NASA/CR–2017-219024

Security Vulnerability Profiles of NASA Mission Software: Empirical Analysis of Security Related Bug Reports

Katerina Goseva-Popstojanova, PI
Jacob Tyo
Brian Sizemore

West Virginia University, Morgantown, WV 26506, USA

January 2017

NASA/CR–2017-219024

Security Vulnerability Profiles of NASA Mission Software: Empirical Analysis of Security Related Bug Reports

Katerina Goseva-Popstojanova, PI
Jacob Tyo
Brian Sizemore

West Virginia University, Morgantown, WV 26506, USA

National Aeronautics and
Space Administration

January 2017

Acknowledgments

This work was funded by the NASA Software Assurance Research Program (SARP) in FY-2016. The authors would like to thank Brandon Bailey, Craig Burget, and Dan Painter for their input, feedback, and support.

Executive Summary

NASA develops, runs, and maintains software systems for which security is of vital importance. Therefore, it is becoming an imperative to develop secure systems and extend the current software assurance capabilities to cover information assurance and cybersecurity concerns of NASA missions.

The research work presented in this report was funded by the NASA Software Assurance Research Program (SARP). The results are based on the information provided in the issue tracking systems of one ground mission and one flight mission. The extracted data were used to create three datasets: Ground mission IV&V issues, Flight mission IV&V issues, and Flight mission Developers issues. In each dataset, we identified the software bugs that are security related and classified them in specific security classes. This information was then used to create the security vulnerability profiles (i.e., to determine how, why, where, and when the security vulnerabilities were introduced) and explore the existence of common trends, with specific emphasis on identifying the dominant types of vulnerabilities.

The main findings of our work include:

- Code related security issues dominated both the Ground and Flight mission IV&V security issues, with 95% and 92%, respectively. Therefore, enforcing secure coding practices and verification and validation focused on coding errors would be cost effective ways to improve mission's security. (Flight mission Developers issues dataset did not contain data in the Issue Category.)

- In both the Ground and Flight mission IV&V issues datasets, the majority of security issues (i.e., 91% and 85%, respectively) were introduced in the Implementation phase. This was expected, having in mind that the overwhelming majority of security issues were code related. In most cases, the phase in which the issues were found was the same as the phase in which they were introduced. The most security related issues of the Flight mission Developers issues dataset were found during Code Implementation, Build Integration, and Build Verification; the data on the phase in which these issues were introduced were not available for this dataset.

- The location of security related issues, as the location of software issues in general, followed the Pareto principle. Specifically, for all three datasets, from 86% to 88% the security related issues were located in two to four subsystems.

- The severity levels of most security issues were moderate, in all three datasets.

- Out of 21 primary security classes, five dominated: Exception Management, Memory Access, Other, Risky Values, and Unused Entities. Together, these classes contributed from around 80% to 90% of all security issues in each dataset. This again proves the Pareto principle of uneven distribution of security issues, in this case across CWE classes, and supports the fact that addressing these dominant security classes provides the most cost efficient way to improve missions' security.

The findings presented in this report uncovered the security vulnerability profiles and identified the common trends and dominant classes of security issues, which in turn can be used to select the most efficient secure design and coding best practices compiled by the part of the SARP project team associated with the NASA's Johnson Space Center. In addition, these findings provide valuable input to the NASA IV&V initiative aimed at identification of the two 25 CWEs of ground and flight missions.

Contents

1 Introduction **1**

2 Key Terms **2**

3 Related Work **3**

4 Vulnerability Profiles **5**
- 4.1 Description of the Datasets 5
- 4.2 Classification Schema 6
- 4.3 Classification Approach 8
- 4.4 Ground Mission IV&V Issues 8
- 4.5 Flight Mission IV&V Issues 13
- 4.6 Flight Mission Developers Issues 19

5 Comparison of the Results **24**

6 Threats to Validity **26**

7 Conclusion **27**

1 Introduction

NASA's portfolio of IT assets includes systems that control spacecrafts, collect and process scientific data, and enable NASA personnel to collaborate with colleagues around the world [1]. NASA currently makes extensive use of domestic and international internet connectivity. The use of public internet instead of previously used dedicated circuits, on one side, resulted in significant cost savings and large increases in performance. On the other side, it exposed NASA systems to cyberattacks. Based on the information provided by the Office of the Chief Information Officer [2], in a typical week NASA experiences 29,000 malicious incidents against its systems, 17,500 suspicious e-mails, and 250 unique incidents against its Web sites. Among these are sophisticated cyberattacks known as advanced persistent threats (APTs) [3]. The individuals behind APTs are typically well organized and well-funded and often target high profile organizations like NASA. Examples include the 2007-2008 hacking attempts against NASA's Landsat-7 and Terra AM-1 satellites. The hacker used the internet connection to get into the ground station's information system [4]. Although the responsible party achieved all steps required to command the satellite, the commands were not sent to the satellites.

Nowadays, space missions provide valuable services to the society - from navigation, to earth observation, weather forecasting or communication. Consequently, space is becoming another critical element of the US infrastructure. Cyber threats to space missions are expected to continue to grow in the future [5]. Even more, increased complexity of missions, coupled with ambitious operational scenarios and cooperation between government agencies and commercial enterprises, are likely to lead to increased number of security vulnerabilities. NASA's multi-tiered approach that aligns cybersecurity management to mission assurance integrates the IT service security, Data and Application security, and Infrastructure security [2]. This report is focused on Application security throughout the software life cycle. This is an important aspect of the overall cybersecurity because once an attacker has gained access to internet-accessible computer, he/she could use the compromised computer as a means to exploit vulnerabilities on mission computers that could significantly disrupt NASA's space flight operations and/or steal sensitive data [1]. Therefore, it is becoming an imperative to extend the current software assurance capabilities to cover information assurance and cybersecurity concerns of NASA missions.

The goal of our research work funded by the NASA Software Assurance Research Program (SARP) was to build security vulnerability profiles of mission ground and flight software, and to integrate the findings with the Secure Design and Coding Best Practices developed by part of the SARP project team associated with the NASA's Johnson Space Center (JSC) [6]. These Secure Design and Coding Best Practices are aimed to reduce the number of vulnerabilities and improve missions' resilience to cyberattacks.

A *security vulnerability* is defined as a weakness in a system, application, or network that could be subject to exploitation or misuse that would allow an attacker to compromise any aspect of cybersecurity (i.e., confidentiality, integrity, availability, authentication, authorization, and non-repudiation). Our research is based on utilizing the information provided in issue tracking systems of pilot NASA missions to explore the vulnerability landscape, that is, to identify the vulnerability profiles of ground and flight software, with specific emphasis on identifying the dominant types of vulnerabilities. Whenever possible, our study assessed the vulnerabilities throughout the lifecycle.

In general, a *profile* is defined as "a set of data portraying the significant features of something." In this work, we introduce the term *security vulnerability profile* which is defined as a set of data that allows NASA missions to determine how, why, where, and when the security vulnerabilities were introduced. Uncovering the security vulnerability profiles and underling trends helps developers and IV&V analysts to focus their efforts on preventing and eliminating the vulnerabilities in the most effective ways, at the most effective time. The results presented in this report are based on data extracted from three NASA issue tracking systems, which were used to create three datasets. In each datasets, we identified the software bugs that are security related and classified them in specific security classes. This information was then used to create the security vulnerability profiles and explore the existence of trends. Our main research questions are as follows:

RQ1 What are the dominating types of vulnerabilities in NASA ground and flight software systems?

RQ2 Are dominating types of vulnerabilities consistent across missions and mission types?

The rest of this report is organized as follows. In main terms used in this report are defined in section 2. Section 3 summarizes the related works on characterization of software bugs in general and software security vulnerabilities in particular. Section 4 presents the description of the datasets, used classification schema and classification approach, and the results on security vulnerability profiles. In section 5 we compare the results across the three datasets and identify the dominating types pf vulnerabilities and common trends. The threats to validity are enumerated in section 6 and the report is concluded in section 7.

2 Key Terms

This section introduces the terminology used in the report related to software quality and cybersecurty fields.

A **failure** is a departure of the system or system component behavior from its required behavior. A **fault** is an accidental condition, which if encountered, may cause the system or system component to fail to preform as required. Thus, faults represent problems that developers see, while failures represent problems that the users (human or computer) see. Not every fault corresponds to a failure since the conditions under which fault(s) result in a failure may never be met. It should be emphasized that faults can be introduced at any phase of the software life cycle, that is, they can be tied to any software artifact (e.g., requirements, design, and source code). Throughout this report we use the terms 'fault' and 'bug' interchangeably. Note that in some cases the term 'defect' is used to refer only to faults, but in other cases to refer collectively to faults and failures. To avoid confusion, the term 'defect' is not used in this report.

A **security vulnerability** is defined as a weakness in a system, application, or network that could be subject to exploitation or misuse that would allow an attacker to compromise any aspect of cybersecurity (i.e., confidentiality, integrity, availability, authentication, authorization, and non-repudiation). This work is focused on security vulnerabilities in software applications, specifically in case studies based on NASA missions' ground and flight software. We define a **security vulnerability profile** as a set of empirical data that allows software projects to determine how, why, where, and when the security vulnerabilities were introduced into the system.

An **issue tracking system** is a software application that allows a project to record and follow the progress of every **issue** that developers and/or software system users identify until the issue is resolved. Issues can belong to multiple categories, such as (software or hardware) bugs, improvements, and new functionality. We believe that issue tracking systems are valuable sources of information that hold high potential for conducting empirical studies that benefit both the practitioner and research communities. In this report, we are focused on studying a subset of software issues related to software bugs and specifically identifying and studying software bugs that are security vulnerabilities. Note that the terms **vulnerability**, **security issue**, and **security related bug** are used interchangeably in this report.

The **Common Weakness Enumeration (CWE)** taxonomy aims at creating a catalog of software weaknesses and vulnerabilities. It is maintained by the MITRE Corporation with support from the Department of Homeland Security [7]. Each individual CWE represents a single vulnerability type or category. For example, CWE 121 is "Stack-based Buffer Overflow", CWE 78 is "OS Command Injection" and so on. The CWEs are organized in a hierarchical structure with broad category CWEs at the top level. These top level CWEs may have multiple children. Each child CWE may have one or more parents and zero or more children. The further down this hierarchy one goes the more specific the vulnerabilities become. A **CWE View** is a particular perspective (or view) to the CWE taxonomy. Different views organize, categorize, or group individual CWEs within the overall taxonomy differently.

Common Vulnerabilities and Exposures (CVE) stands for a dictionary of common names for publicly known cybersecurity vulnerabilities and exposures. CVE's common identifiers are assigned by CVE Numbering Authorities from around the world, with a goal to support data exchange between security products and provide a baseline index point for evaluating coverage of tools and services [8]. Note that CVE is sponsored by US-CERT in the Office of Cybersecurity and Communications at the U.S. Department of Homeland Security.

National Vulnerability Database (NVD) is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol. NVD includes databases of security checklists, security related software flaws, misconfigurations, product names, and impact metrics [9].

3 Related Work

Studies focused on software faults can be dated back to the mid 1970s. However, there continues to be a dearth of published data relating to the quality and reliability of realistic commercial software systems. Even less studies have been focused on the types and characteristics of software security vulnerabilities.

First, we summarize the papers that explored the characteristics of software faults in general [10–19]. Fenton and Ohlsson empirically studied a large software-intensive telecommunication application from Ericsson Telecom AB [10]. This work was focused on a range of software engineering hypotheses related to the Pareto principle of distribution of faults and failures, the use of early fault data to predict later fault and failure data, and metrics for fault prediction. The results confirmed that small number of modules contain most of the faults discovered in pre-release testing and that a very small number of modules contain most of the faults discovered in operation. However, the fault proneness of the modules could not be explained using their size or complexity. The authors claimed that the most surprising result was that the most fault-prone modules pre-release were among the lease fault-prone post-release.

Our previous research work, which was based on data extracted from a large NASA mission with over 8,000 files and two millions lines of code, was focused characterizing and quantifying relationships among faults, failures and fixes. Over 1,200 failures, recorded during development and operation in a period of over ten years, were extracted from the change tracking system. The fact that the mission kept detailed records on the changes made to fix fault(s) associated with each failure allowed us to close loop from failures to faults that caused them and changes made to fix the faults. Our results showed that software failures are often associated with faults spread across multiple files [11]. Our results further showed that a significant number of software failures required fixes in multiple software components and/or multiple software artifacts (i.e., 15% and 26%, respectively), and that the combinations of software components that were fixed together were affected by the software architecture [12]. More recently, we studied the types of faults that caused software failures, activities taking place when faults were detected or failures were reported, and the severity of failures [13]. Our results showed that components that experienced more failures pre-release were more likely to fail post-release and that the distribution of fault types differed for pre-release and post-release failures. Interestingly, both post-release failures and safety-critical failures were more heavily associated with coding faults than with any other type of faults. Last but not least we systematically explored the effort associated with investigating and reporting the failures (i.e., investigation effort), as well as the effort associated with implementing the fix to correct all faults associated with an individual failure (i.e., fix implementation effort), with special focus on factors that affect them [14]. We also proposed a data mining approach for predicting the level of fix implementation effort using the data provided in the software change requests, when the failure was reported.

Another empirical study based on space mission software data conducted by Grottke et al. analyzed 520 anomalies, each of which represented a unique fault in the flight software of eighteen JPL space missions [15]. The authors defined Bohrbugs as bugs that are easily isolated and removed during software testing, and Mandelbugs as bugs that appear to behave chaotically, and reported that 61% of bugs were Bohrbugs and 37% were Mandelbugs. Furthermore, they determined that there is a significant relationship between the fault type and the failure risk. Alonso et al. analyzed the mitigation associated with the Bohrbugs and Mandelbugs and concluded that both types of bugs were most frequently mitigated via fixes instead of other measures such as proactive reboots [16]. In addition, earlier missions had lower frequencies of fixes/patches than the more recent missions.

Several papers explored some aspects of software bugs for different application domains. Motivated by the increasing popularity and capabilities of open source operating systems for hand-held devices, Maji et al. utilized bug reports, bug fixes, developer reports, and failure reports to explore the manifestation of failures in Android [17]. This work was focused on the frequency of failures and the persistence of faults. The results showed that over 90% of the faults were permanent in nature, the kernel layer was sufficiently robust but much effort is needed to improve the middleware layer. Furthermore, it was found that between 11% and 50% of bugs were due to the customizability of Android and that most bugs required only minor changes for correction (e.g., update of configuration parameters). Frattini et al. analyzed a small dataset of 146 bug reports from Apache Virtual Computing Lab, which is an open source cloud platform [18]. This analysis identified the components where bugs were likely to be found in future releases, the phases of the life cycle during which such bugs may be discovered, and the modification required to fix them. Xia et al. utilized the bug databases and code repositories for open source software applications Ant, Maven, CMake, and QMake containing 199, 250, 200, and 151 bug reports respectively [19]. Each sample was manually classified into several categories and the results showed that 21% of bugs belonged to the external interface category, 18% belonged to the logic category, and 13% belonged to the configuration category.

The number of studies that were specifically focused on studying software vulnerabilities is even smaller and includes only several works [20–22]. Alhazmi et al. used density of vulnerabilities, fraction of software bugs that are security related, the dynamics of vulnerability discovery, and the vulnerability discovery rate to estimate the magnitude of the undiscovered vulnerabilities still present in the system [20]. The analysis was based both on commercial and open-source systems and the results revealed that the vulnerability densities fell within a range of values, similar to fault density for general faults. The authors also investigated if it is possible to predict the number of vulnerabilities that can potentially be present in a software system, but may not have been found yet and claimed that the vulnerability discovery can be modeled using logistic model, which can sometimes be approximated by a linear model as a function of time. Venter et al. proposed an approach aimed at standardization of vulnerability category using self-organizing maps (SOMs) and data extracted from CVE [21]. This work, however, did not provide quantification of the results. Younan analyzed the vulnerabilities reported from 1998 to 2012 [22] based on the CVE information provided in the NVD databases. One of the finding was that despite the progress in mitigating attacks against buffer overflows, they remain one of the top ranking vulnerabilities. Furthermore, it appeared that while fewer vulnerabilities were reported in the last couple of years, the percentage of more critical vulnerabilities has increased. Results also showed that Microsoft products have improved significantly within the last couple of years and their browser and mobile operating systems were better than their competitors' in terms of vulnerabilities discovered. When it comes to mobile OS, iPhone had significant lead in vulnerabilities, while Android had very few. On the contrary, Safari had the fewest vulnerabilities compared to the other browsers, while Chrome ranked as one of the highest for vulnerabilities.

4 Vulnerability Profiles

In this section, we first describe the datasets used in this study and then present the rationale behind the choice of the classification schema and the approach used for manual classification (i.e., labeling) of software issues, followed by the analysis of the vulnerability profiles of the three datasets.

4.1 Description of the Datasets

The three main datasets from NASA utilized for this work were created by extracting relevant information from a ground mission IV&V issues, flight mission IV&V issues, and flight mission developers issues. For all three datasets only the "closed" issues from their corresponding issue tracking systems were included. The details about these datasets are provided next.

The first dataset was extracted from the IV&V issue tracking system of a NASA ground mission and in this report is referred to as *Ground mission IV&V issues.* The ground mission software consist of approximately 1.36 million source lines of code and the issue tracking system contained 1,779 issues created over four years. Since this is a recent mission, the IV&V analysts specifically considered the impact of each issue on security, and as a result 350 (i.e., 20%) of the issues were marked as potentially security related. Most issues contained very detailed descriptions, titles, and comments. In addition, the issue descriptions contained security related information, making this a very good dataset for our research. The fields in the IV&V issue tracking system are detailed in Appendix A, Table 4.

The second dataset consists of the IV&V issues extracted from the issue tracking system of a NASA flight mission and is referred to as *Flight mission IV&V issues*. The flight mission software had approximately 924 thousand source lines of code, and the issue tracking system contained 506 issues created over four years. After removal of issues marked as "Withdrawn" or "Not an Issue," 383 issues remained. Although this dataset was also created by IV&V analysts, security aspects of issues were not specifically and consistently considered. Consequently, issue descriptions contained very little security related information. Rather, descriptions were mainly focused on system operation. The Flight mission IV&V issues were extracted from the same tracking system as the Ground mission IV&V issues and therefore had the same fields shown in Table 4.

The third dataset consists of developer issues extracted from the issue tracking system of the same NASA flight mission as flight mission IV&V issues and is referred to as *Flight mission Developers issues*. This issue tracking system consisted of 1,947 Developer Change Requests (DCRs) created over five and a half years. The analysis presented in this report is based on 573 of these DCRs that were marked as "Defects". (The others issues were marked as "Change Requests" or some other non bug related category, and are not included in the analysis.) This dataset originated from the developers instead from the IV&V analysts, resulting in much greater focus on software operation than security aspects. The fields of this issue tracking system are detailed in Appendix A, Table 5.

4.2 Classification Schema

In order to classify the issues, a classification schema is needed. Common problems among classification schemas are undefined levels of specificity, very complicated structure, and non-hierarchical structure. This section explores several software vulnerability and/or weakness classification schemas along with their strengths and weaknesses, which are then used to select a classification schema for software vulnerabilities.

Common Weakness and Enumeration (CWE) is a taxonomy of software weakness types aimed at serving as a common language for describing software security weaknesses in architecture, design, or code [7]. The CWE serves as a standard measuring stick for software security tools targeting these weaknesses, and to provide a common baseline standard for weakness identification, mitigation, and prevention efforts [7]. Each individual CWE represents a single vulnerability type or category. The CWEs are organized in a hierarchical structure with broad category CWEs at the top level. The further down this hierarchy, the more specific the vulnerabilities become. The CWE taxonomy has 1004 CWEs and rather complex structure; each CWE may have one or more parents (expect the top level CWEs) and zero or more children. Therefore, using the complete CWE taxonomy for classification of software vulnerabilities is not very practical, which is the reason why a number of views (as defined in Section 2) have been developed to ease the grouping of similar CWEs and provide simpler, more generalized structure. Next we discuss several CWE views: CWE-2000, CWE-1000, CWE-888, CWE-700 and CWE-699.

CWE-2000 is the Comprehensive CWE Dictionary View which covers all elements in CWE. It contains 1004 total CWEs with no particular grouping or classification [23]. CWE-

2000 offers no classifying advantage as there is no attempt to ease the complex structure and large number of CWE categories. Therefore, this view is not suitable for use as classification schema in our work.

CWE-1000 is the Research Concepts view and was created with the intent to facilitate research of weaknesses, including their inter-dependencies [24]. It classifies weaknesses in a way that largely ignores how they can be detected, where they appear in code, and when they are introduced in the software development life-cycle. Instead, it is mainly organized according to the abstractions of software behaviors. It uses a deep hierarchical organization which provides many levels of abstraction and specificity. CWE-1000 explicitly identifies relationships that form chains and composites, which have not been a formal part of past classification efforts. This classification schema contains a total of 723 CWEs, grouped into 11 main classes. One drawback of this classification schema is the deep hierarchical structure and the fact than the level of specificity is not the same across all depths.

CWE-888 is the Software Fault Pattern (SFP) view and is a classification schema developed by the Department of Defense (DoD) sponsored project through KDM Analytics [25,26]. This view provides a formal specification of software weaknesses/vulnerabilities that enable automation through focusing on characteristics that are discernible in code, while also ensuring systematic coverage of the weakness space. The classification schema contains 705 CWEs, grouped into 21 primary and 62 secondary classes. Furthermore, every CWE within this view is classified to exactly one primary and one secondary class, creating a three level hierarchical view. The CWE-888 structure does not have the specificity problem of CWE-1000. The three levels have well defined levels of specificity, with the primary class being the most general, and the third level (individual CWEs) being the most granular. This, along with the very intuitive structuring, makes the CWE-888 view a very good classification schema for our work.

CWE-700 is the Seven Pernicious Kingdoms View which originated from Cigital [27,28]. The creators argued that other security taxonomies are too complex and, motivated by the belief that people on average are good at keeping track of seven (plus or minus two) things, created a taxonomy with only seven primary categories. CWE-700 follows a two level hierarchical structure and contains only 97 CWEs, grouped into seven primary classes. It appears that this schema is too simplistic and not sufficiently specific for use in our work.

CWE-699 is the Development Concepts view which organizes weaknesses around concepts that are frequently used in software development [29]. Accordingly, this view aligns closely with the perspectives of developers, educators, and assessment vendors. It borrows heavily from the organizational structure used by Seven Pernicious Kingdoms, but it also provides a variety of other categories that are intended to simplify navigation, browsing, and mapping. This classification scheme contains 756 CWEs, organized in a hierarchical structure similar to CWE-1000. There are six primary classes, which contain different number of CWEs, subclasses, or nested subclasses. This schema, however, has the same specificity problem as CWE-1000 and therefore is not suitable for our work.

Based on the above discussion, we selected CWE-888 Software Fault Pattern (SFP) View as classification schema of NASA mission software vulnerabilities. CWE-888 includes a three level hierarchical structure of which the first two levels (primary and secondary classes) were used for our classification. Namely, each security related software issue was assigned a primary (more general) class and a secondary (more specific) class. The primary and secondary classes of CWE-888 are given in Table 6 in Appendix 7. The specific CWEs and their descriptions can be found at [25].

4.3 Classification Approach

In order to be able to build the vulnerability profiles, for each of the three datasets we first needed to manually classify (i.e., label) each software issue into one of the CWE-888 primary and secondary classes if it was security related or as non-security related. We did the labeling based on the information provided in the following fields from the issue tracking systems: "Title," "Subject," "Description," "Recommended Actions," and "Solution."

Several examples of classification of issues to the CWE-888 primary and secondary classes are as follows:

- An issue with description "...Line 277: Null pointer dereference of 'getServiceStatusInfo(...)' where null is returned from a method," was labeled with the primary cluster of "Memory Access" and the secondary cluster of "Faulty Pointer Use."
- An issue with description "...The stream is opened on line 603 of file1. If an exception were to occur at any point before line 613 where it is closed, then the 'try' would exit and the stream would not be closed," was labeled with the primary class of "Resource Management" and the secondary class of "Failure to Release Resource."
- An issue with description "...Table 1-11 lists XYZ as a unidirectional interfaces, but Figure 1-4 shows this connection as bidirectional," was labeled as non-security related.

Similarly to the labeling done by static code analysis tools, we adopted a conservative labeling (i.e., classification) approach that treats as security related every issue to which a CWE class could be assigned. Note that we did not have access to the code and other necessary information to determine if the security related issues could be exploited or what the overall security related impact on the system would be. These aspects are out of the scope of our work.

Upon completion of the labeling, we analyzed each dataset. The results are detailed in the following sections.

4.4 Ground Mission IV&V Issues

Of the 1,779 issues in this dataset, 350 (20%) were tagged as potentially security related by the IV&V analysts. After labeling, it was determined that 133 of the 350 (38%) could be assigned a specific CWE. The remaining security issues were tagged by the IV&V analysts as testing issues; they were related to problems with the testing system instead of problems with the mission software being tested. Since no CWEs exist that cover such a case and testing issues are not dealing with the the actual system under investigation, these issues were not considered in the further analysis of security related issues.

Figure 1 shows the distribution of security and non-security issues across the different Issue Categories (i.e., Concept, Requirements, Design, Code, Test). As shown, the Code category contained 95.5% of all security issues (i.e., 127 out of 133). Even though the Design category had the highest number of issues, only 2.3% (i.e., three out of 133) of all security issues belonged to this category.

Figure 2 shows the distribution of security and non-security issues across different Issues Types, which provide more detailed categorization than the Issue Category. Four dominating issue types were Incomplete Design, Incomplete Code, Incorrect Code, and Incomplete Test Article. Together, the code related issues types Incomplete Code and Incorrect Code contained 84% (112 of 133) of security related issues.

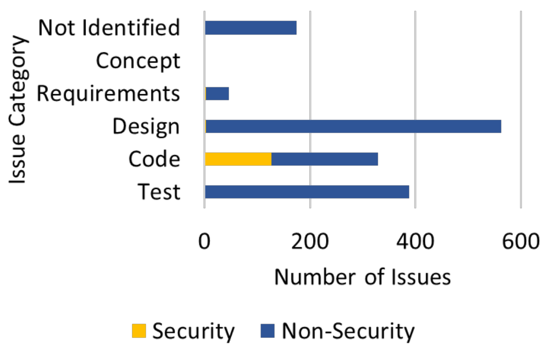

Figure 1. Ground Mission IV&V Issues - Distribution across Issue Categories

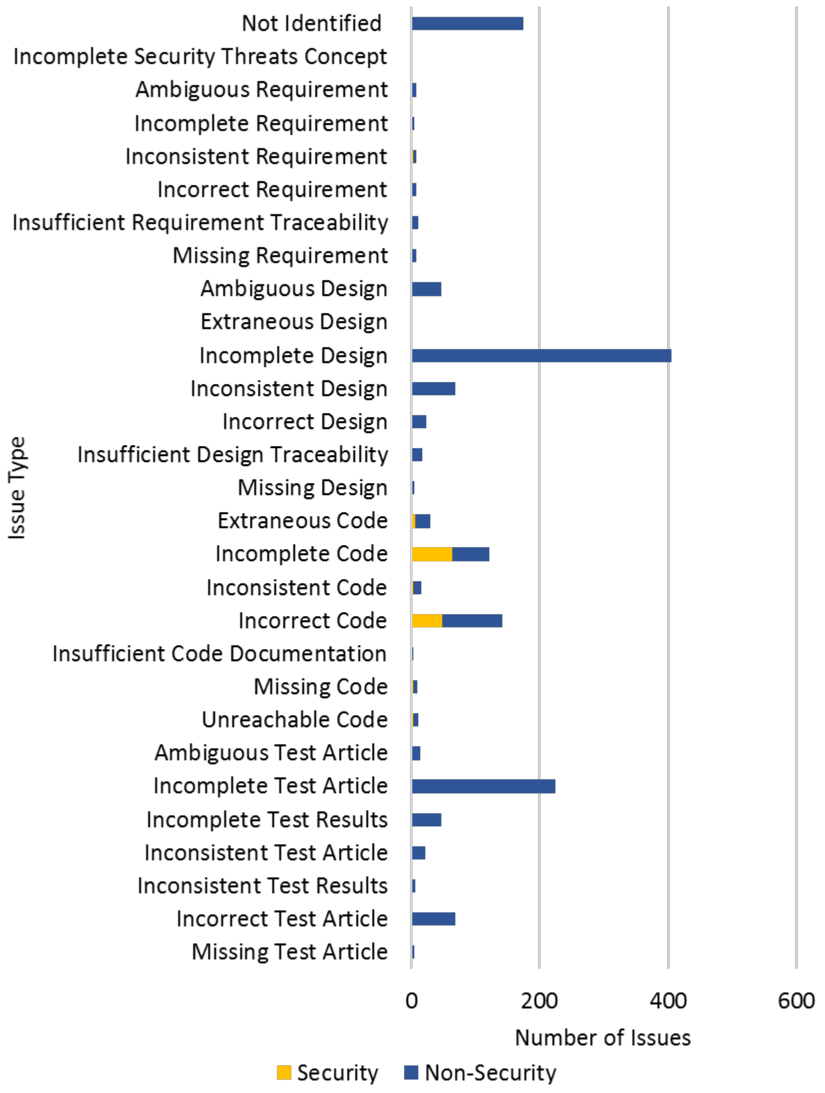

Figure 2. Ground Mission IV&V Issues - Distribution across Issue Types

Figure 3 shows the breakdown of security and non-security issues across Capabilities, ordered from the Capability that contained the highest total number of issues to the Capability with the least total number of issues. Four out of eight Capabilities (i.e., Capabilities 1, 2, 3, and 4) held all of the security issues. These four Capabilities were responsible for 82% of all issues and had similar ratios of security to non-security issues. Capability 3 had the most security issues, specifically 40% (i.e., 53 out of the 133), followed by Capability 2 with 26% (i.e., 34 of 133) of all security issues.

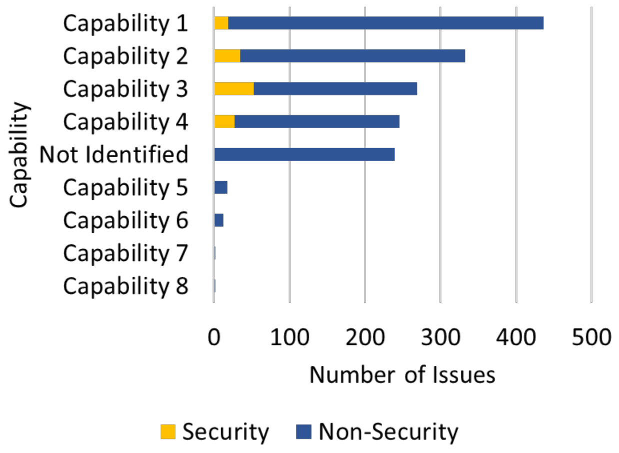

Figure 3. Ground Mission IV&V Issues - Distribution across Capabilities

Figure 4 shows the distribution of security and non-security issues across Subsystems, ordered from the subsystem with the highest total number of issues to the subsystem with the least issues. Subsystem 1 and 2 contributed 86% of all security issues and 70% of all issues, which shows that Pareto principle[1] applies to both the security and total number of issues. This result is consistent with related works focused on fault characterization in general [10], [11], [13], and [30].

Figure 5 shows the distribution of security and non-security issues with respect to the analysis method used to detect the issues. The largest proportion of total issues (30% of all issues) was found using Design Analysis; however this method did not uncover any security issues. The vast majority of security issues were discovered using Implementation Analysis (Static Code Analysis). Specifically, this method was used to discover 91% of all security related issues. Interestingly, the analysis method of Security Analysis (Verify Security Control Implementation) turned up almost no issues.

It should be emphasized here that the amount of time and effort invested in using each Analysis Method affect the number of issues (including security related issues) detected by that method. Unfortunately, the time and effort used for each Analysis Method were not tracked, and therefore we cannot draw conclusions about the effectiveness of the Analysis Methods based on the results presented in Figure 5.

[1]Pareto principle indicates a skewed distribution of software faults, that is, that majority of faults (e.g., raphly 80%) are located in small percent ((e.g., 20%) of software units (e.g., Subsystems or files.

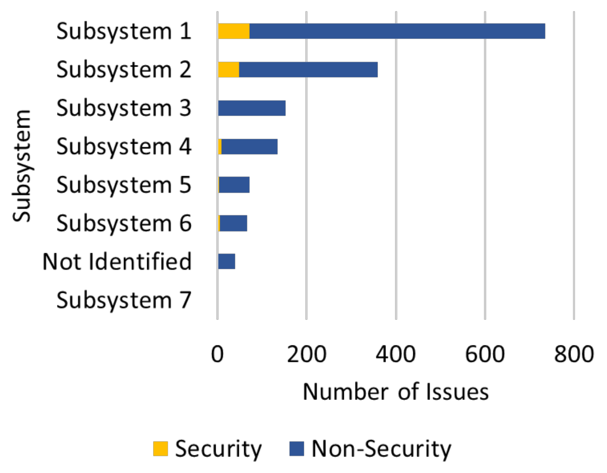

Figure 4. Ground Mission IV&V Issues - Distribution across Subsystems

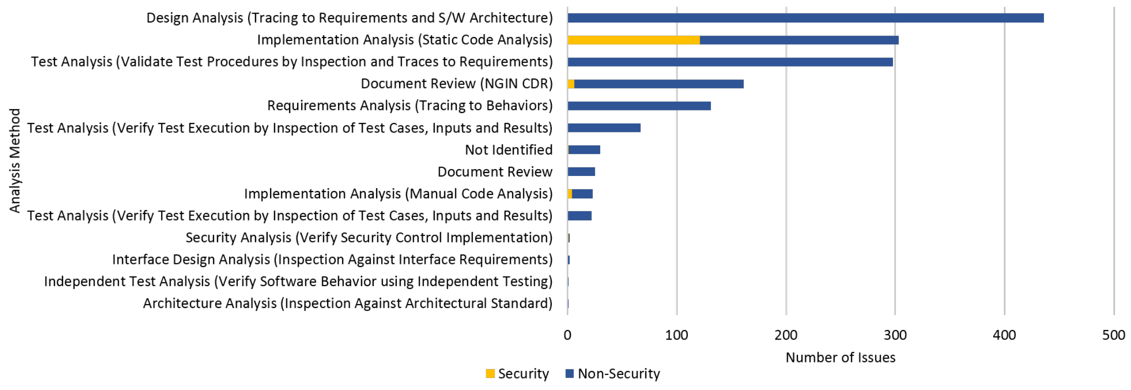

Figure 5. Ground Mission IV&V Issues - Distribution across Analysis Methods

The distribution of security and non-security issues across different Severity levels is shown in Figure 6. NASA's Severity levels range from 1 to 5, with 1 being the most severe. As shown in Figure 6, the majority of all issues (72%) were of severity 3. Similar trend was observed for security related issues as well; specifically, 86% of all security related issues had severity level 3.

Figures 7 and 8 detail the phase in which each issue was introduced and found, respectively. The majority of security issues (91%) were introduced in the Implementation Phase, which again shows how hard implementing secure code is compared to determining the requirements and design from a security standpoint. This result also shows that efforts to enforce secure coding methods (e.g., secure coding standards, check lists, etc) would lead to cost effective improvements of NASA missions security. Comparing Figures 7 and 8 can be observed that the phase in which issues were found closely followed the phase in which they were introduced, which illustrates the effectiveness of the IV&V activities.

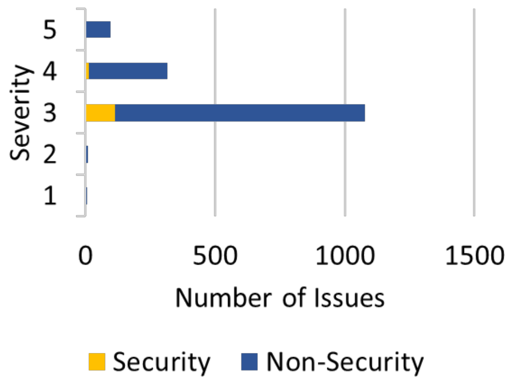

Figure 6. Ground Mission IV&V Issues - Distribution across Severity level

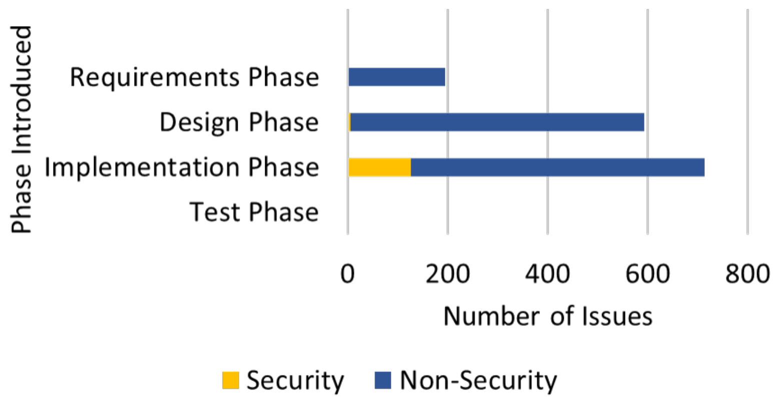

Figure 7. Ground Mission IV&V Issues - Distribution across Phase Introduced

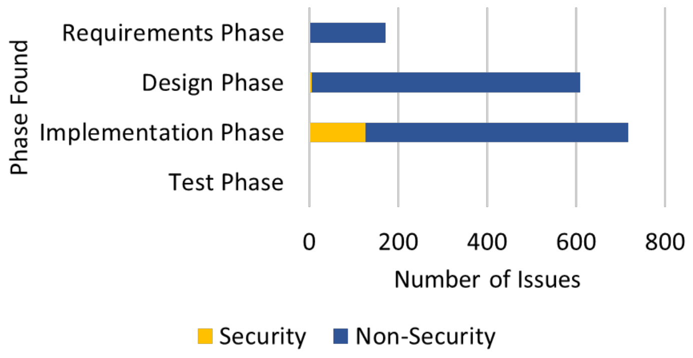

Figure 8. Ground Mission IV&V Issues - Distribution across Phase Found

Next, we focus on the distribution of the security related issues across different CWE-888 classes, with a goal to identify the dominating type of vulnerabilities. As can be seen in Figure 9 the security issues of the NASA ground mission belonged to only 11 out of the total 21 CWE-888 primary classes. The Memory Access dominated the security related issues, containing 53% of all security issues. Furthermore, only five Primary classes (i.e., Memory Access, Unused entities, Exception Management, Risky Values, and Resource Management) contained around 92% of all security issues. Interestingly, this result shows that the Parato principle applies to the distribution of the security issues across Primary classes as well.

The secondary CWE-888 classes provide more specifics about security issues than the primary classes. Figure 10 shows the breakdown of security issues across both the primary and secondary classes, which better represents the types of security issues. This figure shows that the most of the Memory Access primary class issues (i.e., 63 out of 70) belonged to the secondary class Faulty Pointer Use. The remaining dominating primary classes Unused Entities, Exception Management, and Risky Values were comprised mainly of the secondary classes Dead Code, Ambiguous Exception Type, and Glitch in Computation, respectively.

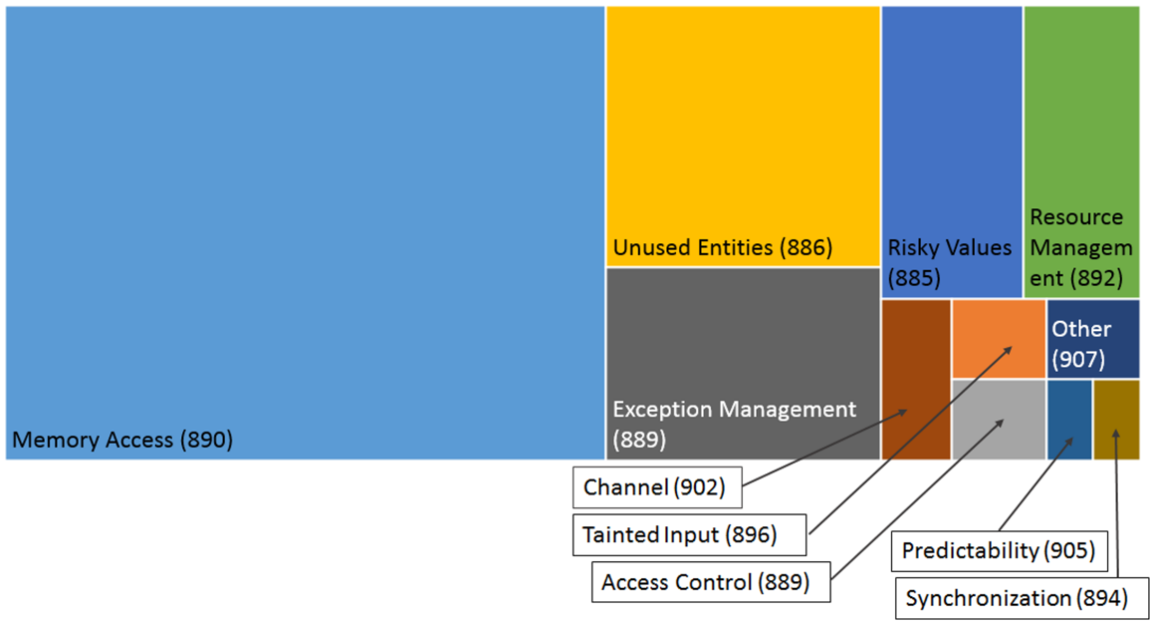

Figure 9. Ground Mission IV&V Issues - Distribution across CWE-888 Primary Classes. The numbers in brackets represent the specific CWE numbers of the Primary Classes.

4.5 Flight Mission IV&V Issues

The Flight Mission had a total of 506 issues in the IV&V issue tracking system. After the removal of issues tagged as “Withdrawn” and “Not an Issue”, 383 issues remained, which were than manually labeled and further analyzed. Note that, unlike in the case of the Ground Mission, the IV&V issues of the Flight Mission were not tagged as security related by the IV&V analysis. We manually labeled these 383 issues using the process described in section 4.3. The results showed that a total of 157 issues were security related (i.e., 41% of all issues). This section describes the results of the analysis.

As shown in Figure 11, most of the security related issues were associated with the Code category, which contributed 92% of all security related issues. This distribution of security related issues aligns with the results for the Ground mission IV&V issues, with majority of security issues related to the code implementation, rather than early life cycle phases.

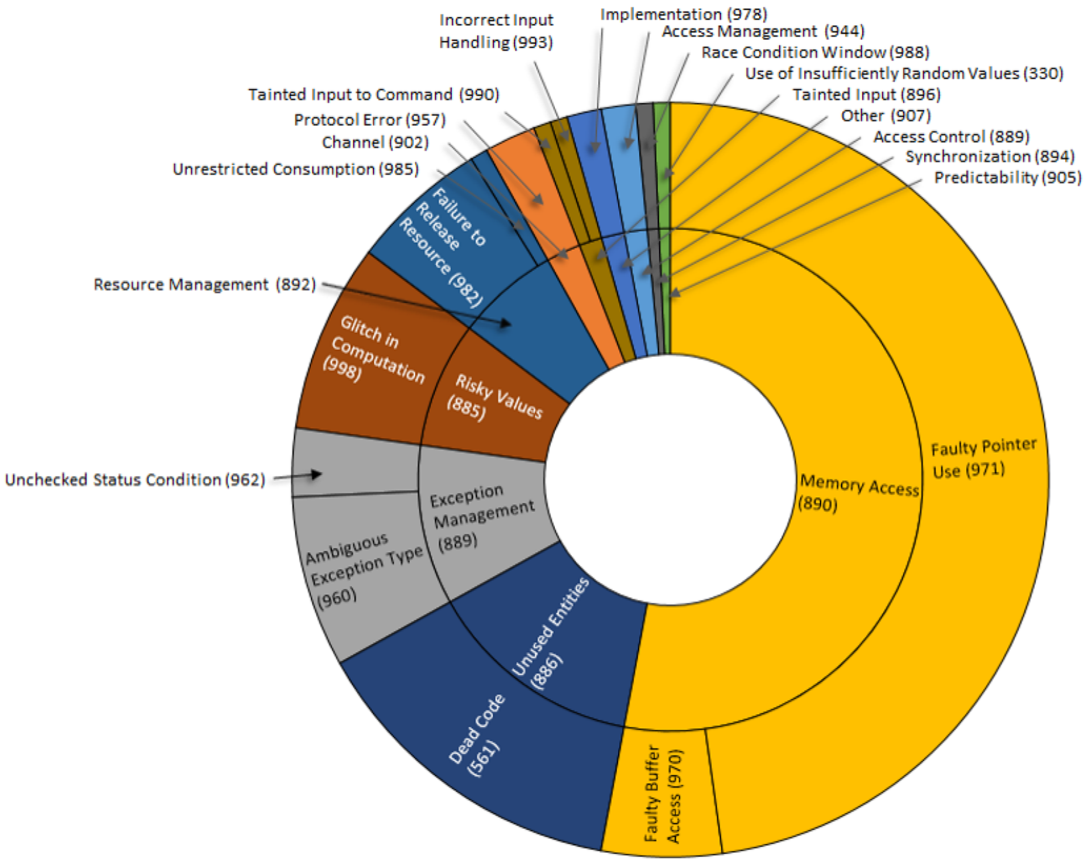

Figure 10. Ground Mission IV&V Issues - Distribution of security issues across CWE-888 Primary and Secondary Classes. The numbers in brackets represent the specific CWE numbers of the Primary and Secondary Classes.

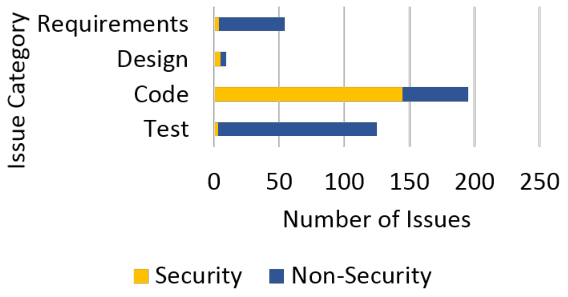

Figure 11. Flight Mission IV&V Issues - Distribution across Issue Categories

Figure 12 shows that security issues are predominately associated with four Issue Types: Incorrect Code, Incomplete Code, Missing Code, and Extraneous Code. It is not surprising that these dominating issues types are all code related having in mind that Code issue category had the most of the security issues (see Figure 11).

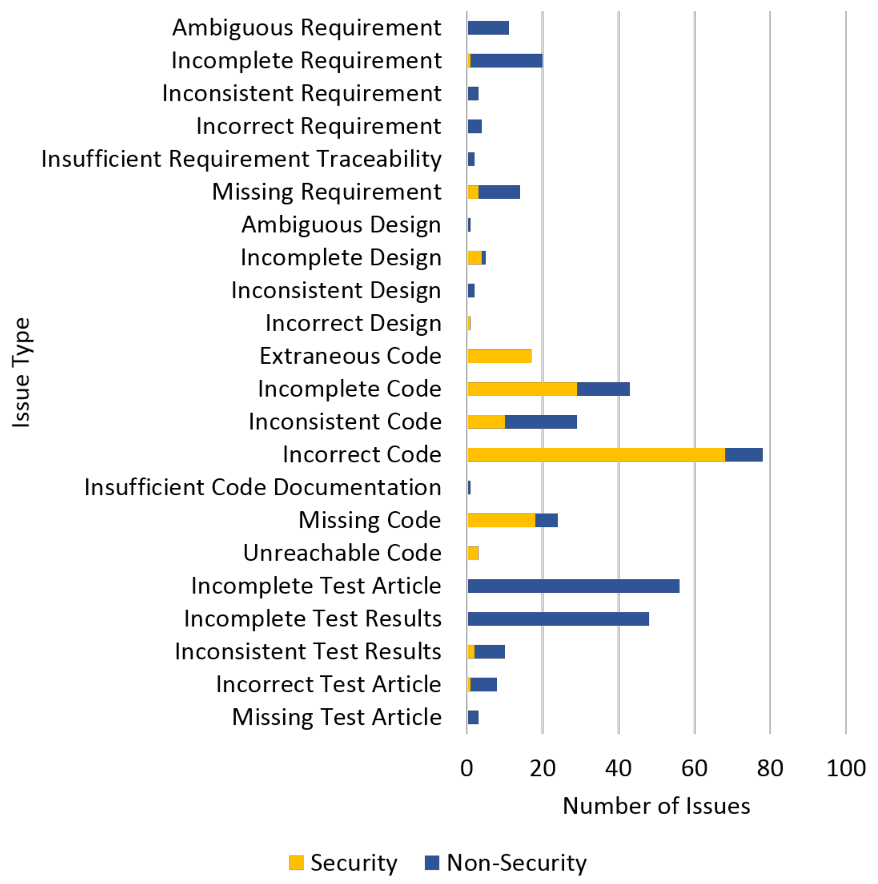

Figure 12. Flight Mission IV&V Issues - Distribution across Issue Types

The Flight Mission IV&V issues had the Defect Categories filed populated. Figure 13 provides a breakdown of security and non-security related issues across Defect Categories. The three dominating Defect Categories were Algorithms and Processing, Control, Logic and Sequence, and Data.

Figure 14 depicts the distribution of security and non-security issues across Flight Mission Capabilities, ordered in decreasing order by the total number of issues. The ratio of security vs. non-security issues was similar across all capabilities, with approximately 30% security and 70% non-security issues.

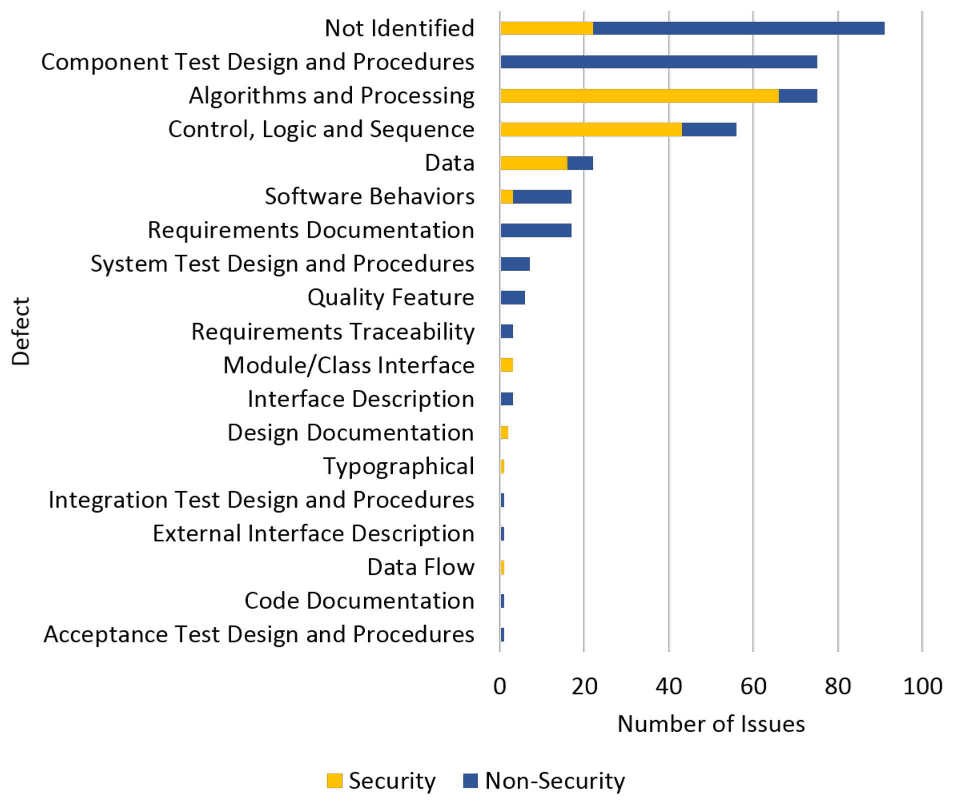

Figure 13. Flight Mission IV&V Issues - Distribution across Defect categories

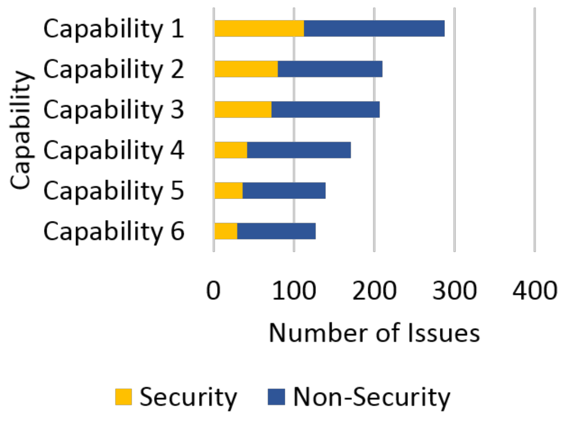

Figure 14. Flight Mission IV&V Issues - Distribution across Capabilities

The distribution of issues across Flight Mission subsystems presented in Figure 15 shows that 88% of all security issues and 88% of all issues fell into three out of five subsystems. Subsystem 3 had slightly more security issues than Subsystems 1 and 2. (Note that Subsystems in Figure 15 are ordered by the total number of issues.)

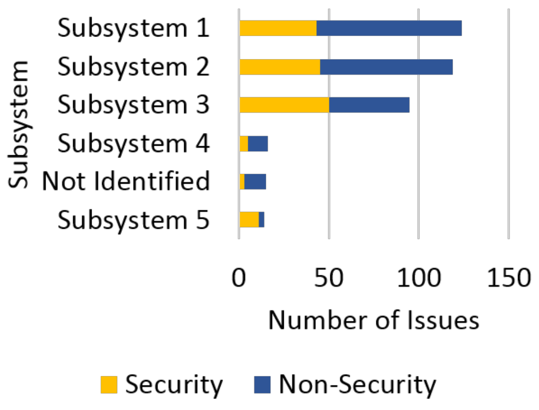

Figure 15. Flight Mission IV&V Issues - Distribution across Subsystems

As shown in Figure 16, Severity levels 3 and 4 together contained 79% of all security issues and 86% of all issues. The fact that not many security issues had high Severity levels (i.e., 1 and 2) is consistent with the Ground mission IV&V security related issues.

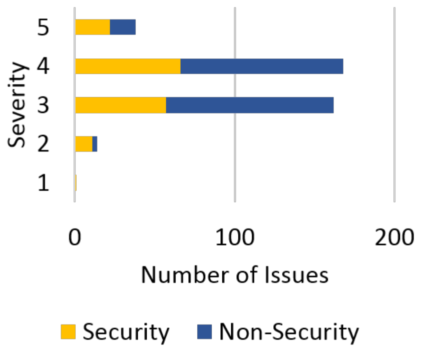

Figure 16. Flight Mission IV&V Issues - Distribution across Severity levels

Figures 17 and 18 also show results consistent with the Ground Mission IV&V issues, with the majority of security issues introduced (85%) and found (85%) in the Implementation phase. Again, the phase in which an issue was found closely followed the phase in which the issue was introduced. The Flight Mission IV&V Issues dataset, in addition included information on the phase in which the issues were resolved. As can be seem in Figure 19, 75% of security related issues were resolved in the Implementation phase, and the remaining 25% were resolved in the Testing phase. Interestingly enough, no security issues were resolved in the Design phase, even though some were introduced and found in that phase.

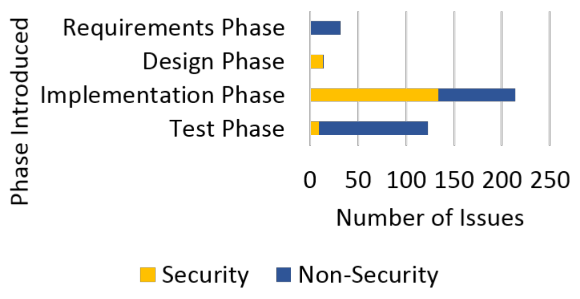

Figure 17. Flight Mission IV&V Issues - Distribution across Phase Introduced

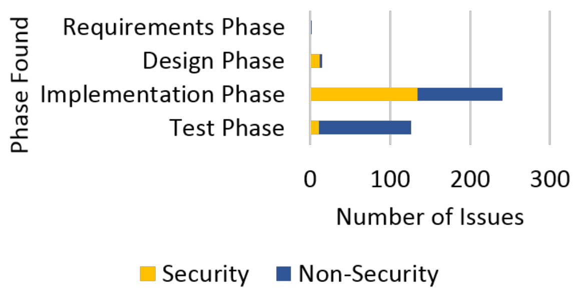

Figure 18. Flight Mission IV&V Issues - Distribution across Phase Found

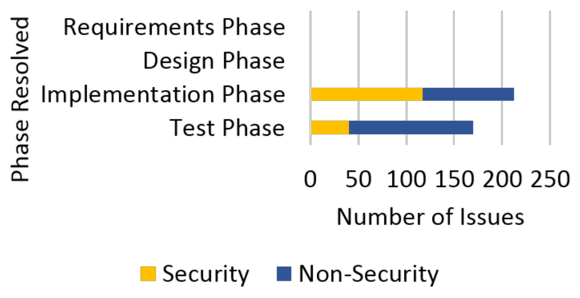

Figure 19. Flight Mission IV&V Issues - Distribution across Phase Resolved

Next, we focus on the distribution of security issues across the CSE-888 Primary classes, which is presented in Figure 20. Similarly as in the case of the Ground Mission IV&V Issues dataset, IV&V issues of the Flight mission belonged to only 9 of the 21 Primary classes, with four overwhelmingly dominating classes: Other, Risky Values, Memory Access, and Unused Entities. Figure 21 shows the distribution of security issues across both the Primary and Secondary classes. Note that the Secondary classes are more specific than the Primary classes, and therefore provide more details on the types of security issues observed in the mission. Thus, of the 59 Other issues, 55 belonged to the Secondary class Implementation and only 4 belonged to the Secondary class Architecture. The Primary class Risky Values contained 30 issues, all of which were associated with the secondary class Glitch in Computation. Of the 20 Memory Access issues, 14 were of the Secondary class Faulty Buffer Access and remaining 6 belonged to the Secondary class Faulty Pointer Use.

The primary class Unused Entities contained 23 issues, 18 of which were of the Secondary class Dead Code and 5 of the Unused Variable.

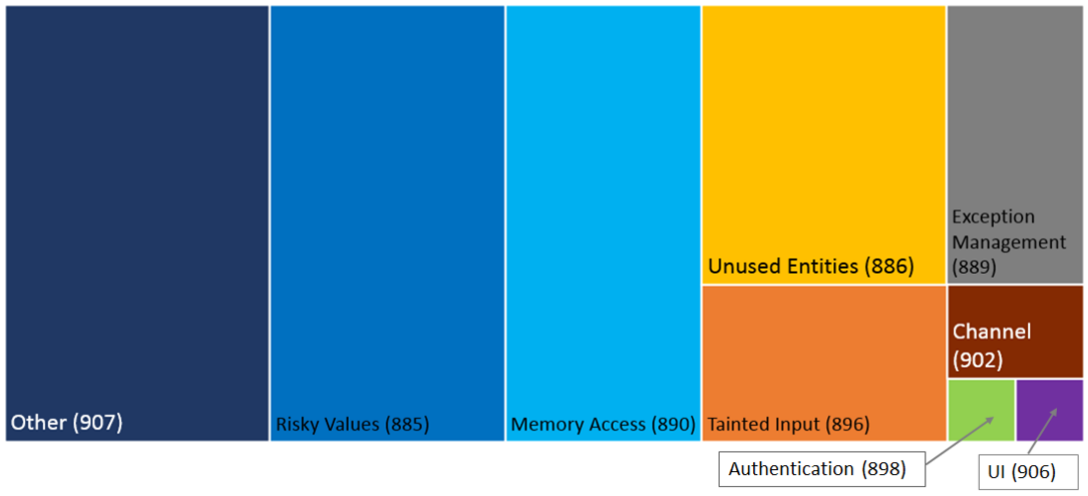

Figure 20. Flight Mission IV&V Issues - Distribution of issues across CWE-888 Primary Classes. The numbers in brackets represent the specific CWE numbers of the Primary Classes.

4.6 Flight Mission Developers Issues

For the Flight mission used as a case study, the issues recorded by the developers were also made available to us. Of the total 1,947 developers issues, after the removal of issues that were not tagged as Defect and were not Closed, 569 issues remained to be further analyzed. The manual labeling of these issues, following the process described in section 4.3, resulted in 374 issues being marked as security related (i.e., 66% of all issues), which is significantly higher percentage of security related issues than in the other two datasets.

Figure 22 shows the distribution security and non-security issues across Issue Types[2]. The two Issues Types – Incorrect Implementation and Incorrect Operation or Unexpected Behavior – significantly outnumbered the other Issue Types.

[2]Note that the values of the Issue Types used in the Flight Mission Developers Change Requests system are different than the Issues Types used for both the Ground and Flight missions IV&V issues.

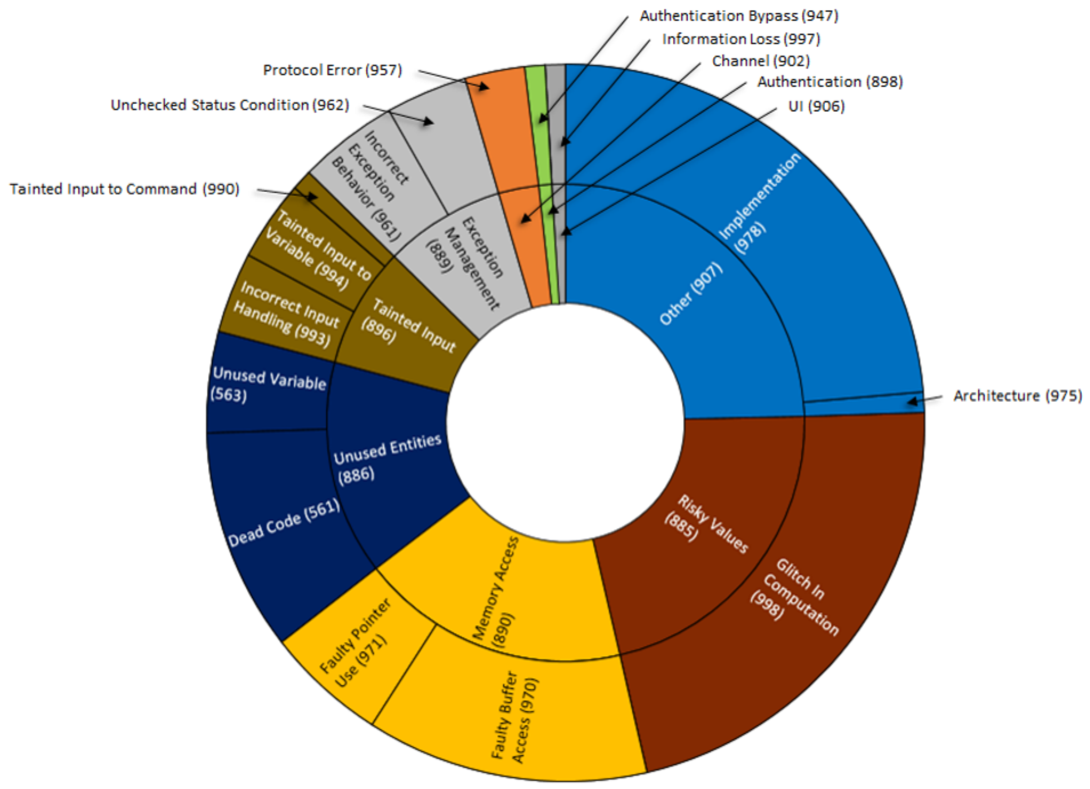

Figure 21. Flight Mission IV&V Issues - Distribution across the Primary and Secondary CWE-888 Classes. The numbers in brackets represent the specific CWE numbers of the Primary and Secondary Classes.

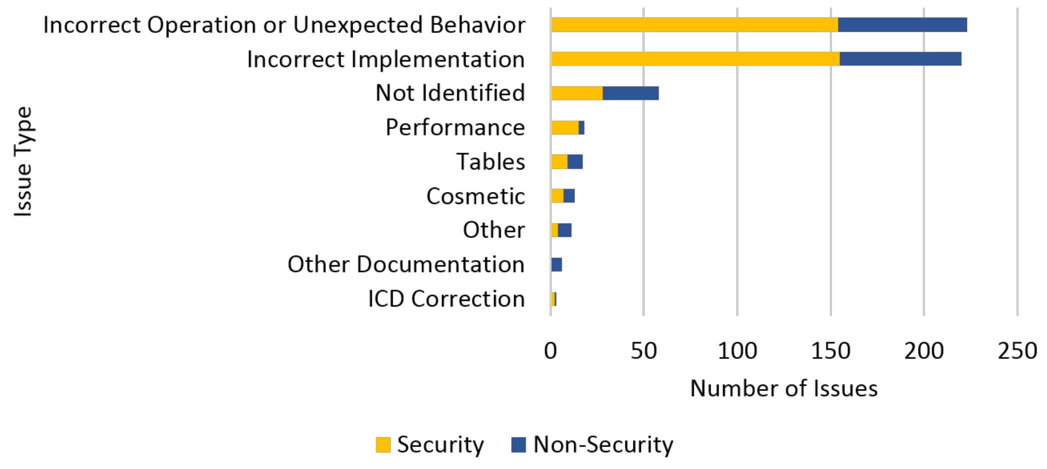

Figure 22. Flight Mission Developer Issues - Distribution across Issue Types

Figure 23 presents the distribution of security and non-security issues across the Flight Mission subsystems. The findings are very similar to the previous datasets, again proving the Pareto principle, with 88% of all security issues found in only four subsystems (out of thirteen), which together accounted for 89% of all issues.

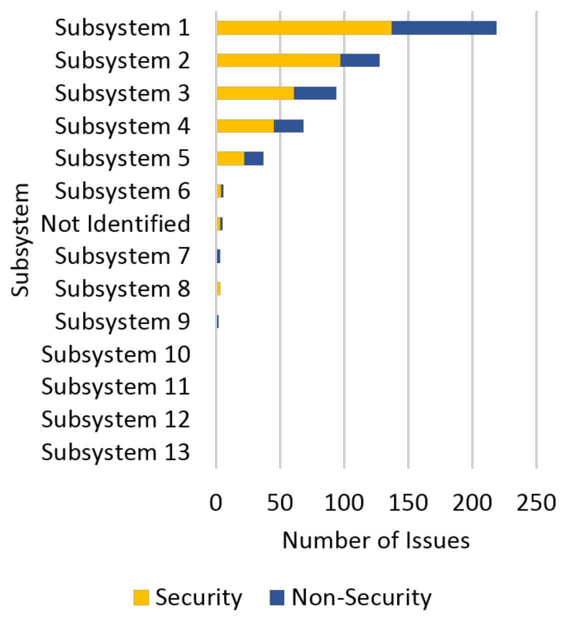

Figure 23. Flight Mission Developer Issues - Distribution across Subsystems

While the severity levels used by the IV&V analysts ranged from 1 to 5, the levels found in this dataset were: Minor, Moderate, and Critical. As shown in Figure 24, the results related to the severity of the Flight mission Developers issues were consistent to the previously analyzed datasets – the moderate severity levels dominated, containing 86% of the security issues, and 85% of the total number of issues. Only 4% of all issues, and 4% of security issues were determined to be critical.

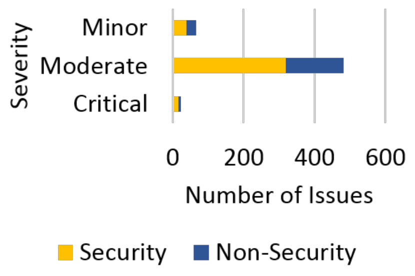

Figure 24. Flight Mission Developer Issues - Distribution across Severity levels

This dataset contained information about the phase in which the issues were found, but no information on when they were introduced or resolved. Also note that the values of the Phases Found are more granular than the ones in case of the IV&V issues of both the Flight and the Ground mission. As shown in Figure 25, most issues were found were during the following there phases listed in decreasing order: Build Verification, Build Integration, and Code Implementation

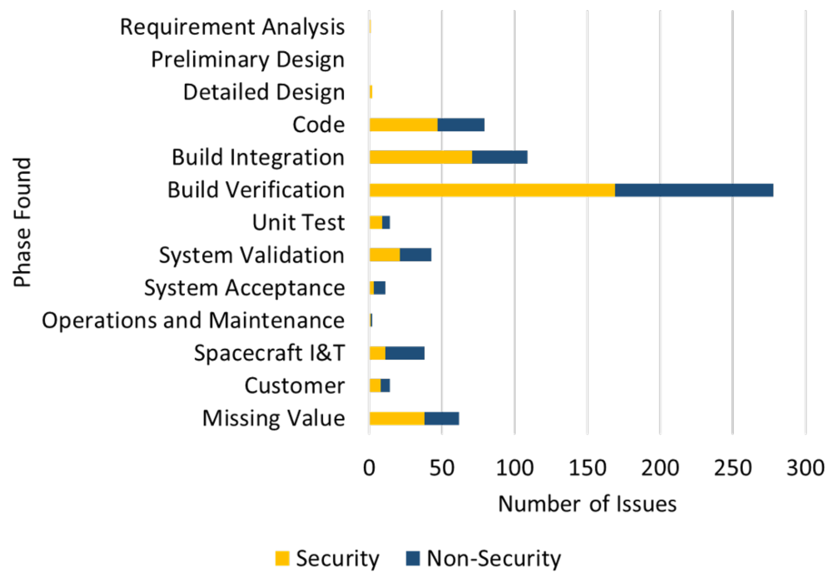

Figure 25. Flight Mission Developer Issues - Distribution across Phase Found

Next we focus on the classification of the security related issues using the CWE-888 view. Similarly as for the other two datasets, as shown in Figure 26, only 13 of 21 Primary class were observed, with three dominating classes (i.e., Risky Values, Exception Management, and Memory Access). Figure 27 shows the distribution of security issues across both the Primary and Secondary classes. The previous two datasets mostly exhibited a single dominating Secondary class within each Primary class, which is not always the case with this dataset, as it can be observed in Figure 27. The Primary class with the most issues was Risky Values, with the only Secondary class Glitch in Computation. However, the next largest Primary class Exception Management, had two approximately equal Secondary classes: Incorrect Exception Behavior and Unchecked Status Condition, with 48 and 42 security issues, respectively. Similarly, the Primary class Memory Access consisted of 22 security issues in the Secondary class Faulty Buffer Access and the remaining 12 security issues in the Secondary class Faulty Pointer Use.

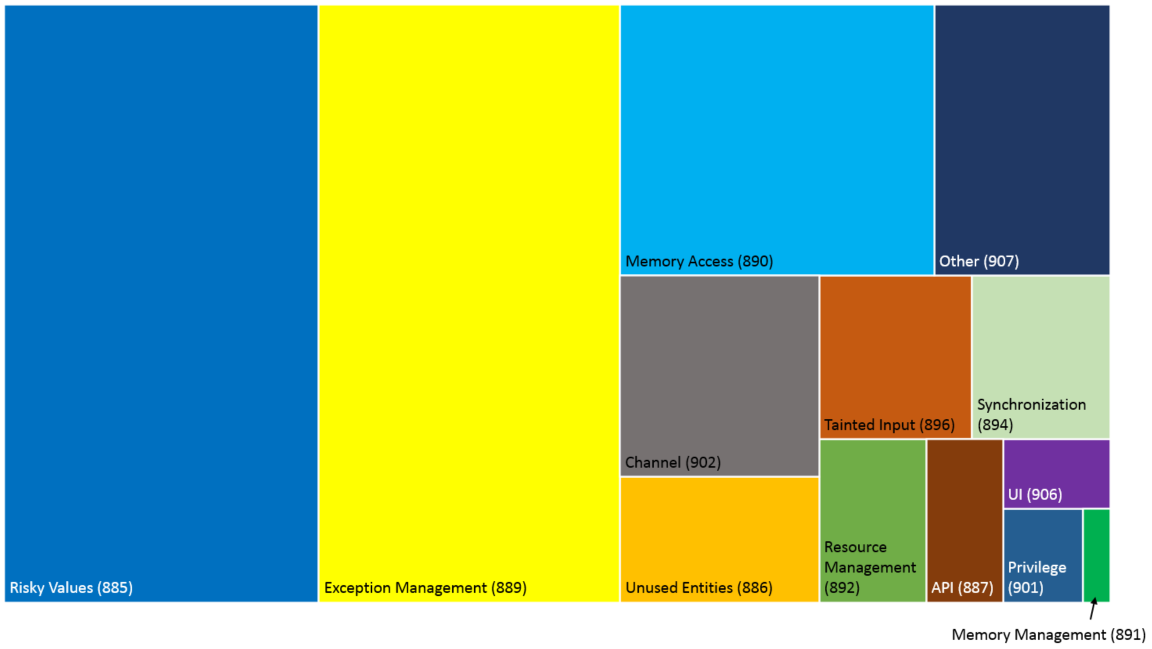

Figure 26. Flight Mission Developer Issues - Distribution across CWE-888 Primary classes

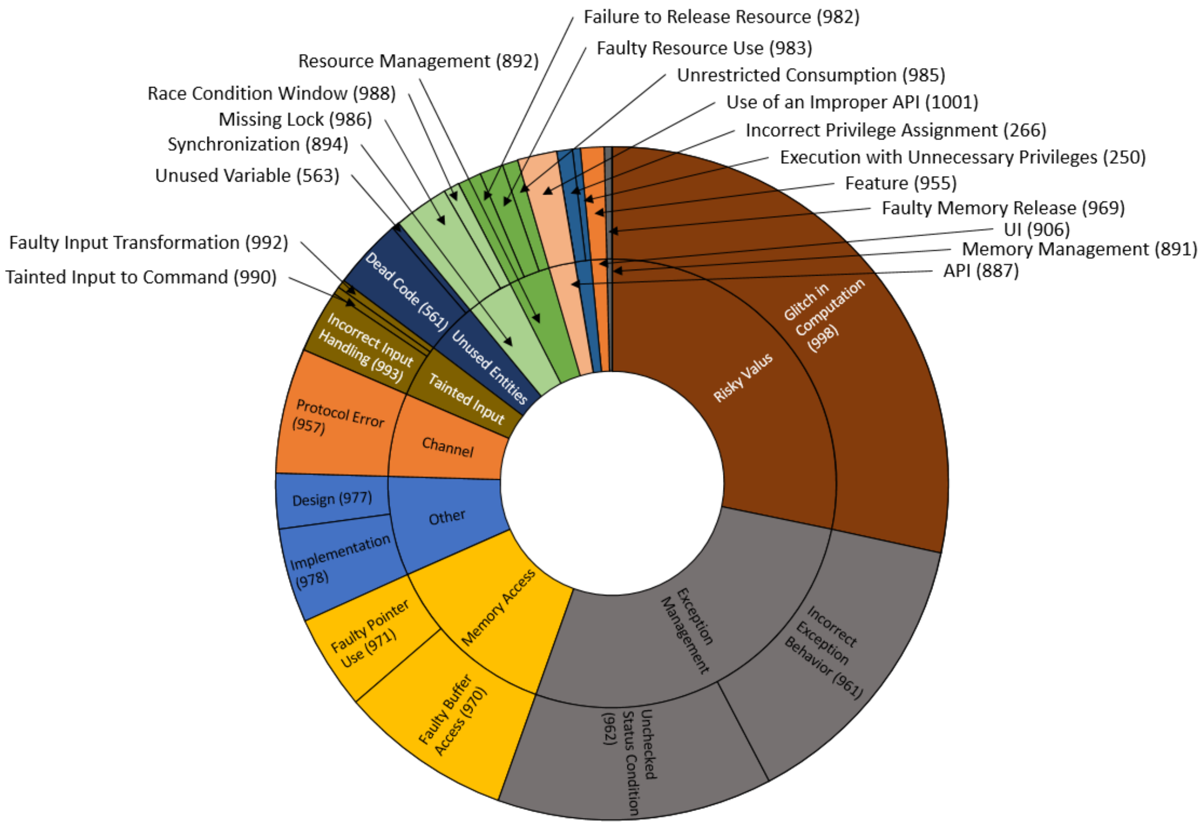

Figure 27. Flight Mission Developer Issues - Distribution across CWE-888 Primary and Secondary classes

5 Comparison of the Results

This section presents a comparison of the results across all datasets. We start with comparing the distribution of security issues across the CWE-888 Primary classes, extracted from the results presented in subsections 4.4, 4.5, and 4.6, which are summarized in Table 1. Figure 28 depict the same information in a graphical form. As can be seen from Table 1 and Figure 28, even though there are fifteen (out of 21) Primary classes that had nonzero security issues for at least one dataset, the vast majority of security issues were distributed among five dominant Primary classes: Exception Management, Memory Access, Other, Risky Values, and Unused Entities. Specifically, these five Primary classes together contained 90%, 87%, and 72% of the security issues in the Ground mission IV&V, Flight mission IV&V and Flight mission Developers issues, respectively. Interestingly, Primary classes which had zero security issues in one or two datasets made up for only very small proportion (from 0.4% to at most 7%) of the security issues in the dataset they appeared in.

Table 1. Comparison of Primary classes across the three datasets. Only the Primary classes with nonzero security issues at least in one of the datasets are shown. The corresponding NASA's Top 25 ground and flight mission CWEs are also shown.

Primary CWE-888 Class	**Ground Mission**		**Flight Mission**		
	IV&V Issues	**Top 25 CWEs**	**IV&V Issues**	**Developer Issues**	**Top 25 CWEs**
API (887)		242,676		1.9%	676
Authentication (898)		798	0.9%		
Channel (902)		353,290,421	2.7%	6.0%	290
Exception Management (889)	10.8%		8.2%	27.2%	391,665,754
Memory Access (890)	54.6%	119, 787, 122, 121, 788, 805, 131, 496, 170	18.3%	12.8%	119, 121, 122, 787, 120, 788, 805, 476, 170, 131, 129
Memory Management (891)				0.4%	415,762
Other (907)	1.5%	464	24.5%	7.1%	464
Predictability (905)	0.8%				326
Privilege (901)				1.2%	
Resource Management (892)	6.9%	399,770		3.0%	399,401,404,416,770
Risky Values (885)	8.5%	835, 468, 681, 682, 192, 457	21.8%	28.3%	457, 835, 467, 681, 682, 192, 190, 193, 197, 195, 562
Synchronization (894)	0.8%	667, 964, 833, 367		3.4%	667, 764, 833
Tainted Input (896)	1.5%	77, 88, 134, 20, 79, 78, 692	8.2%	3.8%	20, 134, 176
UI (906)			0.9%	1.1%	
Unused Entities (886)	14.6%		14.5	3.8%	

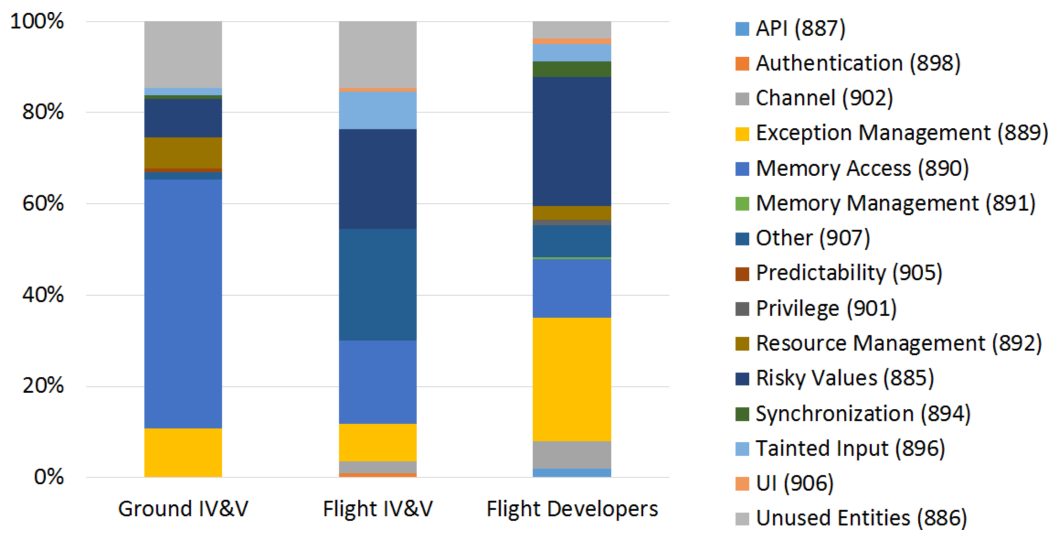

Figure 28. Distribution of security issues across CWE-888 Primary classes, for all three datasets

Table 2 shows the dominating CWE-888 Primary classes along with their corresponding Secondary classes, which provide more detailed information on the nature of security issues in the corresponding dominating Primary classes. The Secondary classes under the Exception Management Primary class included: Ambiguous Exception Type, Incorrect Exception Behavior, and Unchecked Status Condition. Memory Access was another dominating Primary class, consisting of Faulty Buffer Access and Faulty Pointer Use. These categories include common programming errors such as null pointer dereferences and buffer overflows. As shown by Younan [22], buffer overflows continue to be one of the most common vulnerabilities in software systems.

The next dominating Primary class was Other, with the Secondary classes Design and Implementation. Design class consists of weaknesses dealing with insufficient control flow management or reliance on data/memory layout. Implementation class is based around weaknesses such as coding standards violation or containment errors. These Secondary classes were assigned to issues which had security problems related strictly to their design or implementation and could not be placed into any other class.

Another dominating Primary class was Risky Values. This Primary class consists of the Secondary class Glitch in Computation, which deals with calculation errors. These involve everything from a divide by zero error to a function call with an incorrect order of arguments. The Flight mission Developers issues dataset had the highest percentage of these errors, which was closely followed by the Flight mission IV&V issues.

The last dominating Primary class was Unused Entities, consisting of the Dead Code and Unused Variable Secondary classes. These issues were were found much more often in the IV&V datasets, both for the Ground and the Flight mission.

Table 2. Comparison of the Secondary Classes, only for the five dominant Primary classes. The corresponding NASA's Top 25 ground and flight mission CWEs are also shown.

Primary Cluster	**Ground Mission**		**Flight Mission**		
	IV&V Issues	**Top 25 CWEs**	**IV&V Issues**	**Developer Issues**	**Top 25 CWEs**
Exception Management (889)					
Ambiguous Exception Type (960)	7.7%				
Incorrect Exception Behavior (961)			4.5%	14.0%	
Unchecked Status Condition (962)	3.1%		3.6%	13.2%	391,665,754
Memory Access (890)					
Faulty Buffer Access (970)	4.6%	119, 787, 122, 121, 788, 805	12.7%	8.3%	119, 121, 122, 787, 120, 788, 805, 129
Faulty Pointer Use (971)	50.0%	476	5.5%	4.5%	476
Other (907)					
Architecture (975)			0.9%		
Design (977)		464		2.6%	464
Implementation (978)	1.5%		23.6%	4.5%	
Risky Values (885)					
Glitch in Computation (998)	8.5%	835, 468, 681, 682, 192, 457	21.8%	28.3%	457, 835, 468, 681, 682, 192, 190, 193, 197, 195, 562
Unused Entities (886)					
Dead Code (561)	14.6%		10.0%	3.4%	
Unused Variable (563)			4.5%	0.4%	

Tables 1 and 2 also contain columns with the list of the NASA's top 25 CWEs for the ground and flight missions. These top 25 CWEs were determined by the IV&V Program and are maintained at the NASA's Secure Coding Portal. The top 25 CWEs listed in the third and sixth columns in Tables 1 and 2 are actually children of the corresponding Primary CWE class shown in the same row.

Note that the dominant Primary and Secondary classes should not be directly compared with the top 25 CWEs because they do not reflect the same information. Thus, the dominant Primary and Secondary classes in this report were identified based on the actual data extracted from the IV&V and Developers issue tracking systems. On the other side, the top 25 CWEs were identified using the Common Weakness Scoring System (CWSS) [31], which provides a mechanism for prioritization of software weaknesses. CWSS is organized into three metric categories: Base Finding, Attack Surface, and Environmental. Each category contains multiple metrics, which are known as factors. After each factor in the category is assigned a value (typically a guesstimate based on experience or anecdotal knowledge), these values are converted to weights and a category sub-score is calculated. The three sub-scores are multiplied together, which produces the CWSS score. Higher the CWSS score, higher the priority of the particular software weakness. Note that analyzing the Attack Surface and Environmental factors was out of the scope of our work.

It is important to note that the results related to the dominating vulnerability classes identified in this report provide useful, evidence-based input to the current process used to create the top 25 Ground mission and Flight mission vulnerabilities.

6 Threats to Validity

Any empirical study have threats to validity. In this section we briefly discuss the threats to validity to our study.

It should be noted that the number and classes of identified security issues depend on the quality of software artifacts, including the level of provided details related to security (if any). It appears that each dataset contained a small amount (15% or less) of issues that did not contain sufficient information for classification. These issues, which lacked security related information and did not fit in any CWE description, were classified as not-security related. Furthermore, the types of security issues (and consequently the identified Primary and Secondary classes) may depend on the validation and verification (V&V) methods used, as well as the amount of time and effort expended on using these methods.

Another threat to validity was due to the fact that some issues could be correctly classified into multiple CWE classes. This was partially mitigated by the hierarchical structure of the CWE-888 view. Furthermore, the number of issues fitting into multiple CWE classes was small, and for these cases the most relevant of the possible classes was selected. Therefore, the effect of this treat to validity on creation of vulnerability profiles was likely insignificant.

In the case of the Ground mission IV&V dataset, as described in Section 4.4, there was a significant number of security related issues that were tagged as testing related. Since no CWE exists that covers such a case, these testing related security issues were not considered in the further analysis.

7 Conclusion

This report focuses on exploring the security related issues, that is, vulnerabilities present in NASA missions based on the information provided in issue tracking systems. While some prior work exists on characteristics of software faults (i.e., bugs) and failures, very little work has been published on analysis of software vulnerabilities, in NASA projects and in general. The most interesting empirical findings from our study are summarized in Table 3. The percentage of security issues was the lowest in the Ground mission IV&V dataset (9% if testing issues are excluded, 20% if they are included) followed by the Flight mission IV&V dataset (41%) and it was the highest in the Flight mission Developers dataset. One explanation for the lowest proportion of security related issues in the Ground mission IV&V dataset may be the fact that this mission is still under development and that the testing phase has not yet begun. Unfortunately, the Ground mission Developers issues were not available to the research team, so we cannot confirm the trend observed in the Flight mission, which exhibited higher proportion of security issues in the Developers dataset than in the IV&V dataset.

Interestingly, the Code related security issues dominated both the Ground mission IV&V security issues, as well as the Flight mission IV&V security issues, with 95% and 92%, respectively. This is an important finding because it indicates that even when requirements and design are done adequately, significant number of vulnerabilities are introduced during the implementation. Therefore, enforcing secure coding practices and verification and validation focused on coding errors (for example by using check list for inspection) would be a cost effective way to improve mission's security. (Note that the Flight mission Developers issues dataset did not contain the data in Issue Category.)

In both the Ground mission IV&V issues dataset and the Flight mission IV&V issues dataset for which the information was available, majority of issues (i.e., 91% and 80%, respectively) were introduced in the Implementation phase. This is consistent with the fact that majority of security issues were code related. It is important that software vulnerabilities (i.e., security related issues) are fixed in a timely manner. The good news is that in most cases the phase in which the issues were found was the same as the phase in which they were introduced. The most security related issues of the Flight mission Developers issues dataset were found during Code Implementation, Build Integration, and Build Verification, which is consistent with the other datasets. However, the data on the Phase these issues were introduced were not available for the Flight mission Developers issues dataset.

The location of security issues, as the location of issues in general, followed the Pareto principle. Specifically, from 88% to 90% the security issues were located in three to four subsystems for all three datasets.

With respect to severity, the results showed that the security issues, as the majority of all issues, were with moderate severity across all three datasets. It should be noted, however, that in the case of the Flight mission datasets (both the IV&V issue and Developers issues) the security aspects of software bugs were not explicitly addressed in the issue tracking system and it is possible that the potential security implications were not accounted for.

The final row in Table 3 lists the five dominating Primary CWE-888 classes (out of 21 classes), which together contributed from around 80% to 90% of all security issues in each dataset. This again proves the Pareto principle of uneven distribution of security issues across CWE classes and supports the fact that addressing these dominant security classes provides the most cost efficient way to improve the mission security.

Table 3. Main findings across all datasets

	Ground mission IV&V issues	Flight mission IV&V issues	Flight mission Developers issues
% Security Issues	9% (20% if Testing security issues are considered)	41%	66%
Security Issues Category	95% Code (39% Code and 53% Testing related if testing security issues are considered.)	92% Code	Data not available
Phase Introduced	91% in the Implementation Phase	85% in the Implementation Phase	Data not available
Phase Found	Followed closely the phase introduced distribution	Followed closely the phase introduced distribution	Most found during Code Implementation, Build Integration, and Build Verification
Subsystem	86% found in two subsystems (70% of all issues)	88% in three subsystems (88% of all issues)	88% in four subsystems (90% of all issues)
Severity of Security Issues	Level 3 dominated (86%)	Levels 3 and 4 dominated (together 78%). 7% were level 2	Moderate dominated (84%)
Five (out of 21) most frequent Primary Classes	Exception Management 10.8% Memory Access 54.6% Other 1.5% Risky Values 8.5% Unused Entities 14.6% Total 90%	Exception Management 8.2% Memory Access 18.2% Other 24.5% Risky Values 21.8% Unused Entities 14.5% Total 87%	Exception Management 27.2% Memory Access 12.8% Other 7.1% Risky Values 28.3% Unused Entities 3.8% Total 79%

References

1. "Inadequate security practices expose key NASA network to cyber attack," Audit report, May 2011.

2. "NASA cybersecuirty presentation," NASA Office of the Chief Information Officer, Nov 2014.

3. "NASA cybersecurity: An examination of the agency's information security," Testimony before the Subcommittee on Investigations and Oversight, House Committee on Science, Space, and Technology, Feb 2012.

4. L. del Monte, "Towards a cyber-security policy for a sustainable, secure and safe space environment," in *26th Symposium on Space Policy, Regulations and Economics*, ser. IAC-13, 2013.

5. K. Osborn, "Air force faces increasing space threats: Shelton," in *DefenseTech*, Sep 2013.

6. S. Hetherington, S. Gunawardena, and E. Levy, "Secure coding best practices," Technical Report, Sep 2016.

7. "Common Weakness Enumeration," September 2013, https://cwe.mitre.org/ (accessed January 23, 2017).

8. "Common Vulnerabilities and Exposures," January 2017, https://cwe.mitre.org/ (accessed January 23, 2017).

9. "National Vulnerability Database," January 2017, https://nvd.nist.gov/ (accessed January 23, 2017).

10. N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 797–814, Aug 2000.

11. M. Hamill and K. Goseva-Popstojanova, "Common trends in software fault and failure data," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 484–496, July 2009.

12. ——, "Exploring the missing link: an empirical study of software fixes," *Software Testing, Verification and Reliability*, vol. 24, no. 8, pp. 684–705, 2014. [Online]. Available: http://dx.doi.org/10.1002/stvr.1518

13. ——, "Exploring fault types, detection activities, and failure severity in an evolving safety-critical software system," *Software Quality Journal*, vol. 23, no. 2, pp. 229–265, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11219-014-9235-5

14. ——, "Analyzing and predicting effort associated with finding and fixing software faults," *Information and Software Technology*, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2017.01.002

15. M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," in *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, June 2010, pp. 447–456.

16. J. Alonso, M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault repairs and mitigations in space mission system software," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2013, pp. 1–8.

17. A. K. Maji, K. Hao, S. Sultana, and S. Bagchi, “Characterizing failures in mobile oses: A case study with Android and Symbian,” in *21st IEEE International Symposium on Software Reliability Engineering*, Nov 2010, pp. 249–258.

18. F. Frattini, R. Ghosh, M. Cinque, A. Rindos, and K. S. Trivedi, “Analysis of bugs in Apache Virtual Computing Lab,” in *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2013, pp. 1–6.

19. X. Xia, X. Zhou, D. Lo, and X. Zhao, “An empirical study of bugs in software build systems,” in *13th International Conference on Quality Software*, July 2013, pp. 200–203.

20. O. Alhazmi, Y. Malaiya, and I. Ray, “Measuring, analyzing and predicting security vulnerabilities in software systems,” *Computers & Security*, vol. 26, no. 3, pp. 219 – 228, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404806001520

21. H. Venter, J. Eloff, and Y. Li, “Standardising vulnerability categories,” *Computers & Security*, vol. 27, no. 3–4, pp. 71 – 83, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404808000096

22. “25 years of vulnerabilities: 1988-2012,” Research Report.

23. “CWE-2000: Comprehensive CWE dictionary, MITRE Corporation,” https://cwe.mitre.org/data/slices/2000.html.

24. “CWE-1000: Research concepts, MITRE Corporation,” https://cwe.mitre.org/data/graphs/1000.html.

25. N. Mansourov, “Software fault patterns: Towards formal compliance points for cwe,” 2011, [online] https://buildsecurityin.us-cert.gov/sites/default/files/Mansourov-SWFaultPatterns.pdf.

26. “CWE-888: Software fault pattern (SFP) clusters, MITRE Corporation,” https://cwe.mitre.org/data/graphs/888.html.

27. K. Tsipenyuk, B. Chess, and G. McGraw, “Seven pernicious kingdoms: a taxonomy of software security errors,” *IEEE Security Privacy*, vol. 3, no. 6, pp. 81–84, Nov 2005.

28. “CWE-700: Seven pernicious kingdoms, MITRE Corporation,” https://cwe.mitre.org/data/definitions/700.html.

29. “CWE-699: Development concepts, MITRE Corporation,” https://cwe.mitre.org/data/graphs/699.html.

30. V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882

31. “Common Weakness Scoring System,” January 2017, https://cwe.mitre.org/cwss/cwss_v1.0.1.html (accessed January 23, 2017).

Appendix A

Table 4: IV&V issue tracking system field descriptions

Field title	**Description**	**Is field used?**
Issue ID	An ID is assigned to each issue and recorded in this field	Yes
Project	Contains the project name that the issue is from	Yes
State	The current state of the issue (i.e. Closed, Submitted, Withdrawn, etc)	Yes
Subject	The subject or title of the issue	Yes
Attachments	N/A	No
Capability	General Grouping of Functionality - A capability is made up of several subsystems which is made up of several software components	Yes
Comments	Updates about the progression and search for solution of the issue	Yes
Count	N/A	Yes
Data Restrictions	N/A	Yes
Defer Date	N/A	No
Defer Issue	N/A	No
Defer Notify Recipients	N/A	No
Description	The full description of the issue	Yes
Impact	The projected or observed impact of the issue on the system	Yes
Issue Category	The category the issue falls into (i.e. Code, Design, etc.)	Yes
Issue Type	The type of issue, more specific than Issue Category (i.e. Incomplete Design, Incorrect Code, etc.)	Yes
Severity	How sever the issue is	Yes
Method	The analysis method used to detect the issue	Yes
Originator	N/A	Yes
Phase Found	The project phase in which the issue was found	Yes
Phase Introduced	The project phase in which the issue was introduced	Yes
Phase Resolved	The project phase in which the issue was resolved	No
Recommended Actions	The action that the analyst recommends taking in response to the issue	Yes
References	Any material that can be reference to support the issue	Yes
Related Issues	Any issues highly related to the current issue	Yes
Resolution Chronology	A history of the solution of the issue	Yes
Technical Framework Level 1	N/A	Yes
Technical Framework Level 2	N/A	Yes
Technical Framework Level 3	N/A	Yes

Field title	**Description**	**Is field used?**
Workaround	If issue cannot be solved, what was put in place to account for it	Yes
Defect	The defect of the issue (i.e. Software Behaviors, Requirements Documentation, etc)	Yes
Defect Category	The category of the defect (i.e. Design, Requirements, etc.)	Yes
Analysis Method	The method used to review the issue	Yes
Element	The element that the issue originates from, similar to the “Subsystem” in the Developer Issue Tracking System - A subsystem is made up of several software components	Yes
Date Submitted to POC	N/A	Yes
Reqt Number	N/A	No
Developer ITA	N/A	Yes
Verification Procedure Review	The procedure used to verify the fix of the issue	Yes
Created By	The analyst or developer that entered the issue into the issue tracking system	Yes
Created Date	The date the issue was entered into the issue tracking system	Yes
Updated By	The last analyst or developer to update the issue	Yes
Updated Date	The last data the issue was updated	Yes

Table 5: Developers issue tracking system field descriptions

Field title	**Description**	**Is field used?**
Issue ID	An ID is assigned to each issue and recorded in this field	Yes
DCR Product	The product the DCR relates to	Yes
Type	The project and product of the DCR	Yes
DCR Solution	The actions taken to resolve the DCR	Yes
DCR Severity	The criticality of the DCR	Yes
DCR Subtype	A general category of the problem under concern	Yes
DCR/Test Description	The description of the DCR	Yes
DCR Subsystem	The part of the system the DCR originates from, similar to “Element” from the IV&V Issue Tracking System - A subsystem is made up of several software components	Yes
DCR Type	The type of DCR (i.e. Defect, Change Request, etc.)	
DCR Title	The title or subject of the DCR	Yes
DCR Priority	How urgent fixing the DCR is	Yes
DCR Application	The application the DCR originates from	Yes
DCR Closure Notes	Points of interest detailing the solution of the DCR or deviations from normal routine	Yes
State	What lifespan stage the DCR is in	Yes
DCR Date Closed With Defect	N/A	Yes
DCR Date In Test	The date the DCR is ready for testing	Yes
Backward Relationships	Any previos DCR’s that the current is related to	Yes
DCR Test Procs Used to Verify	The procedures used to verify the DCR	Yes
DCR Date On Hold	If the DCR was put on hold, the data of which this took place	Yes
DCR Date Test Completed	The date at which the testing on the DCR was completed	Yes
DCR Date Ready For Test	The date the DCR is ready to be tested	Yes
DCR Affects FSRL	N/A	Yes
Attachments	Any attachments that assist with the description, testing, or resolution of the DCR	Yes
DCR Date Closed	The date the DCR was closed	Yes
Implements	Any other DCR solutions that the DCR under concern implements	Yes
DCR Build Target	N/A	Yes
DCR Test Assigned Tester	The developer assigned to test the DCR	Yes
DCR Test Log Init Files Folder	N/A	Yes
Signature Comment	N/A	No

Field title	**Description**	**Is field used?**
DCR IRB Comments	N/A	Yes
DCR Test Outcome	Initial test results	Yes
DCR Test Tester Comments	Comments left by the testing developer	Yes
DCR Date In Work	Date when work starts on the DCR	Yes
DCR Date Work Completed	Date the work is finished on the DCR	Yes
DCR Phase Found	The development phase of which the DCR was found	Yes
DCR IRB Comments History	N/A	No
DCR Workflow	N/A	Yes
Forward Relationships	The related DCR's created after the one under concern	Yes
DCR Date Assigned	The date the DCR is assigned to a dveloper	Yes
DCR Test Log Init Files	N/A	Yes
DCR Document Type	The type of DCR document (i.e. requirements, algorithms, etc)	Yes
Links to Tests from DCR	The tests relevant to the DCR	Yes
DCR Additional Products Affected	Products other than the one the DCR originated from that are effected	Yes
Modified Date	The last date the DCR was modified	Yes
DCR Date Ready For Closure	The date the DCR is marked as ready to close	Yes
Signed By	N/A	No
Modified By	The developers to modify the DCR	Yes
DCR Test Verification	How was the DCR verified	Yes
DCR Date Build Integration	The date the DCR was integrated	Yes
DCR/Test Leads Comments	Project lead comments	Yes
DCR Test Log Folder Verify	N/A	Yes
DCR Test Log Files Verify	N/A	Yes
Is Related To	Other DCR's the one under concern is related to	Yes
DCR Assigned To	The Developer the DCR was assigned to	Yes
Created By	The creater of the DCR	Yes
DCR Build Found	The build in which the DCR was found	Yes
DCR Test Procs Init Used	N/A	Yes
Assumed Issue Category	N/A	No

Appendix B

Table 6: CWE-888 Software Fault Pattern (SFP): Primiary and Secondary classes

Primary	Secondary	# of CWEs	Total CWEs	SFP #
Risky Values			31	
	Glitch in Computation	31		SFP1
Unused Entities			3	
	Unused Entities	3		SFP2
API			28	
	Use of an Improper API	28		SFP3
Exception Management			27	
	Unchecked Status Condition	17		SFP4
	Ambiguous Exception Type	2		SFP5
	Incorrect Exception Behavior	8		SFP6
Memory Access			20	
	Faulty Pointer Use	3		SFP7
	Faulty Buffer Access	11		SFP8
	Faulty String Expansion	2		SFP9
	Incorrect Buffer Length Computation	3		SFP10
	Improper NULL termination	1		SFP11
Memory Management			5	
	Faulty Memory Release	5		SFP12
Resource Management			17	
	Unrestricted Consumption	4		SFP13
	Failure to Release Resource	7		SFP14
	Faulty Resource Use	2		SFP15
	Life Cycle	4		-
Path Resolution			51	
	Path Traversal	43		SFP16
	Failed chroot Jail	1		SFP17
	Link in Resource Name Resolution	7		SFP18
Synchronization			22	
	Missing Lock	13		SFP19
	Race Condition Window	5	4	SFP20
	Multiple Locks/Unlocks	3		SFP21
	Unrestricted Lock	1		SFP22
Information Leak			96	
	Exposed Data	76		SFP23
	State Disclosure	7	0	-
	Exposure Through Temporary File	3		-
	Other Exposures	7		-
	Insecure Session Management	3		-

Primary	Secondary	# of CWEs	Total CWEs	SFP #
Tainted Input			138	
	Tainted Input to Command	87		SFP24
	Tainted Input to Variable	8		SFP25
	Composite Tainted Input	0		SFP26
	Faulty Input Transformation	15		-
	Incorrect Input Handling	17		-
	Tainted Input to Environment	11		SFP27
Entry Points			11	
	Unexpected Access Points	11		SFP28
Authentication			43	
	Authentication Bypass	10		-
	Faulty Endpoint Authentication	11		SFP29
	Missing Endpoint Authentication	2		SFP30
	Digital Certificate	6		-
	Missing Authentication	2		SFP31
	Insecure Authentication Policy	6		-
	Multiple Binds to the Same Port	1		SFP32
	Hardcoded Sensitive Data	4		SFP33
	Unrestricted Authentication	1		SFP34
Access Control			16	
	Insecure Resource Access	4		SFP35
	Insecure Resource Permissions	7		-
	Access Management	5		-
Privilege			12	
	Privilege	12		SFP36
Channel			13	
	Channel Attack	8		-
	Protocol Error	5		-
Cryptography			13	
	Broken Cryptography	5		-
	Weak Cryptography	8		-
Malware			11	
	Malicious Code	8		-
	Covert Channel	3		-
Predictability			15	
	Predictability	15		-
UI			14	
	Feature	7		-
	Information Loss	4		-
	Security	3		-
Other			46	
	Architecture	11		-
	Design	29		-
	Implementation	5		-
	Compiler	1		-

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD-MM-YYYY)* 01-01-2017	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)* 03/2016–12/2016

4. TITLE AND SUBTITLE Security Vulnerability Profiles of NASA Mission Software: Empirical Analysis of Security Related Bug Reports	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Katerina Goseva-Popstojanova, Jacob Tyo, and Brian Sizemore	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA	**8. PERFORMING ORGANIZATION REPORT NUMBER** L–
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA/TM–2017– ——-

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category 38
Availability:

13. SUPPLEMENTARY NOTES
An electronic version can be found at http://ntrs.nasa.gov.

14. ABSTRACT

15. SUBJECT TERMS
Software vulnerabilities, CWE, Vulnerability profiles, Ground mission, Flight mission

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Information Desk (`help@sti.nasa.gov`)
U	U	U	UU		**19b. TELEPHONE NUMBER** *(Include area code)* (757) 864-9658