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

Katerina Goseva-Popstojanova and Jacob Tyo
Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506, USA

Abstract—**While some prior research work exists on characteristics of software faults (i.e., bugs) and failures, very little work has been published on analysis of software applications vulnerabilities. This paper aims to contribute towards filling that gap by presenting an empirical investigation of application vulnerabilities. The results are based on data extracted from issue tracking systems of two NASA missions. These data were organized in three datasets: Ground mission IV&V issues, Flight mission IV&V issues, and Flight mission Developers issues. In each dataset, we identified security related software bugs and classified them in specific vulnerability classes. Then, we created the security vulnerability profiles, i.e., determined where and when the security vulnerabilities were introduced and what were the dominating vulnerabilities classes. Our main findings include: (1) In IV&V issues datasets the majority of vulnerabilities were code related and were introduced in the Implementation phase. (2) For all datasets, around 90% of the vulnerabilities were located in two to four subsystems. (3) Out of 21 primary classes, five dominated: Exception Management, Memory Access, Other, Risky Values, and Unused Entities. Together, they contributed from 80% to 90% of vulnerabilities in each dataset.**

1. Introduction

Nowadays, space missions provide valuable services to the society – from navigation, to earth observation, weather forecasting, and communication. Consequently, space is becoming a part of the critical infrastructure, regularly targeted by attackers. 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 [1]. Among these are sophisticated cyberattacks known as advanced persistent threats (APTs) [2]. Cyber threats to space missions are expected to continue to grow in the future [3]. 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. A multi-tiered approach to cybersecurity management integrates the IT service security, Data and Application security, and Infrastructure security [1]. This paper is focused on Application security throughout the software lifecycle. 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 space flight operations and/or steal sensitive data [4]. Therefore, it is becoming an imperative to use software development and assurance practices that account for cybersecurity concerns.

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). The terms 'vulnerability', 'security issue', and 'security related bug' are used interchangeably in this paper.

Our research is based on utilizing the information provided in issue tracking systems. We believe that issue tracking systems are valuable sources of information, with high potential for conducting empirical studies that could benefit both the research and practitioners communities. This paper is focused on studying software bugs reports and specifically identifying those that are security related. It should be noted that our study accounts for vulnerabilities that may have been introduced, found, and fixed throughout the software 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 describes where and when the security vulnerabilities were introduced and what were the dominating vulnerability classes. Uncovering the security vulnerability profiles and underling trends helps developers and Independent Verification and Validation (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 paper are based on data extracted from issue tracking systems of two NASA missions. These data were organized in three datasets: Ground mission IV&V issues, Flight mission IV&V issues, and Flight mission Developers issues. In each datasets, we identified 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 security issues' categories and types?
RQ2: Where are security issues located?
RQ3: When are the security related issues typically introduced and found?
RQ4: What are the severity levels of security related issues?

RQ5 What are the dominating classes of security issues?

RQ6 Are the dominating classes of security issues and the other findings consistent across missions and datasets?

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 missions' security. (Flight mission Developers issues dataset did not contain data in the Issue Category.)
- The location of security related issues, as the location of software bugs in general, followed the Pareto principle. Specifically, for all three datasets, around 90% of the security related issues were located in two to four subsystems.
- In both the Ground and Flight mission IV&V issues datasets, the majority of security issues (i.e., 91% and 80%, respectively) were introduced in the Implementation phase. In most cases, the phase in which the security 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 severity levels of most security issues were moderate, in all three datasets.
- Out of 21 primary security classes, the following five classes 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 rest of the paper is organized as follows. Section 2 summarizes the related works. Section 3 provides details on the classification approach. The basic facts about the two NASA missions used as case studies and the created datasets are given in section 4. The results on the security vulnerability profiles are presented in section 5, followed by the comparison of the results across the three datasets in section 6. The threats to validity are enumerated in section 7 and the conclusion is presented in section 8.

2. Related Work

While some prior research work exists on characteristics of software faults (i.e., bugs) and failures, very little work has been published on analysis of software applications vulnerabilities. We first summarize the papers that explored the characteristics of software faults in general [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], followed by a few prior works that were specifically focused on studying software application vulnerabilities [15], [16].

Fenton and Ohlsson studied a large telecommunication application from Ericsson Telecom AB [5]. 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.

Our previous research work, which was based on data extracted from a large NASA mission with over two millions lines of code, was focused on characterizing and quantifying relationships among faults, failures and fixes. The results showed that software failures were often associated with faults spread across multiple files [6]. The 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 [7]. In addition, 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 [8]. The results showed that 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 explored the effort associated with implementing the fix to correct all faults associated with an individual failure (i.e., fix implementation effort) [9]. 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 data, conducted by Grottke et al., analyzed 520 anomalies from the flight software of eighteen JPL space missions [10]. 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. In a follow up work, 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 [11].

Several papers explored some aspects of software faults for other application domains. Maji et al. utilized bug reports, bug fixes, developer reports, and failure reports to explore the manifestation of failures in Android and Symbian [12]. Frattini et al. analyzed a small dataset of 146 bug reports from the open source cloud platform Apache Virtual Computing Lab [13]. The analysis identified the components where bugs were likely to be found in future releases, the phases of the lifecycle 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 the open source software applications Ant, Maven, CMake, and QMake containing 199, 250, 200, and 151 bug reports, respectively [14]. Each sample was manually classified into several fault categories (e.g., external interface category, logic category, and configuration category).

The number of studies that were focused on studying software vulnerabilities is even smaller[15], [16]. Alhazmi et al. used density of vulnerabilities, dynamics of vulnerability discovery, and vulnerability discovery rate to estimate the magnitude of the undiscovered vulnerabilities still present in the system [15]. The analysis was based both on commercial and open-source software systems and the results revealed that the vulnerability densities fell within a range of values (similarly to fault density for general faults) and that the vulnerability discovery can be modeled using a logistic model. Venter et al. proposed an approach aimed at standardization of vulnerability categories using self-organizing maps (SOMs) and data extracted from CVE [16]. This work, however, did not provide quantification of the results.

3. Classification approach

In order to classify the security related bugs (i.e., vulnerabilities), a classification schema is needed. An obvious candidate for a classification schema is the Common Weakness and Enumeration (CWE) taxonomy of software weakness types, which serves as a common language for describing software security weaknesses in architecture, design, or code [17]. Each individual CWE represents a single vulnerability 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. 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 have been developed to ease the grouping of similar CWEs and provide simpler, more generalized structure. These include: CWE-2000 [18], CWE-1000 [19], CWE-888 [20], [21], CWE-700 [22], [23], and CWE-699 [24]. Upon close review, we selected CWE-888 Software Fault Pattern (SFP) View as classification schema because it provides very intuitive hierarchical structure, with a good trade-off between the level of details and generality. CWE 888 contains 705 CWEs organized in 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 bug was assigned a primary (more general) class and a secondary (more specific) class. Overall, there are 21 primary and 62 secondary classes (see Table 1). More detailed descriptions and the specific CWEs can be found in [20].

We conducted manual classification of software bug reports in all three datasets using the information provided in the "Title," "Subject," "Description," "Recommended Actions," and "Solution" fields from the issue tracking systems. Two examples of software bugs classification to the CWE-888 Primary and Secondary classes are described next. A bug report with description "…Null pointer dereference of 'getServiceStatusInfo(...)' where null is returned from a method," was classified as the primary class "Memory Access" and the secondary class "Faulty Pointer Use." A bug report 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 classified as the primary class "Resource Management" and the secondary class "Failure to Release Resource."

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

4. Description of the datasets

The three datasets used for this work were created by extracting relevant information from the issue tracking systems of two NASA missions. For all three datasets, only the issues that were marked as bug reports and were closed were included in the analysis.

The first dataset was extracted from the IV&V issue tracking system of a NASA ground mission and is referred to as *Ground mission IV&V issues*. The ground mission software consists 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 of the issues were marked as potentially security related. The issue descriptions contained security related information, making this a very good dataset for our research. Based on the manual classification, it appeared that 133 of the 350 security related issues (38%) could be assigned a specific CWE. The remaining security issues were tagged by the IV&V analysts as testing issues. Since testing issues do not deal with the actual software under investigation and no CWEs exist that cover such issues, testing issues were excluded from the analysis.

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. It should be noted that the IV&V issues of the Flight Mission neither were tagged as security related by the IV&V analysis nor the security aspects of the issues were specifically and consistently addressed in the issues' descriptions. We manually classified these 383 issues, out of which 157 issues appeared to be security related (i.e., 41% of all issues).

The third dataset consists of issues extracted from the Developers issue tracking system of the same NASA flight mission 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. Out of these 573 DCRs were marked as 'Defects' and were 'Closed'. As in case of the Flight mission IV&V issues dataset, security aspects of developers' issues were not specifically addressed in the descriptions and issues were not tagged as security / not security related. The manual classification of the 573 Developers bug reports led to 374

TABLE 1. 21 PRIMARY CLASSES, THEIR DEFINITIONS AND THE CORRESPONDING SECONDARY CLASSES

Primary class	**Definition and corresponding Secondary classes**
Risky Values	Relates to the basic uses of numerical values in software systems. Secondary class: Glitch in Computation.
Unused Entities	Covers unused entities in code, including unused procedures or variables. Secondary class: Unused Entities.
API	Relates to the use of Application Programming Interfaces (API). Secondary class: Use of an Improper API
Exception Management	Relates to management of exceptions and other status conditions. Secondary classes: Unchecked Status Condition, Ambiguous Exception Type, and Incorrect Exception Behavior.
Memory Access	Relates to access to memory buffers. Secondary classes: Faulty Pointer Use, Faulty Buffer Access, Faulty String Expansion, Incorrect Buffer Length Computation, Improper NULL termination.
Memory Management	Relates to the management of memory buffers. Secondary classes: Faulty Memory Release.
Resource Management	Relates to management of resources (i.e., dynamic entities). Secondary classes: Unrestricted Consumption, Failure to Release Resource, Faulty Resource Use, Life Cycle.
Path Resolution	Relates to access to file resources using complex file names. Secondary classes: Path Traversal, Failed Chroot Jail, Link in Resource Name Resolution
Synchronization	Relates to the use of shared resources. Secondary classes: Missing Lock, Race Condition Window, Multiple Locks/Unlocks, Unrestricted Lock.
Information Leak	Relates to the export of sensitive information from an application. Secondary classes: Exposed Data, State Disclosure, Exposure Through Temporary File, Other Exposures, Insecure Session Management.
Tainted Input	Relates to injection of user controlled data into various destination commands. Secondary classes: Tainted Input to Command, Tainted Input to Variable, Composite Tainted Input, Faulty Input Transformation, Incorrect Input Handling, Tainted Input to Environment.
Entry Points	Relates to unexpected entry points into the application. Secondary class: Unexpected Access Points.
Authentication	Relates to establishing the identity of an actor associated with the computation, or the identity of the endpoint involved in the computation through a certain channel. Secondary classes: Authentication Bypass, Faulty Endpoint Authentication, Missing Endpoint Authentication, Digital Certificate, Missing Authentication, Insecure Authentication Policy, Multiple Binds to the Same Port, Hardcoded Sensitive Data, Unrestricted Authentication.
Access Control	Relates to validating resource owners and their permissions. Secondary classes: Insecure Resource Access, Insecure Resource Permissions, Access Management.
Privilege	Relates to code regions with inappropriate privilege level. Secondary class: Privilege.
Channel	Relates to various protocol issues. Secondary classes: Channel Attack, Protocol Error.
Cryptography	Relates to cryptography issues. Secondary classes: Broken Cryptography, Weak Cryptography.
Malware	Relates to any malicious code present in the software systems. Secondary classes: Malicious Code, Covert Channel.
Predictability	Relates to random number generators and their properties. Secondary class: Predictability.
UI	Relates to security issues of User Interfaces (UI). Secondary classes: Feature, Information Loss, Security.
Other	Relates to miscellaneous architecture, design, and implementation issues. Secondary classes: Architecture, Design, Implementation, Compiler.

bug reports being marked as security related (i.e., 66% of all issues).

The basic facts of the two missions and the three datasets are summarized in Table 2.

TABLE 2. BASIC FACTS OF THE THREE DATASETS

Mission	Size	Total issues	Security issues	Dataset
Ground	1.3 MLOC	1,779	133	Ground mission IV&V issues
Flight	924 KLOC	506	157	Flight mission IV&V issues
		569	374	Flight mission Developers issues

5. Vulnerability profiles

5.1. Ground Mission IV&V Issues

Figure 1 shows the distribution of security and non-security issues across different Issue Categories. As shown, the Code category contained 95% of all security issues (i.e., 127 out of 133). Even though the Design category had the highest number of issues, only around 2% (i.e., three out of 133) of 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. Two most dominating issue types were Incomplete Code and Incorrect Code, which together contained 84% (112 of 133) of all security related issues.

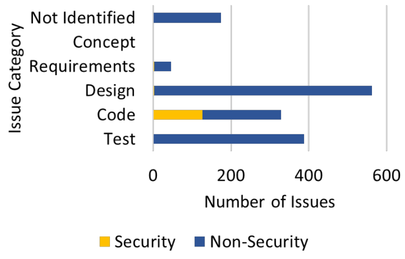

Figure 1. Issue Categories of the Ground mission IV&V issues

Figure 3 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 security related issues, as to the total number of issues.

Figure 4 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

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

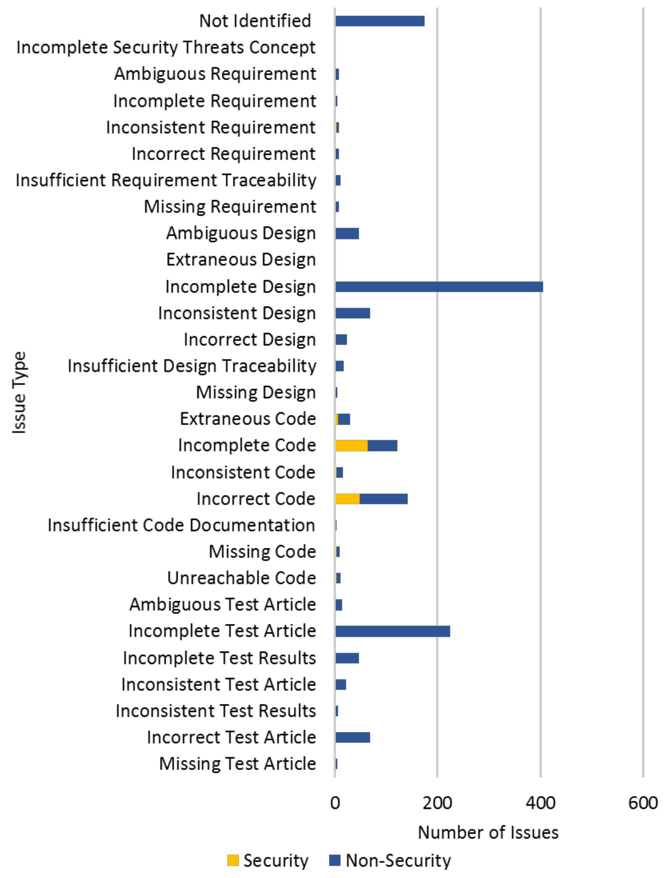

Figure 2. Issue Types of the Ground mission IV&V issues

method led to finding 91% of all security related issues[2]. It should be noted 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 4.

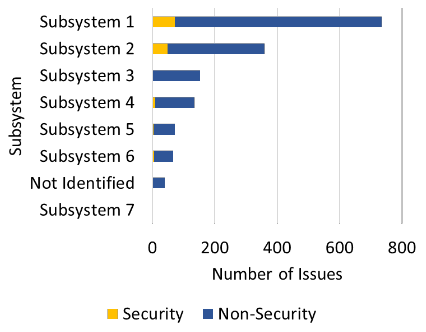

Figure 3. Distribution of Ground mission IV&V issues across subsystems

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

Figures 6 and 7 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 indicates how hard it is to implement secure code. This result also shows that efforts to enforce secure coding standards would lead to cost effective improvement of mission's security. Comparing Figures 6 and 7 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. (Note that this mission is under development and has not yet entered the Test phase.)

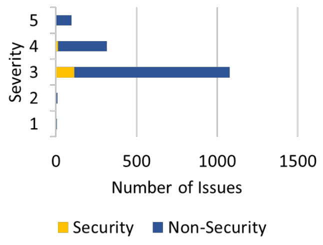

Figure 5. Severity levels of Ground mission IV&V issues

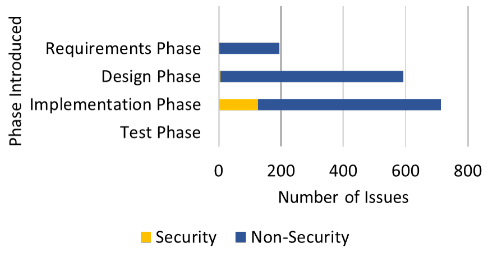

Figure 6. Ground mission IV&V issues: Phase Introduced

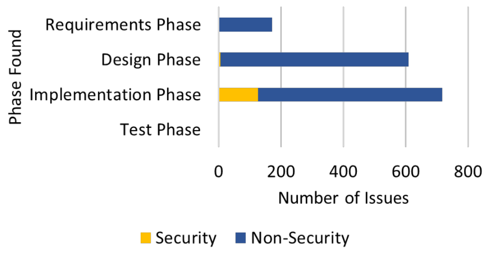

Figure 7. Ground Mission IV&V issues: Phase Found

Next, we focus on the distribution of the security related issues across the CWE-888 Primary classes. As can be seen in Figure 8, the IV&V security issues of the ground mission belonged to only 11 out of the total 21 CWE-888 primary classes. The Memory Access dominated, 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

2. Static code analysis tools are known to produce high number of false positives. In this case the output produced by the static code analysis tool was manually inspected by the IV&V analysts and only true positive warnings were entered as bug reports in the issue tracking system. Remember that only closed bug reports (i.e., fixed vulnerabilities) are included in the analysis.

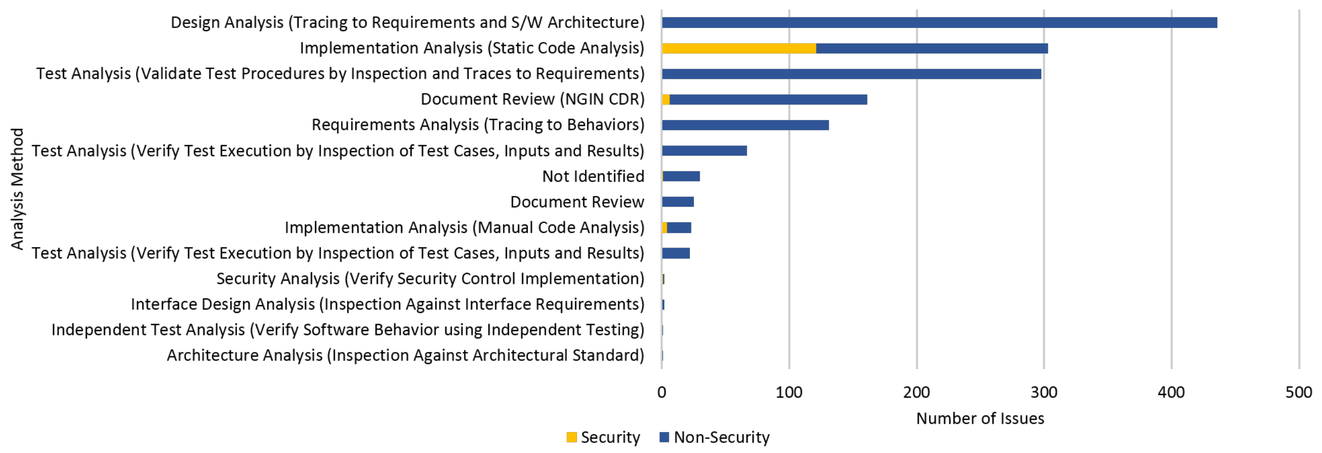

Figure 4. Distribution of Ground mission IV&V issues across Analysis Methods

the security issues across Primary classes as well.

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.

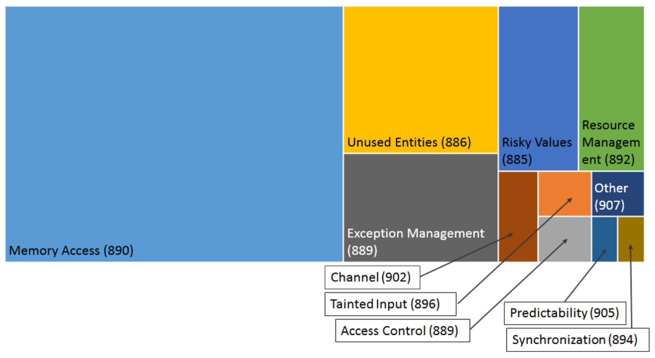

Figure 8. Ground mission IV&V issues: Distribution across CWE-888 Primary Classes. The numbers in brackets represent the specific CWE numbers of the Primary Classes.

5.2. Flight Mission IV&V Issues

As shown in Figure 9, 92% of all security related issues were associated with the Code Issue Category. This distribution of security related issues is consistent with the results for the Ground mission IV&V issues.

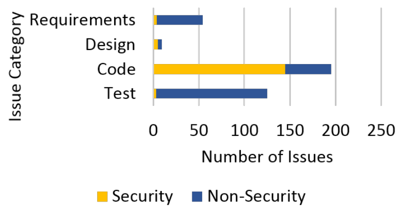

Figure 9. Issue Categories of Flight mission IV&V issues

Figure 10 shows the distribution of Flight mission IV&V issues across Issue Types, which provide more detailed information than Issue Category. The results show that security issues were predominately associated with Incorrect Code, Incomplete Code, Missing Code, and Extraneous Code.

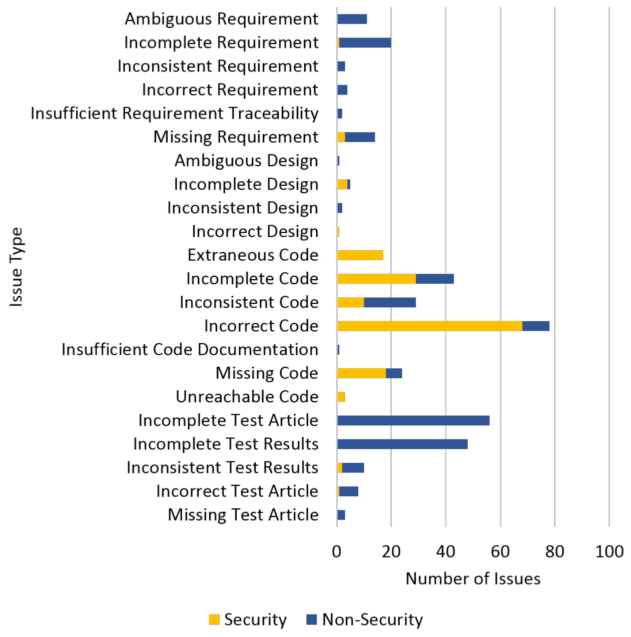

Figure 10. Issue Types of Flight mission IV&V issues

The distribution of issues across Flight mission subsystems presented in Figure 11 shows that 88% of all security issues (and 88% of all issues) fell into three out of five subsystems. (Note that Subsystems in Figure 11 are ordered by the total number of issues.)

As shown in Figure 12, 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.

Figures 13 and 14 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

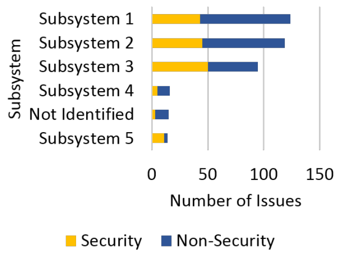

Figure 11. Distribution of Flight Mission IV&V issues across Subsystems

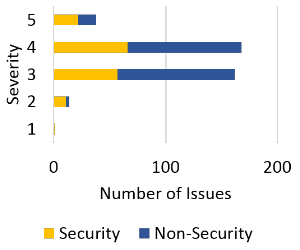

Figure 12. Severity levels of Flight Mission IV&V issues

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 15, 75% of security related issues were resolved in the Implementation phase, and the remaining 25% were resolved in the Testing phase. Interestingly, no security issues were resolved in the Design phase, even though some were introduced and found in that phase.

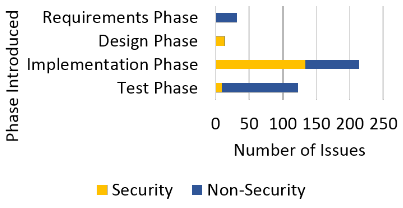

Figure 13. Flight Mission IV&V issues: Phase Introduced

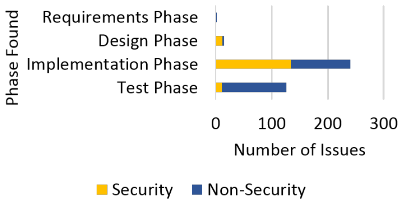

Figure 14. Flight Mission IV&V issues: Phase Found

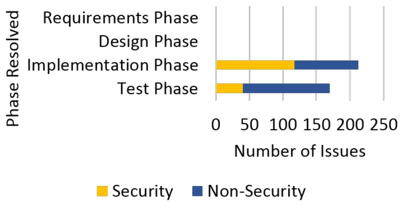

Figure 15. Flight Mission IV&V issues: Phase Resolved

Next, we focus on the distribution of security issues across the CSE-888 Primary classes, which is presented in Figure 16. Similarly as in the case of the Ground Mission IV&V Issues dataset, IV&V issues of the Flight mission belonged to only 9 out of the 21 Primary classes, with four dominating classes: Other, Risky Values, Memory Access, and Unused Entities.

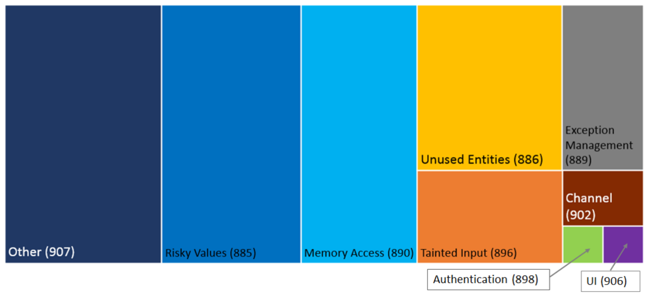

Figure 16. Flight Mission IV&V issues - Distribution of issues across CWE-888 Primary Classes. The numbers in brackets represent the specific CWE numbers of the Primary Classes.

5.3. Flight Mission Developers Issues

Figure 17 shows the distribution of security and non-security issues across Issue Types[3]. The two Issues Types – Incorrect Implementation and Incorrect Operation or Unexpected Behavior – significantly outnumbered the other Issue Types. (The Developer's issue tracking system did not contain the Issue Category field, which was present and populated in the IV&V issue tracking system.)

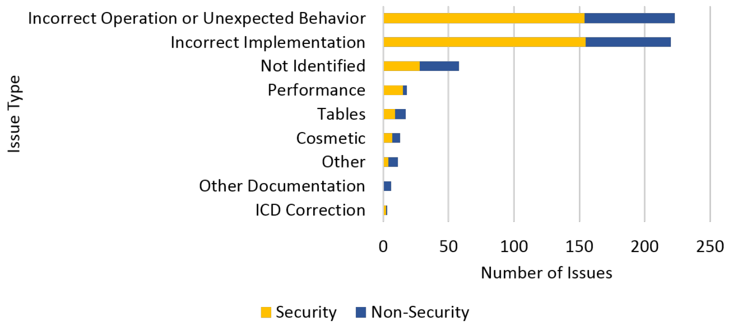

Figure 17. Issue Types of Flight mission Developer issues

Figure 18 presents the distribution of security and non-security issues across the Flight mission Subsystems. The finding is 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.

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 19, 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

3. 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.

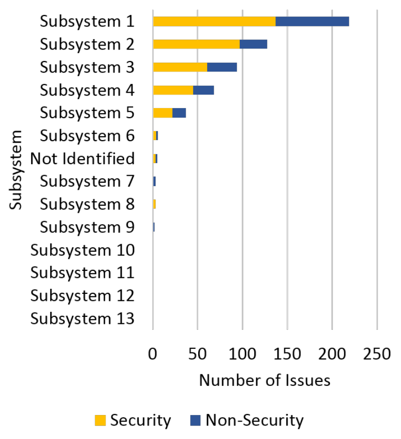

Figure 18. Distribution of Flight mission Developer issues across Subsystems

issues. Only 4% of all issues, and 4% of security issues were determined to be critical.

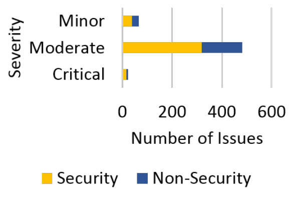

Figure 19. Severity levels of Flight mission Developer issues

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 missions. As shown in Figure 20, most issues were found during the following there phases listed in decreasing order: Build Verification, Build Integration, and Code Implementation. These more fine grained phases are consistent with the Phase Found categories that dominated the IV&V issues.

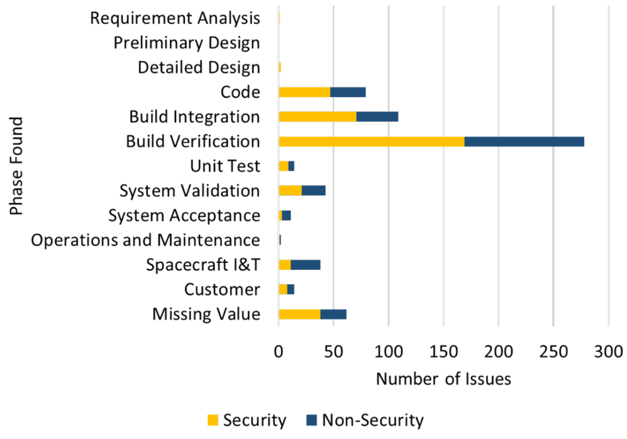

Figure 20. Flight mission Developer issues: Phase Found

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 21, only 13 of 21 Primary class were observed, with three dominating classes: Risky Values, Exception Management, and Memory Access. 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.

TABLE 3. Comparison of Primary classes (with nonzero security issues) across the three datasets. The five dominating classes are shared gray.

Primary CWE-888 Class	Ground Mission	Flight Mission	
	IV&V Issues	**IV&V Issues**	**Developer Issues**
API (887)			1.9%
Authentication (898)		0.9%	
Channel (902)		2.7%	6.0%
Exception Management (889)	10.8%	8.2%	27.2%
Memory Access (890)	54.6%	18.3%	12.8%
Memory Management (891)			0.4%
Other (907)	1.5%	24.5%	7.1%
Predictability (905)	0.8%		
Privilege (901)			1.2%
Resource Management (892)	6.9%		3.0%
Risky Values (885)	8.5%	28.3%	
Synchronization (894)	0.8%		3.4%
Tainted Input (896)	1.5%		8.2%
UI (906)		0.9%	1.1%
Unused Entities (886)	14.6%	14.5	3.8%

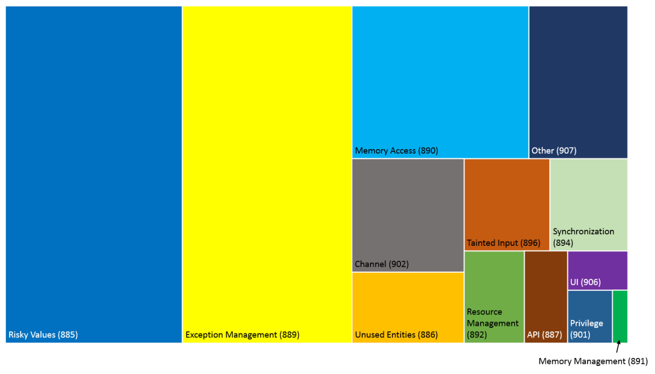

Figure 21. Flight mission Developer issues - Distribution across CWE-888 Primary classes

6. Comparison of the results across datastes

In this section we compare the results across all three datasets. We start with comparing the distribution of security issues across the CWE-888 Primary classes, extracted from the results presented in subsections 5.1, 5.2, and 5.3. As can be seen from Table 3 and Figure 22, 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 79% 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 4 shows the dominating CWE-888 Primary classes along with their corresponding Secondary classes, which provide more detailed information on the nature of security

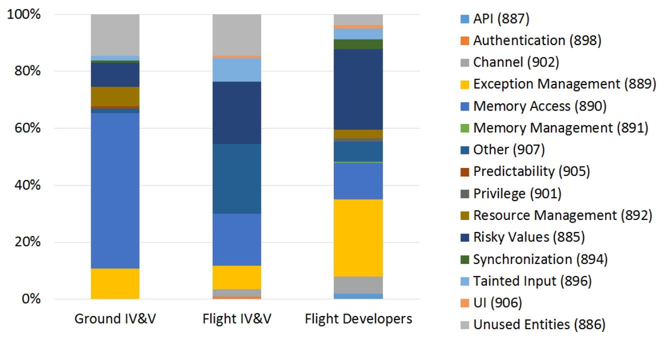

Figure 22. Distribution of security issues across CWE-888 Primary classes, for all three datasets

issues. The Secondary classes under the Exception Management Primary class included: Ambiguous Exception Type, Incorrect Exception Behavior, and Unchecked Status Condition. Security issues assigned to the Primary class Memory Access were distributed across the Secondary classes Faulty Buffer Access and Faulty Pointer Use. These categories include common programming errors such as null pointer dereferences and buffer overflows.

The Primary class Other had security issues distributed predominately in the Secondary class Implementation, which is based around weaknesses such as coding standards violation or containment errors. The Secondary classes Design and Architecture had significantly less issues. (Note that issues were assigned to the Primary class Other and the corresponding Secondary classes when they were related to these classes and could not be placed into any other class.)

The Primary class Risky Values consisted of the Secondary class Glitch in Computation, which deals with calculation errors such as divide by zero error or a function call with an incorrect order of arguments.

The Primary class Unused Entities consisted predominately of the Dead Code and much less of the Unused Variable Secondary class.

TABLE 4. COMPARISON OF THE SECONDARY CLASSES, ONLY FOR THE FIVE DOMINANT PRIMARY CLASSES.

Secondary class	Ground Mission	Flight Mission	
	IV&V Issues	IV&V Issues	Developer Issues
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%
Memory Access (890)			
Faulty Buffer Access (970)	4.6%	12.7%	8.3%
Faulty Pointer Use (971)	50.0%	5.5%	4.5%
Other (907)			
Architecture (975)		0.9%	
Design (977)			2.6%
Implementation (978)	1.5%	23.6%	4.5%
Risky Values (885)			
Glitch in Computation (998)	8.5%	21.8%	28.3%
Unused Entities (886)			
Dead Code (561)	14.6%	10.0%	3.4%
Unused Variable (563)		4.5%	0.4%

Table 5 summarizes the main findings as they pertain to RQ1 - RQ5 and helps identifying trends that are consistent across datasets (i.e., RQ6). 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 (66%). The lowest proportion of security related issues in the Ground mission IV&V dataset may be due to 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 and the Flight mission IV&V security issues, with 95% and 92%, respectively. This is an important finding because it indicates that 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 cost effective ways to improve mission's security. (Note that the Flight mission Developers issues dataset did not contain the data in Issue Category.)

The location of security issues, as the location of software bugs in general, followed the Pareto principle. Specifically, around 90% the security issues were located in two to four subsystems, for all three datasets. This result is consistent with related works focused on fault characterization [5], [6], [8], and [25].

In both the Ground mission and Flight mission IV&V issues datasets majority of issues (i.e., 95% and 85%, 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.

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.

The final row in Table 5 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.

7. Threats to Validity

In this section we discuss the threats to validity to our study. **Construct validity** is concerned with whether we are measuring what we intend to measure. The number and classes of identified security issues depend on the quality of software artifacts, as well as the level of provided details related to security. In the case of the Ground mission IV&V dataset, there was a significant number of security related

TABLE 5. MAIN FINDINGS ACROSS ALL DATASETS

	Ground mission IV&V issues	Flight mission IV&V issues	Flight mission Developers issues	RQ
% Security Issues	9%	41%	66%	
Security Issues Category	95% Code related	92% Code related	Data not available	RQ1
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)	RQ2
Phase Introduced	95% in the Implementation Phase	85% in the Implementation Phase	Data not available	RQ3
Phase Found	Followed closely the phase introduced distribution	Followed closely the phase introduced distribution	Most found during Code Implementation, Build Integration, and Build Verification	
Severity of security issues	Level 3 dominated (86%)	Levels 3 and 4 dominated (together 78%).	Moderate dominated (84%)	RQ4
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%	RQ5

issues that were tagged as testing related. Since no CWE exists that covers such cases and testing issues are not related to software itself, these testing related security issues were not included in the further analysis. Another threat to construct validity is related to the fact that, in general, some security issues could be correctly classified into multiple CWE classes. In our case, this threat was partially mitigated by the fact that the selected classification schema CWE 888 has a hierarchical structure. Even more, the number of issues fitting into multiple CWE classes was small, and for these cases the most relevant of the possible classes was selected.

Internal validity threats are concerned with unknown influences that may affect independent variables. Data quality is one of the major concerns to the internal validity. It should be noted that NASA issue tracking systems follow high record keeping standards, which provides some guarantee for the quality and consistency of data. As mentioned in section 4, in the case of the Flight mission datasets (both the IV&V issue and Developers issues) software issues were not tagged as security related and security aspects of software bugs were not explicitly addressed in the descriptions. Therefore, it is possible that the available information did not account for potential security implications.

Conclusion validity threats impact the ability to draw correct conclusions. One threat to conclusion validity is related to data sample sizes. The work presented in this paper is based on three dataset, with a total of 2,854 issues, out of which 664 were classified security related. The number of security issues per dataset ranged from 133 to 374, which are sufficiently large numbers to develop vulnerability profiles. The results and conclusions may be affected by the fact that 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. The analysis methods in our case were explicitly available for only one of the datasets. However, we have confirmed with the NASA personnel that the V&V and IV&V activities for both missions (and the three datasets) spanned the whole software lifecycle and were focused on different software artifacts, including requirements, design, and code.

External validity is concerned with the ability to generalize results. The breadth of this study, including the facts that (1) it is based on two large NASA mission containing around one millions lines of code each and (2) the missions were developed by different teams over multiple years, allow for some degree of external validation. Nevertheless, we cannot claim that the results would be valid across all software products. Generalizations are based on multiple empirical studies that have replicated the same phenomenon under different conditions. Since most of the analyses were done for the first time in this paper, the external validity remains to be established by future similar studies that will use other software products as case studies.

8. Conclusion

While prior empirical work on characteristics of software faults (i.e., bugs) and failures exists, few works focused on analyses of software application vulnerabilities have been published. This paper aims at filling out that gap and contributing towards building an evidence-based knowledge about different aspects of software vulnerabilities.

The empirical findings presented in the paper are based on the data extracted from the issue tracking systems of two NASA missions. Using the extracted data, organized in three datasets, we built so called security vulnerability profiles that address several aspects of software application vulnerabilities, such as where and when the security vulnerabilities were introduced and what were the dominating vulnerabilities classes. An important aspect of this paper is the identification of trends that are consistent across the three datasets.

The main findings of this work indicate that the majority of vulnerabilities were code related and were introduced in the Implementation phase, and that the Pareto principle (i.e., uneven distributions) applied both to the location of vulnerabilities across subsystems and to the distribution across different vulnerability classes. It appears that development and testing efforts focused on these vulnerability prone subsystems and dominant security classes provide the most cost efficient ways to improve missions' security.

We believe that mining the information related to software vulnerabilities is helpful for building a knowledge base that can be reused on other similar systems. Exploring the same research questions on other case studies (from the space and other domains) would test the generalizability of our findings and help establishing characteristics of software application vulnerabilities that are invariant across different software systems.

Acknowledgments

This work was funded in part by the NASA Software Assurance Research Program (SARP) in 2016 fiscal year. Any opinions, findings, and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agency. The authors thank the following NASA personnel for their support: Brandon Bailey, Craig Burget, and Dan Painter.

References

[1] "NASA cybersecuirty presentation," NASA Office of the Chief Information Officer, Nov 2014.

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

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

[4] "Inadequate security practices expose key NASA network to cyber attack," Office of Inspector General, Audit report, May 2011.

[5] 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.

[6] 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.

[7] ——, "Exploring the missing link: an empirical study of software fixes," *Software Testing, Verification and Reliability*, vol. 24, no. 8, pp. 684–705, 2014.

[8] ——, "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.

[9] ——, "Analyzing and predicting effort associated with finding and fixing software faults," *Information and Software Technology*, vol. 87, pp. 1–18, 2017.

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

[11] 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 *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2013, pp. 1–8.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] H. Venter, J. Eloff, and Y. Li, "Standardising vulnerability categories," *Computers & Security*, vol. 27, no. 3–4, pp. 71 – 83, 2008.

[17] "Common Weakness Enumeration," January 2017, https://cwe.mitre.org/ (accessed May 4, 2017).

[18] "CWE-2000: Comprehensive CWE dictionary, MITRE Corporation," https://cwe.mitre.org/data/slices/2000.html.

[19] "CWE-1000: Research concepts, MITRE Corporation," https://cwe.mitre.org/data/graphs/1000.html.

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

[21] "CWE-888: Software fault pattern (SFP) clusters, MITRE Corporation," https://cwe.mitre.org/data/graphs/888.html.

[22] 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.

[23] "CWE-700: Seven pernicious kingdoms, MITRE Corporation," https://cwe.mitre.org/data/definitions/700.html.

[24] "CWE-699: Development concepts, MITRE Corporation," https://cwe.mitre.org/data/graphs/699.html.

[25] 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.