



## Reasoning Inspection Services

It is possible for software defects and security vulnerabilities to go undetected through the software development and testing phases, particularly when they do not demonstrate easily recognizable symptoms.

When defects and vulnerabilities remain dormant, they can manifest at customer sites, or in deployment, and evolve into costly support and security issues. The longer they remain unidentified, the greater the potential for incurring tremendous cost, effort, and time.

“We established four criteria for success in this project – two relating to target numbers of defects that needed to be found, one relating to the likelihood of the defects becoming apparent in the field, and one relating to return on investment. Reasoning’s automated inspection service met or exceeded our expectations in all four areas.”

**Bob Ellis**  
Engineering Director  
Passenger Division  
Rockwell Collins

Software inspection (code review) has long been known to be the single most effective way to identify defects in software, but is rarely broadly used because it has been an entirely manual process, requiring significant time from the most experienced (and hence expensive) engineers.

**Alternative to Manual Process** • Fortunately, automated inspection techniques are available that make it practical to inspect every line of source code during any phase of development or testing to quickly improve the security, integrity and reliability of the produced code.

**Overcomes Limitations of Testing** • Automated inspection and assessment services augment existing software testing solutions, which are restricted by the need to execute code, build, manage, and run test cases, while testing only for expected behavior. Designed to complement a structured testing process, our services provide an essential path to ensuring software security and reliability over time.

**Early Detection Increases Productivity, Lowers Cost** • Reasoning’s automated software inspection and assessment services help you reduce risks by detecting and diagnosing vulnerabilities and defects well before they become discernable problems, and provide the exact location and root-cause for remedy and resolution. In addition, Reasoning boosts the productivity of software development organizations by finding security vulnerabilities faster, earlier, and at a far lower cost than traditional approaches. We provide many of the benefits of a manual code review, but in significantly less time and at dramatically lower cost.

### Integrates Smoothly with Existing Process •

Reasoning services are hardware independent, do not require running code or test cases, and do not affect code size or execution speed. Because the solution is outsourced, there is no training required.

Our services enhance an organization's existing QA processes, and bolster efforts to provide secure, reliable, maintainable software. As a result, Reasoning enables fast return on investment and permits software developers to focus on their core competency – software development. Reasoning offers Security and Reliability Inspection Services.

**Security Inspection Service •** For software development organizations coding in C and C++, Reasoning helps eliminate critical security risks and provides enterprises with an effective weapon against unauthorized access and hacker attacks. Our application-level security inspection finds potential

problems missed by existing developer solutions, such as application scanning and dynamic testing tools, which can only test for the expected behavior of hackers.

In addition, we provide the exact location and root cause of vulnerabilities, making it easy for developers to isolate and resolve issues quickly and effectively – before an application is put into production.

Reasoning produces security data reports that make identification, analysis, and repair easy to accomplish. The reports serve as detailed roadmaps that clearly list the class and location of vulnerabilities, along with a full description.

The Metrics Report is designed for the management team to provide insight into problem areas within an application, and including industry comparisons and ratings.

**DEFECT CLASS** Memory Leak

**LOCATION** \websrv\_1.1\src\os\win32\readdir.c : 43

**DESCRIPTION** Local variables **dp** and **filespec**, declared on lines **22** and **23**, are assigned pointers to blocks of memory allocated by **malloc()** on lines **34** and **27**. No other pointer refers to these memory blocks, so they are inaccessible (still allocated, but unreachable) once **dpr** and **filespec** go out of scope after line **43**.

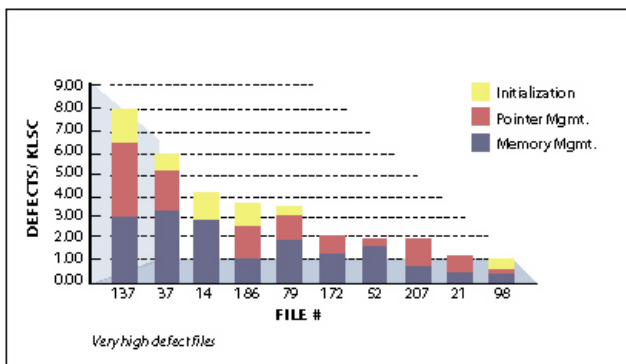
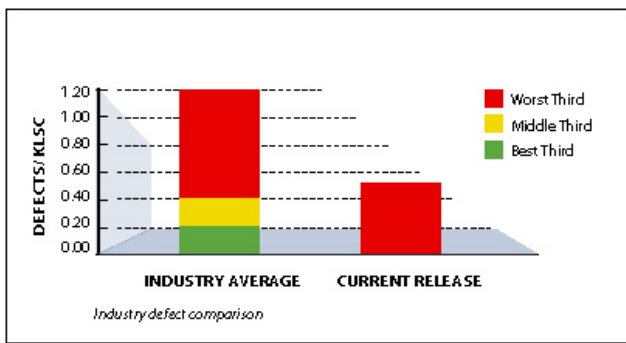
**PRECONDITIONS** The condition `((handle = _findfirst(filespec, &(dp->fileinfo))) < 0)` on line **39** evaluates to true, AND `(errno == ENOENT)` on line **40** evaluates to false.

#### CODE FRAGMENT

```
20 API_EXPORT(DIR *) opendir(const char *dir)
21 {
22     DIR *dp;
23     char *filespec;
24     long handle;
25     int index;
26
27     filespec = malloc(strlen(dir) + 2 + 1);
28     strcpy(filespec, dir);
29     index = strlen(filespec) - 1;
30     if (index >= 0 && (filespec[index] == '/' || filespec[index] == '\\'))
31         filespec[index] = '\0';
32     strcat(filespec, "/*");
33
34     dp = (DIR *)malloc(sizeof(DIR));
...
39     if ((handle = _findfirst(filespec, &(dp->fileinfo))) < 0) {
40         if (errno == ENOENT)
41             dp->finished = 1;
42         else
43             return NULL;
44     }
```

**Vulnerabilities Reported** • These broad classes of security vulnerabilities are reported by Reasoning's Security Inspection Service:

- **Buffer Overflows** On the SANS/FBI Top 20 list of Internet security vulnerabilities, buffer overflows are the most common security flaw exploited by hackers. By sending more data than the software is designed to handle, hackers gain unauthorized access.
- **Race Conditions** This term describes time lapses between the verification that a planned operation is safe and the execution of the operation itself. The period between verification and execution is vulnerable to exploitation by hackers.
- **Tainted Data** Whenever a software program obtains data from the outside world, it needs to validate that the data is within the design specifications of the program. When data is not validated, it is called "tainted." The use of tainted data may cause programs to perform operations that do not conform to their original design.
- **Risky Operations** This category consists of sub-classifications of security vulnerabilities such as the use of weak random number generators, the use of poor temporary file names, and the loading and or execution of external programs or libraries.



**Reliability Inspection Service** • Reasoning's Reliability Inspection Service focuses on identifying Java, C, and C++ software defects that cause crashes, corrupt data, and create unexpected behavior—all of which are potentially application stoppers. The Reasoning solution seamlessly reviews the entire available code base, pinpoints logic defects that reduce reliability and identifies the areas of highest risk.

Reasoning delivers defect data reports that make defect identification, analysis, and repair easy to accomplish. The reports serve as detailed roadmaps that clearly list the class and location of defects, the conditions under which they will happen, and a full description of defects down to the code fragment.

Companion defect metric reports, designed for managers, that compare the organization's code quality to industry averages, identify areas of code with the highest risk, and measure code quality and trends.

**Reliability Defects Reported** • Defects are reported in these broad classes for Java, C and C++:

- **Resource leaks** Java file descriptor, socket handle, and database resource leaks are critical security-related defects that can be potentially exploited for the purpose of denial-of-service attacks, and can also cause applications to exhibit unreliable behavior or sudden failure.
- **Memory leaks** Memory leak refers to the loss of available memory space that occurs when dynamic data is no longer used, but is never de-allocated. Each time the leak occurs, the application drains the available memory pool. On some systems, memory may be allocated from a global pool, in which case the loss of available memory may affect the entire system. Eventually, the degradation may lead to a fatal out-of-memory condition.
- **NULL pointer dereferences** Occur when a null variable is accessed; they can cause issues from resource leaks to application failure.
- **Out of bounds array access defects** Occur when an array index expression is not within the upper and lower bounds of the array; they can result in data corruption and application failure.
- **Uninitialized variable** With this defect, local and dynamic variables are not explicitly initialized prior to use. Usage of uninitialized variables can cause unpredictable results in the program, because the value of the variable is essentially random.



- **Dangling pointer** A dangling pointer refers to a defect in which a pointer is dereferenced after the memory that it points to has been deallocated. This defect can cause application crashes, unpredictable behavior and data corruption.
- **Bad deallocation** Bad deallocation uses an inappropriate memory release operation for deallocating memory, or to the deallocation of memory that was never explicitly allocated. Depending on the compiler and the specific operation, the impact of this defect can range from an application crash, to unexpected results, a memory leak, or memory corruption.
- **String comparison defects** An erroneous string compare occurs when two Java string objects are compared using either the "=" or "!=" operators, instead of the String comparison methods available

on the String object. If Strings are not properly compared, the software program will not execute as designed. What is especially important to note about these defects is that there is often no indication that a problem exists

**About Reasoning** • Reasoning is a pioneer in the field of software static analysis technology and the leading provider of automated software inspection services. Since 1996, Reasoning has focused on software quality needs for hundreds of customers and examined over a billion lines of source code. Reasoning serves customers around the world from headquarters in Menlo Park, California. The company is privately held.



*For more information, contact:*

**Reasoning, LLC** 650 324-2510 (phone)  
PO Box 478 415 763-1992 (fax)  
Menlo Park, CA 94026-0478 Email: [reasoninginfo@reasoning.com](mailto:reasoninginfo@reasoning.com)