and related methods suggest that the project does not have a guideline to tell developers how to write code for Azureus. Without this type of instruction, developers will likely be inconsistent with their approaches.
3) Another possibility is that the project does not make regular use of static analysis tools and address the problems identified. These tools can be tailored to verify that security coding guidelines have been followed properly.

Again, these are only conjecture, and are included here to demonstrate that all application security issues can be traced to root causes in the organization that produced the code.

5.4 HOW TO MAKE STRATEGIC APPLICATION SECURITY DECISIONS

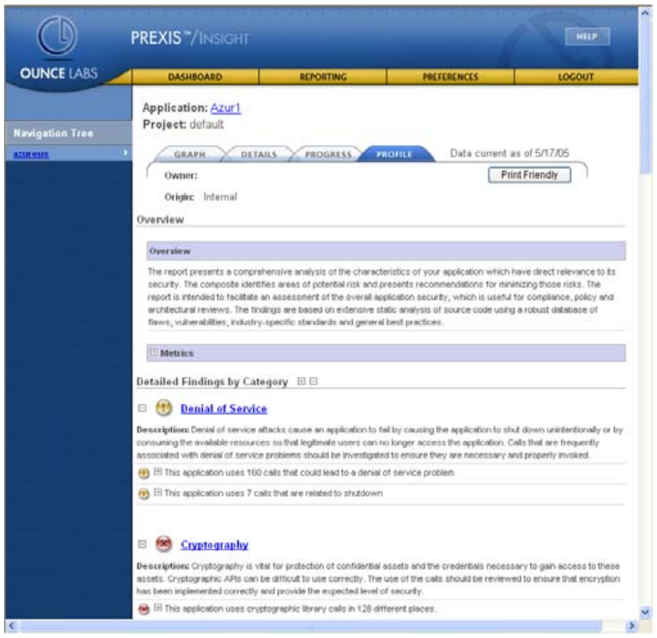
The decision to deploy an application must weigh the benefits against the best possible evaluation of potential risks. An organization should understand the various costs it would incur if a successful attack on the application led to data theft or a critical business function being taken offline. As in the case of Azureus, security analysis must take into consideration the context of how vulnerabilities will affect systems connected to the application.

Obviously, investment in security should be made where it will make the biggest reduction in risk to the organization over time. As discussed above, the first step in figuring out what improvements to make is to identify the root causes. The next step is to identify specific improvements that target the most significant issues across the most projects.

Ideally, specified security policies will be assigned for elements that dictate how applications manage information in relation to the business function it is performing. Clearly defined security standards, including the use of features like encryption, access control, and network communications, can serve as a simple checklist for applications before deployment, presenting the easy option of a pass/fail decision.

A practical model for this is Ounce Labs' Profile Report, which generates a detailed audit of design features that affect an application's overall security. Depending on the context of where and how the application will be deployed, these results allow users to make quick judgments on whether the application exhibits poor design or does not meet defined security standards. (For example, applications built to store sensitive data should be expected to employ a certain level of encryption for those functions.)

Source code analysis products should also provide quantitative measures of the number and severity of



Application: Azur1
Project: default

Data current as of 5/17/05

Owner:

Origin: Internal

Overview

Overview

The report presents a comprehensive analysis of the characteristics of your application which have direct relevance to its security. The composite identifies areas of potential risk and presents recommendations for minimizing those risks. The report is intended to facilitate an assessment of the overall application security, which is useful for compliance, policy and architectural reviews. The findings are based on extensive static analysis of source code using a robust database of flaws, vulnerabilities, industry-specific standards and general best practices.

Metrics

Detailed Findings by Category

Denial of Service

Description: Denial of service attacks cause an application to fail by causing the application to shut down unintentionally or by consuming the available resources so that legitimate users can no longer access the application. Calls that are frequently associated with denial of service problems should be investigated to ensure they are necessary and properly invoked.

This application uses 160 calls that could lead to a denial of service problem

This application uses 7 calls that are related to shutdown

Cryptography

Description: Cryptography is vital for protection of confidential assets and the credentials necessary to gain access to these assets. Cryptographic APIs can be difficult to use correctly. The use of the calls should be reviewed to ensure that encryption has been implemented correctly and provide the expected level of security.

This application uses cryptographic library calls in 128 different places.

Figure 6: Ounce Labs Profile Report

identified vulnerabilities (in our review, Ounce Labs' reports generated measurements based on V-Density, a metric representing both the number and severity of flaws). Checklists of mandatory security policies and measurements of vulnerability levels should both be used as thresholds for the acquisition and deployment of applications. Many organizations currently use these as acceptance criteria in contracts or product releases, with a corresponding review process to serve as a 'gatekeeper'.

5.5 Trending Security over Time

Part of the security analysis for any application – regardless of whether it is commercial, custom, or open source – should include an evaluation of how security issues are addressed over time. The patching and update process for software vendors continues to be a much-debated topic; however, it is often overlooked for non-commercial application sources. Ideally, development organizations have controls in place to include proper security features and eliminate coding errors during development, but knowing that vulnerabilities are inevitable, a responsive remediation process is crucial.

Supporters of open source software often claim that applications they develop are more secure because of the large number of engineers that pour through each other's code. We tested this claim by analyzing security trends of open source applications, including Azureus, eMule, and LimeWire, to determine whether or not they exhibit improvements in security over time.

The Ounce Labs trend results showed a clear pattern in almost every test application of decreased vulnerabilities over the long term. The most common trend (an example of which is shown in Figure 7) was actually a growth in security flaws among the first few releases corresponding with a sharp increase in number of files and lines of code. It appears that after several early releases however, contributors turned their attention from features to security improvements, demonstrated by a reversal in the trend curve as vulnerabilities decreased while lines of code rose at a slower rate.

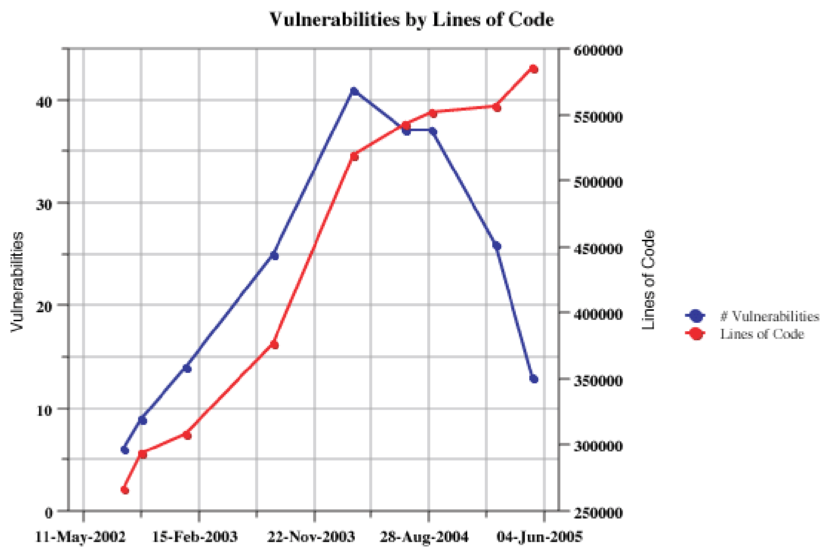


Figure 7: Sample Open Source Security Trend

Just as encouraging, the remediation efforts observed in these open source applications evidently focused on the high severity flaws before those identified as medium or low severity. A pattern one would hope to find in the remediation efforts of large enterprises, this shows that the open source projects we observed had a good understanding of criticality and logical workflow.

6 How to Establish an Organization-Wide Application Security Initiative

Companies do not simply inherit the ability or capacity to assure software meets security standards. It takes a concerted effort and a high level of support across all levels of the organization. Those that are successfully evaluating their software security and managing risk are those that set up reasonable priorities and make conscious improvements in the areas of people, process, and technology.

- People – training developers, testers, auditors, and management as well as standing up appropriate teams to address application security;
- Process – creating policies, methodologies, and best practices for developing, operating, and maintaining applications securely;
- Technology – establishing and tailoring standard coding guidelines, selecting and tuning tools (like those for application vulnerability scanning, penetration testing, and risk management), and defining and tailoring security libraries.

These changes should ultimately take place at the organizational level, though the application project level may provide a more manageable test run. Most important is that, whatever people, process, or technology improvements are put place, they must complement the company's current culture, methods, structures, and environments, including its existing development methodology and environment.

6.1 Building the Business Case for Software Security Assurance

Recognizing the importance of securing applications does not automatically mean that appropriate budget will be allocated for these efforts. Software security assurance will not attain the priority level it deserves until those responsible for protecting critical data and resources can make the business case that investments will result in acceptable return.

A solid business case provides the context for decision-making by accomplishing the following objectives:

- Explain the relationship with and benefits to existing projects
- Detail the investments required for people, processes, and technologies
- Define metrics and timeframe for success

For example, if a company is able to identify specific application security issues that have a significant impact on overall risk, the goal would be to trace them back to individual root causes. In this case, relatively small investments would result in significant reduction of risk and have a better chance of gaining budget approval. It is also important to consider the types of changes most likely to fit into the organization's current goals. While budget may not be available to hire additional security professionals to assess application vulnerabilities, better training or tools might have the same level of impact and fit within financial limitations.

This is just an example to demonstrate why gathering data is so important to making the right strategic investments in application security. Like the root causes, the possible improvements span teams, policies, processes, and technologies. When the right approach is utilized, the benefits will be measurable and justify their expense quickly.

6.2 Establishing a Roadmap and Measuring Success

When just starting an application security effort, the number of vulnerabilities may be overwhelming. Choosing the problem area on which to focus improvement efforts may seem secondary to getting the actual vulnerabilities fixed in your software, but this approach is short-sighted. If root causes are not addressed, the same problems will keep reoccurring and require eventual remediation later in the application lifecycle, when much more costly to eliminate.

Assuring software security in development, during acquisition, and after deployment requires gradual improvements across many layers of the organization. There is no single formula for success here, but there are specific activities that have proven to bring companies measurable success.

Below is an example roadmap that outlines potential efforts in a suggested order. This roadmap however, should be heavily customized and tailored based on the metrics, culture, processes, and structure of individual organizations.

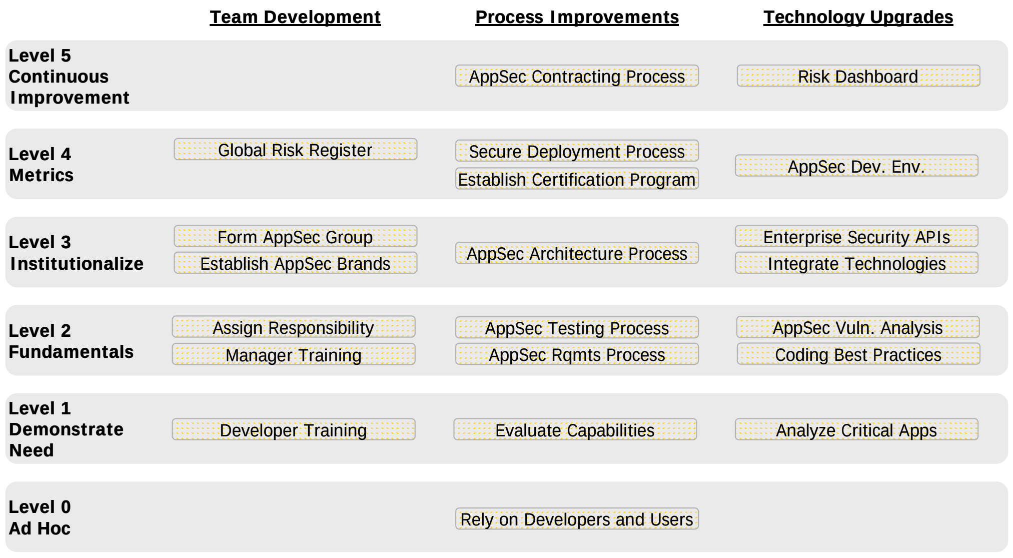
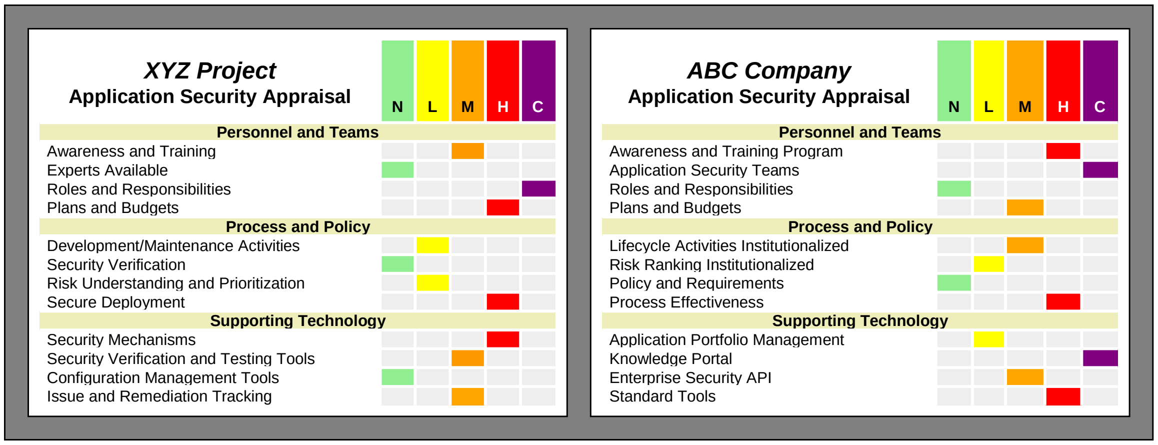


Figure 8: Application Security Roadmap

As the roadmap activities progress, it is critical to measure the effects of each effort. These measurements can range from very formal, clearly quantitative, and purely objective to somewhat informal, slightly subjective, and mostly qualitative. The most obvious measure is whether the number of security flaws in your software is improving. Initially, as your security verification capabilities improve, the number of discovered vulnerabilities may actually increase, but committed efforts should ultimately show long-term, measurable gains.

Ongoing efforts and results may be tracked in an existing process improvement framework if the organization has one implemented. Otherwise, tracking adherence to policies and results of vulnerability scans against security baselines can show positive return on investment over time.



XYZ Project
Application Security Appraisal

	N	L	M	H	C
Personnel and Teams					
Awareness and Training			🟧		
Experts Available	🟩				
Roles and Responsibilities					🟪
Plans and Budgets				🟥	
Process and Policy					
Development/Maintenance Activities		🟨			
Security Verification	🟩				
Risk Understanding and Prioritization		🟨			
Secure Deployment				🟥	
Supporting Technology					
Security Mechanisms				🟥	
Security Verification and Testing Tools			🟧		
Configuration Management Tools	🟩				
Issue and Remediation Tracking			🟧		

ABC Company
Application Security Appraisal

	N	L	M	H	C
Personnel and Teams					
Awareness and Training Program				🟥	
Application Security Teams					🟪
Roles and Responsibilities	🟩				
Plans and Budgets			🟧		
Process and Policy					
Lifecycle Activities Institutionalized			🟧		
Risk Ranking Institutionalized		🟨			
Policy and Requirements	🟩				
Process Effectiveness				🟥	
Supporting Technology					
Application Portfolio Management		🟨			
Knowledge Portal					🟪
Enterprise Security API			🟧		
Standard Tools				🟥	

Figure 9: Project and Organization
Application Security Appraisal Scorecards

The scorecard above demonstrates a simple method for tracking the level at which software assurance activity is operating, each area labeled as either: Note, Low, Medium, High, or Critical. Together, the organization-level scorecard and project scorecards across a sample of projects provide insight into security characteristics to help assign future priorities.

6.3 Initial Priorities: Getting Started

Beginning an application security initiative is often an extremely daunting task with no immediately clear process for success. The first step is to find out whether such an initiative is needed and create a business case. If the business case is convincing, then the following strategic people, process, and technology improvements can help to generate progress quickly.

- Building Support
  - **Verify Critical Applications** – There is no better way to get an overall impression of an organization's software security than to verify a few sample applications. Choosing high profile applications is often the best way to raise awareness of the problem and gain support for establishing an initiative.
  - **Train Developers, Security Reviewers, and Project Managers** – Provide training on application security for members of software development and security review teams. Developers, architects, and testers should be able to identify and address a broad range of application security issues. Project managers should understand the role of key application security activities and be able to measure and track application security on their projects.

After establishing the need for an application security initiative, the following activities have proven to be effective in a broad range of organizations. The goal is to establish a foundation for application security across an entire organization, so tailoring these activities to the culture, development process, and structure is critical to their success.

- People
  - **Assign Responsibility** – Ensure that on every project, someone has responsibility for application security. This role ensures that the requirements and architecture properly follow design requirements, the implementation is properly coded and tested, and that the deployment is performed securely. At any point, the application security architect should be able to defend the security of the application to information security teams, audit teams, legal teams, and customers.
  - **Establish an Application Security Team** – Establish a team of internal application security experts that can perform assessments throughout the organization. Ideally each project team would have an assigned application security expert available to help ensure that projects implement security requirements as specified and follow corporate standards for software development and implementation.

- Process
  - **Prioritize Applications** – Identify key characteristics that impact an application's associated risk level based on the overall business context. Establish a mechanism for assigning a risk profile to each application based on these characteristics. Using the risk profile, prioritize applications so that limited resources can be directed most effectively to improve the organization's overall application security.
  - **Establish Application Security Requirements and Testing Processes** – The most effective way for organizations to focus on application security is to be sure that the requirements are clearly specified at the beginning of the project lifecycle. Combined with effective security testing before deployment, most projects will successfully cover critical security areas. These activities will also produce metrics to guide further enhancements.

- Technology
  - **Establish Coding Guidelines and Standards** – Develop coding guidelines and deployment standards to provide project teams with essential guidance and direction. Move toward the development of standard, well-tested security mechanisms that can be reused across the organization.
  - **Introduce Appropriate Tools** – Invest in appropriate tools to support internal application vulnerability assessment efforts. These tools should support automated scanning, static code analysis as well as manual application analysis and testing. Additional tools facilitating capture and management of findings and dashboard views will enhance workflow and should be part of a fully integrated workbench.

7 CONCLUSIONS

The concept of risk assessment and acceptance are well-developed in most arenas of business operations, especially those with reliable metrics such as financial investments and insurance. Our detailed assessment of the Azureus application demonstrates that the level of information and metrics used for risk management decisions are evolving rapidly for application security as well. The process detailed above provides a model by which organizations should plan their own security assessments as part of development, procurement, and maintenance decisions. Aspect's application security verification process supported by free assessment tools and Ounce Labs' software security assurance technology found the Azureus application to exhibit a strong level of security. Our assessment of the open source process also generated positive results, demonstrating a long-term improvement in almost every application, including a primarily focus on eliminating high-severity vulnerabilities first.

The processes and techniques described in the report are viable initiatives currently employed at some of the country's largest corporations and government agencies. While source code review is especially common before the deployment of internally-developed, outsourced, and open source applications, many organizations have also mandated in contracts the review or proof-of-review for custom and commercial applications before final approval.

Close scrutiny of application security across all industries continues to grow among media, customers, partners, and auditors compelled by detailed regulations. This closer attention to security details is warranted by the increased reliance on open source and outsourced application components (even by commercial software vendors) as well as the determined application-layer threats from attackers with growing financial incentives.

The most effective way to prevent these increasing threats is to implement an application security initiative, the combination of people, processes, and technologies used to verify applications, track down vulnerability root causes, promote organizational improvements, and measure ongoing effectiveness. Organizations continue to see fewer security incidents, lower long-term software costs, stronger customer and partner loyalty, and more efficient regulatory compliance by taking this proactive approach to protecting critical data and assets.