

## ABSTRACT

SUBRAMANI, SHWETA. Security Profile of Fedora. (Under the direction of Dr. Mladen A. Vouk.)

The process of software development and evolution has proven difficult to improve. For example, well documented security issues such as SQL injection (SQLi), after more than a decade, still top most vulnerability lists. Quantitative security process and quality metrics are often subdued due to lack of time and resources. Security problems are hard to quantify and even harder to predict or relate to any process improvement activity.

The goal of this thesis is to assess usefulness of “classical” software reliability engineering (SRE) models in the context of open source software security, the conditions under which they may be useful, and the information that they can provide with respect to the security quality of a software product.

We start with security problem reports for open source Fedora series of software releases. We illustrate how one can learn from normal operational profile about the non-operational processes related to security problems. One aspect is classification of security problems based on the human traits that contribute to the injection of problems into code, whether due to poor practices or limited knowledge (epistemic errors), or due to random accidental events (aleatoric errors). Knowing the distribution aids in development of an attack profile. In the case of Fedora, the distribution of security problems found post-release was consistent across four different releases of the software. The security problem discovery rate appears to be roughly constant but much lower than the initial non-security problem discovery rate. Previous work has shown that non-operational testing can help accelerate and focus the problem discovery rate and that it can be successfully modeled. We find that some classical reliability models can be used with success to estimate the residual number of security problems, and through that provide a measure of the security characteristics of the software. We propose an agile software testing process that combines operational and non-operational (or attack related) testing with the intent of finding more security problems faster.

© Copyright 2014 by Shweta Subramani

All Rights Reserved

Security Profile of Fedora

by  
Shweta Subramani

A thesis submitted to the Graduate Faculty of  
North Carolina State University  
in partial fulfillment of the  
requirements for the Degree of  
Master of Science

Computer Science

Raleigh, North Carolina

2014

APPROVED BY:

---

Dr. Aldo Dagnino

---

Dr. Emerson Murphy-Hill

---

Dr. David Wright

---

Dr. Mladen A. Vouk  
Chair of Advisory Committee

## **DEDICATION**

To my grandparents, my parents, my younger brother Abhishek and my friend Vivek for their unconditional love and support.

## **BIOGRAPHY**

Shweta Subramani was born in Mumbai, India. She did her primary and secondary schooling from Smt. Sulochanadevi Singhanian School, Thane, India. She received her Bachelor's degree in Computer Engineering from the University of Mumbai in 2012. She joined North Carolina State University in the Fall of 2012 for pursuing her Master's degree in Computer Science. Apart from academics Shweta has professional training in the Carnatic vocal style of Indian Classical Music.

## ACKNOWLEDGEMENTS

There are two kinds of teachers: the kind that fill you with so much quail shot that you can't move, and the kind that just gives you a little prod behind and you jump to the skies

—Robert Frost

I would like to thank my advisor Dr. Mladen A. Vouk for having believed in me throughout this journey. Every step of mine where I have been unsure of myself, he has always encouraged me to place it confidently. Thanks to my advisor, because of whom all the little dreams that I carried on my way here from India could be fulfilled. I am extremely grateful for his invaluable guidance in performing this research, writing this thesis and his constant support throughout the course of my graduate studies. I would also like to express my gratitude to Dr. Aldo Dagnino, Dr. Emerson Murphy-Hill and Dr. David Wright for their kind consent to serve on my thesis defense committee.

Two people who have been instrumental in me undertaking this journey are Vivek and Vikas, whom I would like to thank for inspiring me and believing in my capabilities.

This work is supported in part through NSF grants 0910767 and 1330553, the U.S. Army Research Office (ARO) grant W911NF-08-1-0105 managed by the NCSU Science of Security Initiative and the NSA Science of Security Lablet, and by the IBM Share University Research and Fellowships program Funding.

## TABLE OF CONTENTS

<b>LIST OF TABLES</b> .....	<b>vii</b>
<b>LIST OF FIGURES</b> .....	<b>viii</b>
<b>Chapter 1 Introduction</b> .....	<b>1</b>
1.1 Motivation and Goals .....	1
1.2 Terminology .....	4
1.3 Software Reliability Engineering .....	4
1.4 Software Security Engineering .....	5
1.5 Approach .....	6
1.6 Outline .....	7
<b>Chapter 2 Related Work</b> .....	<b>8</b>
2.1 Vulnerabilities .....	8
2.2 Operational Profiles .....	9
2.2.1 Determining the Operational Profile .....	10
2.3 Reliability Models .....	11
2.4 Non-Operational Profiles .....	12
2.4.1 Attack Profiles .....	14
2.5 Epistemic Versus Aleatory .....	14
2.5.1 Uncertainty in Software Security Engineering .....	15
2.6 Classifying Security Problems .....	15
2.7 Approaches to Non-Operational Testing .....	17
<b>Chapter 3 Data</b> .....	<b>18</b>
3.1 Data Sources .....	18
3.1.1 Common Weakness Enumeration (CWE) .....	18
3.1.2 Bugzilla for RedHat Fedora .....	19
3.1.3 CVE, NVD and OSVDB .....	22
3.2 Classifying the CWE Top 25 .....	23
3.2.1 Summary - CWE Top 25 .....	26
3.3 Classification Scheme .....	26
3.4 Examples in Fedora .....	27
3.4.1 Epistemic Security Problems .....	27
3.4.2 Aleatoric Security Problems .....	32
3.4.3 Security problems classified on the “Border” of epistemic and aleatoric problems or left “Unclassified” .....	34
3.5 Results .....	36
3.6 Time Distribution .....	36
3.6.1 Fedora Core Substrate .....	37
3.6.2 Fedora Applications Substrate .....	40
<b>Chapter 4 Security Reliability Modeling</b> .....	<b>43</b>

4.1	The Yamada Delayed S-Shaped Software Reliability Growth Model . . . . .	43
4.1.1	Model Assumptions [70] . . . . .	43
4.1.2	Notation . . . . .	44
4.1.3	Model Form . . . . .	44
4.2	The CASRE Tool . . . . .	45
4.2.1	Available Models . . . . .	45
4.2.2	Using the tool . . . . .	46
4.2.3	CASRE Fits for Fedora Security Data . . . . .	47
4.2.4	CASRE fits for Fedora 15 non-security data . . . . .	55
4.3	Summary . . . . .	56
<b>Chapter 5 Operational Profile Discovery and Adjustment . . . . .</b>		<b>57</b>
5.1	Hybrid Approach . . . . .	57
5.2	Non-Operational Profile Generation . . . . .	59
5.3	Accounting for Risk . . . . .	62
5.4	Summary . . . . .	65
<b>Chapter 6 Conclusions and Future Work . . . . .</b>		<b>67</b>
6.1	Conclusions . . . . .	67
6.2	Future Work . . . . .	68
<b>References . . . . .</b>		<b>70</b>
<b>Appendices . . . . .</b>		<b>80</b>
Appendix A Fedora Data . . . . .		81
A.1	Fedora Operational Usage Weekly Problem Reports . . . . .	81
A.2	Categorical Distribution Results for Security Problem Reports in Fedora . . . . .	83
A.3	Time Distribution Results for Security Problem Reports in Fedora . . . . .	84
Appendix B CASRE Input Data Format . . . . .		87
B.1	Input Data Format . . . . .	87

## LIST OF TABLES

Table 3.1	2011 CWE/SANS Top 25 Most Dangerous Software Errors . . . . .	20
Table 3.2	Fedora Core (Kernel) Security Problems (Actual Data) . . . . .	38
Table 4.1	CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Security Data . . . .	47
Table 4.2	CASRE: Yamada S-Shaped Model Specifications for Fedora 16 Security Data . . . .	48
Table 4.3	CASRE: Yamada S-Shaped Model Specifications for Fedora 17 Security Data . . . .	50
Table 4.4	CASRE: Yamada S-Shaped Model Specifications for Fedora 18 Security Data . . . .	51
Table 4.5	CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Security Data Using 12 Weeks Data . . . . .	53
Table 4.6	CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Security Data Using 24 Weeks Data . . . . .	53
Table 4.7	CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Security Data Using 36 Weeks Data . . . . .	53
Table 4.8	CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Non-Security Data .	55
Table 5.1	Statistics for CVSS Scores Assigned to Fedora 15 Security Problems . . . . .	63
Table A.1	Fedora v15 Security Problems Categorical Distribution . . . . .	83
Table A.2	Fedora v16 Security Problems Categorical Distribution . . . . .	83
Table A.3	Fedora v17 Security Problems Categorical Distribution . . . . .	83
Table A.4	Fedora v18 Security Problems Categorical Distribution . . . . .	84
Table A.5	Fedora v15 Security Problems Time Distribution . . . . .	84
Table A.6	Fedora v16 Security Problems Time Distribution . . . . .	85
Table A.7	Fedora v17 Security Problems Time Distribution . . . . .	85
Table A.8	Fedora v18 Security Problems Time Distribution . . . . .	86
Table B.1	Fedora 15 Prepared Data (Security Problems)- fv15.dat . . . . .	87
Table B.2	Fedora 16 Prepared Data (Security Problems)- fv16.dat . . . . .	89
Table B.3	Fedora 18 Prepared Data (Security Problems)- fv18.dat . . . . .	91
Table B.4	Fedora 18 prepared Data (Security Problems)- fv18.dat . . . . .	93

## LIST OF FIGURES

Figure 1.1	Fedora v15 Weekly Problem Report Rates Under Operational Usage Over Calendar Time . . . . .	2
Figure 1.2	Fedora v15 Usage over Calendar Time . . . . .	2
Figure 1.3	Fedora v15 Problem Report Rates Under Operational Usage over Inservice Time . . . . .	3
Figure 2.1	Reliability and Failure Intensity . . . . .	11
Figure 2.2	The <i>Sandwich</i> or <i>Layered</i> testing model [103] . . . . .	13
Figure 2.3	The <i>Functional Groups</i> or <i>Disjoint</i> Testing Model [103] . . . . .	13
Figure 3.1	The Red Hat Family Tree [4] . . . . .	21
Figure 3.2	The Classification Scheme . . . . .	28
Figure 3.3	Fedora v15 Security Problems Categorical Distribution . . . . .	37
Figure 3.4	Fedora v16 Security Problems Categorical Distribution . . . . .	37
Figure 3.5	Fedora v17 Security Problems Categorical Distribution . . . . .	38
Figure 3.6	Fedora v18 Security Problems Categorical Distribution . . . . .	38
Figure 3.7	Fedora v15 Core Security Problems Time Distribution . . . . .	39
Figure 3.8	Fedora v16 Core Security Problems Time Distribution . . . . .	39
Figure 3.9	Fedora v17 Core Security Problems Time Distribution . . . . .	40
Figure 3.10	Fedora v18 Core Security Problems Time Distribution . . . . .	40
Figure 3.11	Fedora v15 Applications Security Problems Time Distribution . . . . .	41
Figure 3.12	Fedora v16 Applications Security Problems Time Distribution . . . . .	41
Figure 3.13	Fedora v17 Applications Security Problems Time Distribution . . . . .	42
Figure 3.14	Fedora v18 Applications Security Problems Time Distribution . . . . .	42
Figure 4.1	Fedora v15 Number of Security Problem Reports Per Week - Yamada fit . . . . .	47
Figure 4.2	Fedora v15 Cumulative Number of Security Problem Reports - Yamada fit . . . . .	48
Figure 4.3	Fedora v16 Number of Security Problem Reports Per Week - Yamada fit . . . . .	49
Figure 4.4	Fedora v16 Cumulative Number of Security Problem Reports - Yamada fit . . . . .	49
Figure 4.5	Fedora v17 Number of Security Problem Reports Per Week - Yamada fit . . . . .	50
Figure 4.6	Fedora v17 Cumulative Number of Security Problem Reports - Yamada fit . . . . .	50
Figure 4.7	Fedora v18 Number of Security Problem Reports Per Week - Yamada fit . . . . .	52
Figure 4.8	Fedora v18 Cumulative Number of Security Problem Reports - Yamada fit . . . . .	52
Figure 4.9	Fedora v15 Cumulative Number of Security Problem Reports - 95% Bounds Using 12 Weeks Data . . . . .	54
Figure 4.10	Fedora v15 Cumulative Number of Security Problem Reports - 95% Bounds Using 24 Weeks Data . . . . .	54
Figure 4.11	Fedora v15 Cumulative Number of Security Problem Reports - 95% Bounds Using 36 Weeks Data . . . . .	54
Figure 4.12	Fedora v15 Number of Non-Security Problem Reports Per Week - Yamada fit . . . . .	55
Figure 4.13	Fedora v15 Cumulative Number of Non-Security Problem Reports - Yamada fit . . . . .	56
Figure 5.1	The Hybrid Approach . . . . .	58
Figure 5.2	Fedora v15: A Draft Attack Profile and Remedy . . . . .	60

Figure 5.3	Fedora v16: A Draft Attack Profile and Remedy .....	60
Figure 5.4	Fedora v17: A Draft Attack Profile and Remedy .....	61
Figure 5.5	Fedora v18: A Draft Attack Profile and Remedy .....	61
Figure 5.6	Accounting for Risk in Fedora v15 - Step One .....	63
Figure 5.7	Accounting for Risk in Fedora v15 - Step Two .....	64
Figure 5.8	Accounting for Risk in Fedora v15 - Step Three .....	64
Figure 5.9	Accounting for Risk in Fedora v15 - Step Four .....	65
Figure A.1	Fedora v16 Weekly Problem Report Rates Under Operational Usage Over Calendar Time .....	81
Figure A.2	Fedora v16 Weekly Problem Report Rates Under Operational Usage Over Calendar Time .....	82
Figure A.3	Fedora v17 Weekly Problem Report Rates Under Operational Usage Over Calendar Time .....	82

## 1.1 Motivation and Goals

Quantitative analysis plays an important role in understanding software security [43]. Security metrics and models can inform the selection of software/hardware development and deployment processes and practices; the prioritization of effort towards the areas of the system that indicate the highest security risk; decisions about system architecture and design; and the release readiness of a system that preserves the desired set of security properties [81]. Software Reliability Engineering (SRE) is a metrics-based software development practice used by teams to choose development practices to achieve their desired reliability objective [78]. This thesis explores the use of SRE in the context of software security problems.

Anbalagan [43] made an interesting observation about the number of security problems disclosed per unit time being approximately constant in open source products he examined. When viewed from the calendar perspective, it may appear that the security quality of the software may not be improving. This is illustrated in Figure 1.1 which shows history of Fedora v15 problem reports over some 40+ weeks after its release date. Vertical axis shows the number of problems reported and closed per week [3]. Note that vertical axis is logarithmic. Figure 1.2 shows the growth in usage of the release. The usage curve (number of downloads) is an upper bound estimated from reported downloads of the release to unique IP numbers. As usage grows, there are less non-security problem reports per calendar week. The reporting rate for security problems appears to be low (few problems per week versus tens or hundreds of problems a week for non-security faults), and roughly constant in calendar time. These seem to indicate that under normal operational profile (or usage in the field)

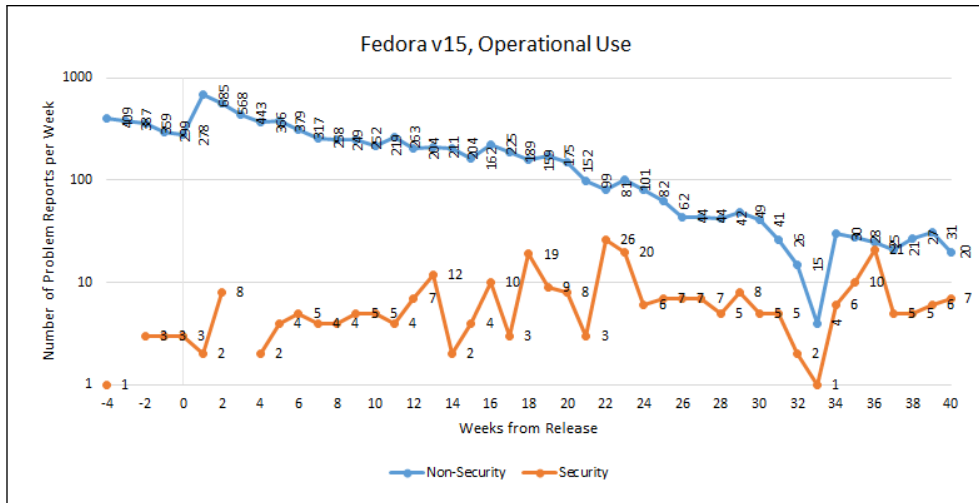


Figure 1.1: Fedora v15 Weekly Problem Report Rates Under Operational Usage Over Calendar Time

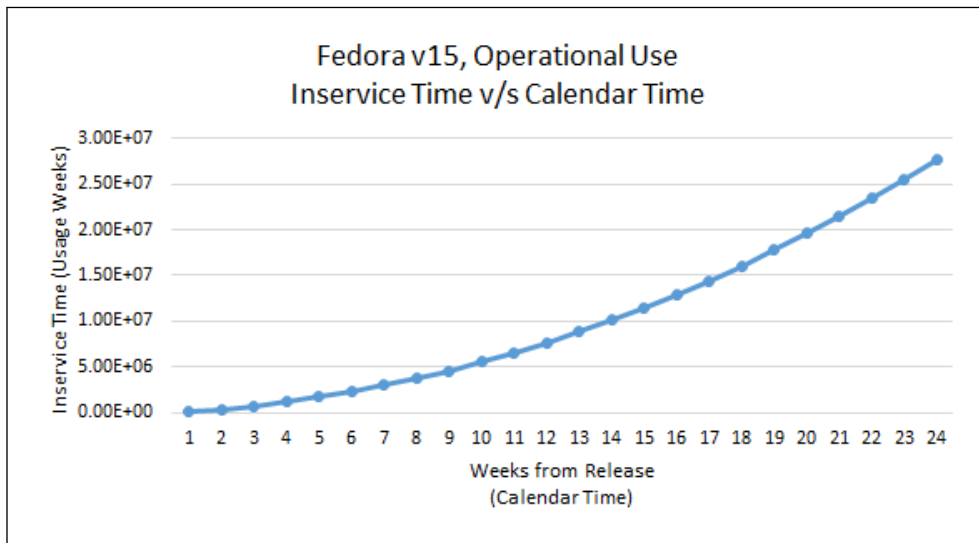


Figure 1.2: Fedora v15 Usage over Calendar Time

security problem discovery is somewhat of a random, perhaps accidental, process of approximately constant intensity, i.e., apparently there is no growth in the security reliability of the product [99]. The ratio of security to non-security problems exhibits unusual behavior. For example, in week one, only about 1% of the new problem reports are security related, in week 38 between 30% and 60% of the problems appear to relate to security. Are “most” residual problems (faults) in highly reliable software vulnerabilities? The good news appears to be that for Fedora [65] and a number of other open source products, [43], classical reliability models may be adapted to estimate the number of

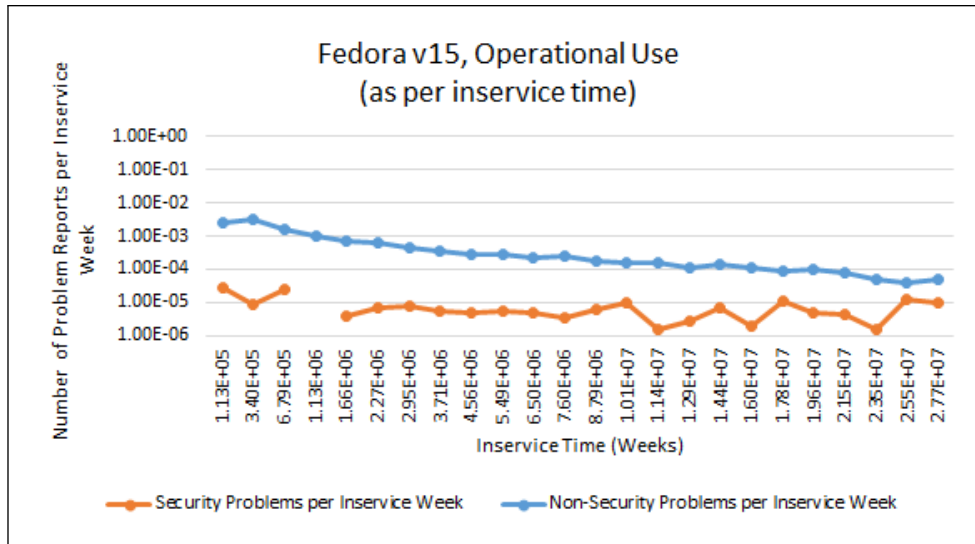


Figure 1.3: Fedora v15 Problem Report Rates Under Operational Usage over Inservice Time

residual security problems under “normal” operational usage (not attacks). However, predictive ability of these models is lower for security faults due to the rarity of security events and because there appears to be no real security reliability growth.

This thesis examines the problem reports for Fedora release 15, 16, 17 and 18 over a period of about 40 weeks after each release to:

- see whether Anbalagan’s findings ([43]) for earlier releases of Fedora still apply, and
- see if SRE models and methods can be used to assess and possibly improve security properties of following releases

One aspect of our empirical analysis was to classify security problems based on the human traits that contribute to the injection of problems into code, whether due to poor practices or limited knowledge(epistemic errors), or due to random accidental events(aleatoric errors). Knowing the distribution of security problems can then aid in development of non-operational attack test cases that can be applied in the next development cycle. In the case of Fedora, the distribution of security problems found post-release was consistent across different releases of the software. It appears that such information can be leveraged in planning the next stage of testing or predicting the vulnerability profile of the product for the rest of its lifespan.

## 1.2 Terminology

Security has many definitions. We focus on its meaning in the context of information technology, software, and cyber-world. The Institute for Secure and Open Methodologies (ISECOM) in the 3rd version of the Open Source Software Testing Methodology Manual (OSTMM 3)[62] defines security as '*a form of protection where a separation is created between the assets and the threat*'. It is a condition that results from the establishment and maintenance of protective measures that enable an enterprise to perform its mission or critical functions despite risks posed by threats to its use of information systems. Protective measures may involve a combination of deterrence, avoidance, prevention, detection, recovery and correction that should form part of the enterprise's risk management approach [89].

In the context of this work, we will assume that security problems or issues are a much smaller and less frequent subset of a general category of software problems and issues. Specifically, when a human or a machine makes an error, it results in a root fault (vulnerability) of either omission or commission. If this fault is not removed right away it may propagate into subsequent software artifacts as a defect, and it may even be amplified in the process. Activities such as verification, validation, inspections, and testing are intended to catch and remove those defects before they reach code operating in the field. If a defect is encountered in operation, it may put the executing software into an error-state. Unless that error-state is recognized or masked, it may result in an observable anomaly in the software behavior or its outputs. An anomaly may indicate a software failure, a vulnerability probe, or an exploit. During software development phases, we assume that discovered defects would be repaired, and if there is growth in the reliability of the software, there would be less and less of new problems reported as software nears its release date. In operation (under normal operational profile), reported problems would also be fixed through patches. Again we would say that software is becoming more reliable as less and less of new problems are observed in the field.

## 1.3 Software Reliability Engineering

The discipline of software engineering, in addition to guiding development and delivery of a software product on time and within cost, has to also monitor for certain quality criteria. The most important is reliability. Software Reliability Engineering(SRE) is the applied Science of predicting, measuring, and managing the reliability of software based systems to maximize customer satisfaction [78]. Musa fathered the practice of SRE in 1973 and utilized it in AT&T[79]; SRE was declared a Best Current Practice by AT&T in 1991 and is today in some form, used by numerous organizations. SRE has the following key practices [69]:

1. Reliability Objective: Determines the customer's *tolerance threshold*. Examples are number of

field failures per unit time, number of residual faults, mean-time-to failure, etc.

2. **Operational Profile:** Describes the system's likely operational usage. It is often presented as the frequency of operations (inputs) requested from a system when it is in normal use in the field. In contrast, a non-operational profile are frequency distributions that would not be expected as the result of normal benign use of the software, but might be expected when cyber-attacks occur. The latter deliberately distort operational profile in order to increase the probability of finding or exploiting vulnerabilities.
3. **Reliability Modeling and Measurement:** Determines if the product has met its reliability objectives. During development reliability estimates give the following information:
  - (a) The reliability of the product at the end of system testing.
  - (b) The amount of additional test time required to reach the product's reliability objective.
  - (c) The reliability growth as a result of product testing.
  - (d) The predicted reliability beyond the system testing.
4. **Reliability Validation:** Compares the projected with the observed field reliability to provide feedback into the SRE process for continuous improvement and better parameter tuning.

While SRE gives a proven best practice method to make software more reliable, consistent engineering of secure software is still somewhat of an art. In part this may be because security problems are in reality rare events and security faults are hard to find during normal testing unless testing approaches are focused on security.

## 1.4 Software Security Engineering

Software Security Engineering (SSE), proposes to leverage and extend the science behind SRE to engineer a desired degree of security into a software product. Security vulnerabilities and successful attacks are in many ways similar to *“classical”* faults and failures except for the presence of deliberate harmful intent and use of attack profiles. Therefore, SSE aims to shift the focus of some SRE practices and/or processes to security. Thus, the key to SSE is a good understanding of natural human traits such as:

- the ability of humans to make random mistakes regardless of other factors such as knowledge, time and other resources invested on them, or
- the human tendency to also make mistakes due to cultural, training, education and other biases, which accounts for the epistemic uncertainty in security problems.

The proportion of security problems that can be attributed to each of the above causes could provide useful information.

## 1.5 Approach

The goal of this work is to examine the statement that security problems are a subset of general class of software problems, and to see if SRE methodology, including operational and non-operational testing may be a way to characterize and improve software security characteristics. The approach is empirical, and is doing this exploratory work in the context open source software, specifically Fedora. There were three broad components integral to that:

1. Classifying security problems on the basis of the type of uncertainty involved in their introduction into the system.
2. Use of “Classical” Software Reliability Growth Models (SRGMs) to describe the process of discovery and removal of security problems during operational use of the software, and to attempt to estimate the number of residual problems in Fedora.
3. Attempt to construct a security-oriented non-operational testing profile that may help find the observed security categories sooner.

A review of relevant literature on different vulnerability/fault taxonomies was followed by a re-classification of the observed problems in terms of whether they appeared to be epistemic or aleatoric in nature.

Fedora was chosen for our studies because it is widely used, its problem reports and information about its usage are readily available, and because work on earlier releases of Fedora offered some interesting findings [43, 65]. In each of the four releases examined in detail, a sample of 60 security problems was selected at random for manual analyses. This is roughly 20-25% of the entire sample of security problems reported for a release. We consider this to be representative of the set of security problems reported for a single release. Problem report information was obtained from the Red Hat bug repository for Fedora [3] and related vulnerability databases such as the National Vulnerability Database(NVD). Due to time constraints, we have looked at data for only four (4) different versions of Fedora namely 15, 16, 17 and 18. The latest version of Fedora as of 25<sup>th</sup> April, 2014 is version 20 (Heisenberg) [26].

Classical SRE models were applied, using the CASRE tool [71], to the security data reported under normal operational usage of Fedora. Resulting residual fault estimates and the distribution of the vulnerabilities was used to construct non-operational (attack) profiles for Fedora 15, 16, 17 and 18. An iterative testing model incorporating non-operational testing in parallel with regular

operational testing is proposed and illustrated. The efficiency of this process in discovering security problems has not been verified and provides a possible direction for future work.

## 1.6 Outline

Chapter- 2 gives an overview on the genesis of vulnerabilities in software, the concepts of operational and non-operational profiles and reliability growth modeling. The chapter also discusses the difference between *epistemic* or knowledge driven uncertainty and *aleatory* variability or the natural randomness present in real physical process, and relating that to vulnerable states in software.

Chapter- 3 discusses the data sources that were used as a part of this thesis, which are primarily Red Hat Bugzilla[3] and the MITRE corporation's Common Weakness Enumeration(CWE)[50]. A classification scheme based on uncertainty(discussed in Chapter- 2) is presented. A detailed discussion on the analyses of security problems reported for open source Fedora versions 15, 16, 17 and 18 is presented.

In Chapter- 4, Software Reliability Growth Models(SRGMs) [107] are discussed along with their assumptions. The fits obtained using the CASRE tool [71] are discussed.

Chapter- 5 presents a proposal for a process change and discusses a hybrid approach to software testing based on a combination of operational and non-operational testing with a goal of improving the security reliability growth of the software.

Finally Chapter- 6 give a summary of the contributions of this thesis along with directions for future work.

## 2.1 Vulnerabilities

In [47] Bishop and Bailey use state transitions to describe authorized, unauthorized, vulnerable and compromised states in a computer. A computer system, as they describe, is composed of states that describe the current configuration of entities that make up the computer system. The system computes by applying state transitions which results in change of system state. All the states reachable from a given initial state using a set of state transitions fall into one of the classes of authorized and unauthorized states, while the state transitions themselves are labeled authorized or unauthorized in accordance with security policies of the specific system.

Thus we have:

1. *Vulnerable State*: An authorized state from which an unauthorized state can be reached using authorized state transitions.
2. *Compromised State*: The unauthorized state reached by authorized state transitions from a *vulnerable* state.
3. *Attack*: A sequence of authorized state transitions which end in a compromised state. Thus, by definition, an attack begins in a vulnerable state. In other words, an attack is a specific method to exploit a vulnerability.

With respect to the earlier discussion (Section 1.2), all states that are not authorized and are not reached by authorized transitions are error states. The IEEE glossary [88] defines a *mistake* (or error

in Section 1.2) as ‘a human action that produces and incorrect result’ and a *fault* as ‘an incorrect step, process or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner’. Therefore, faults are a manifestation of mistakes made by humans (architects, programmers, etc.). Vulnerable states in a system may be instances of such faults. A *failure* is the ‘inability of a system or component to perform its required function within the specified performance requirement’. Failures can be caused by hardware or software faults (defects), or by how-to-use errors. A fault (or defect, or bug) is a defective, missing, or extra instruction, or a set of related instructions, that is the cause of one or more actual or potential failures. Inherent faults are the faults that are associated with a software product as originally written, or modified. Faults that are introduced through fault correction, or design changes, form a separate class of modification faults. Failures can be caused by software faults, functional lacks in software, or user errors (for example, lacks in user’s knowledge). Thus system failure is analogous to security breach where a security breach is a deviation in the system behavior from the security requirements or defined security policies. Hence, failure in the security context could also be characterized as a set of state transitions resulting in a compromised state, as in the case of an attack.

## 2.2 Operational Profiles

An operational profile, is a quantitative characterization of how a system will be used[76]. It describes how the users employ a system. Profiles are used to determine the “norm” or the expected behavior. They represent the *baseline behavior* of the system. In the context of software reliability, this would include information about the system’s ability to operate successfully, without experiencing failures. In the security context the normal operational profile would describe what services are considered normal with respect to the security policies of the system and what is “normal” for each service. That is, the normal operational profile could include *authorized transitions* from authorized states to authorized states which may sometimes expose vulnerable states. In contrast, an attack profile would include transitions (both authorized as well as unauthorized) to unauthorized(compromised) states from vulnerable states.

An operational profile can be used to guide the testing activity and ensure that when the software product needs to be delivered, and testing subsequently stops, the more frequently used functions work reliably in the field. It helps in delivering reliable software within budget and schedule by guiding allocation of development resources to functions based on their use(primarily) in conjunction with other factor(s) such as criticality. Operational profiles are used to select test cases and direct development, testing and maintenance efforts towards the most frequently used or most risky components. In constructing an operational profile, as described in [76] one needs to focus on the “use” from a *progressively narrow* perspective from the customer profile to the more fine grained operations of the system, during which at each step we quantify the frequency of usage of each of

the elements.

### 2.2.1 Determining the Operational Profile

Musa outlines three major steps in determining the operational profile:

1. Divide the execution into runs: Operations are said to be associated with *runs*. A run is described as a segment of a program's execution time which is based on the accomplishment of a user oriented task. Identical runs form a *run type* where each run type is said to have an associate *input state*. The input state is what uniquely determines the set of instructions that a run executes, the operand values, all intermediate input and output values, thus defining a unique path of control flow for the run.
2. Identify the input space: The *input space* is a set of input states that can occur during an operation. Here Musa differentiates the *required* input space from the *designed* input space and further states that areas in the required input space that do not fall into the designed input space are areas that will contain input states with a higher chance of failure. To define the input-space the author recommends defining a *practically complete* list of all input variables, where practically incomplete comprises all variables except those that assume only one value with a very high probability. It is the input states combined with their *occurrence probabilities* that give us the input state profile for a system.
3. Partition input space: It is advised to limit the input state profiles to the order of several hundred or a maximum of several thousand elements as the cost of developing them increases almost in proportion with the number of elements. Clustering the run types into operations partitions the input space allowing lesser number of elements. In essence, part of the input space that is contained within an operation is termed as a *domain*. It is advised to group run types that share the same input variables into one operation, however this is subject to some discretion as discussed in the paper [76]. Partitioning of the input space is important because it provides the basis for sampling non-uniformly across the input-space. As one selects operations randomly based on the operational profile and subsequently selects input-states randomly from within the domain, it results in the selection of non-uniform random test cases that are similar to the operational profile of the system.

Test case selection is carried out by selecting operations in test according to their *occurrence probabilities*, then selecting the run categories and finally the specific run types. This kind of testing guided by the operational profile is efficient as it finds failures in the order of their frequency of occurrence.

Finally, the sampling of operations should be done with replacement to allow for re-selection of the same operation; but selection of run categories within operations must be made without

replacement which would allow the run types associated with the operation to differ in each selection, subsequently exhibiting different failure behaviors.

## 2.3 Reliability Models

$$R(\tau) = e^{-\lambda\tau} \quad (2.1)$$

As mentioned in Section 1.3 an important step in SRE is “Reliability Modeling and Measurement”. Reliability is probably the most important of the characteristics inherent in the concept of “software quality”. Software reliability characterizes the extent to which a software functions to meet its customer’s requirements and hence is a user-oriented representation of the quality of the software product. **Software Reliability** is defined as the *probability of failure-free operation of a computer program for a specified time in a specified environment* [79]. *Failure intensity* is an alternate way of expressing reliability. If the reliability of a system is 0.92 for 8 hours or time, then by Eq. 2.1 (where  $R$  is the *reliability*,  $\tau$  is *execution time* and  $\lambda$  is the *failure intensity or rate*), an equivalent statement is that the failure intensity is 0.01 failure/hour.

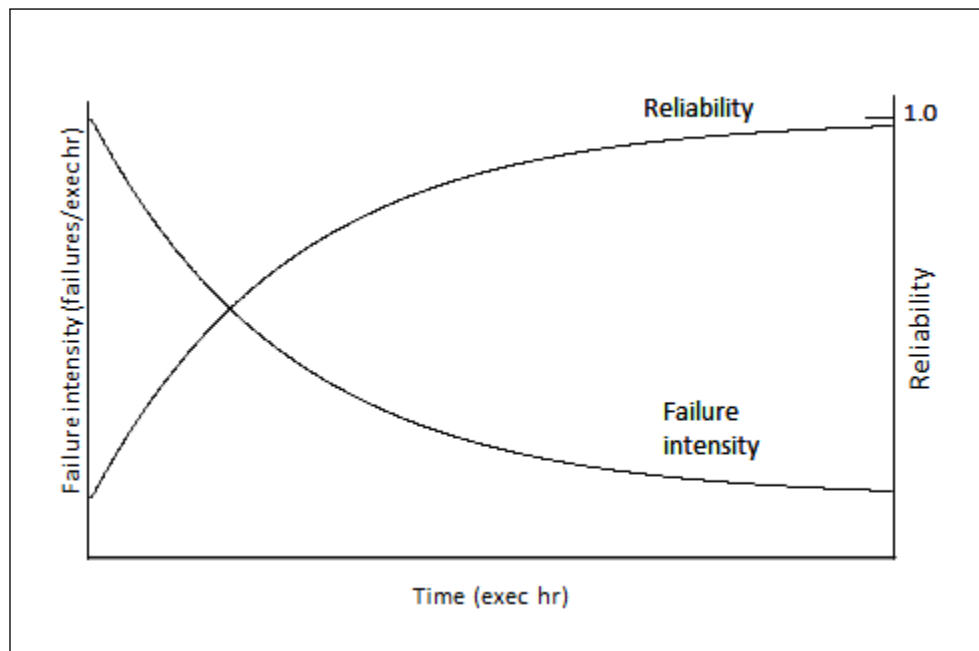


Figure 2.1: Reliability and Failure Intensity

The variation in failure intensity and reliability during the testing phase as faults are removed is depicted in Figure 2.1. As faults are removed, the failure intensity tends to drop while the reliability

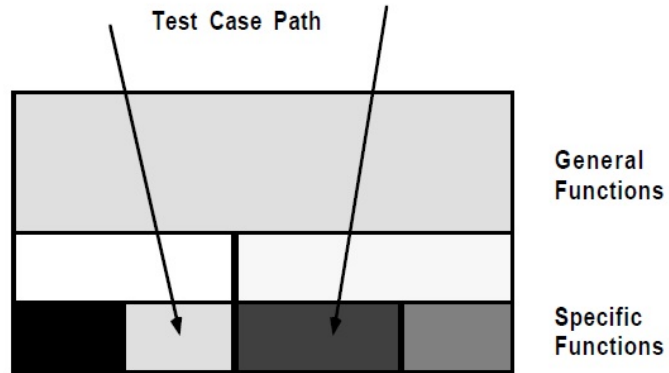
tends to improve. Three factors affect software reliability—fault introduction, fault removal and the environment. Fault introduction depends on the development process and practices. Fault removal depends on time (execution/calendar), operational profile and the quality or efficiency of the repair activity. Lastly, the environment depends directly on the operational profile of the software. Due to the probabilistic and are time dependent, software reliability models are formulated in terms of random processes. The main goal of a software reliability growth model is to be able to predict the failure behavior that will be experienced when the program becomes operational i.e. available on the field to be used. A software reliability model describes the general form of dependence of failure processes on the above mentioned factors. The associated uncertainty in determining the specific mathematical form of these models is expressed in terms of the confidence intervals on the parameter values. As discussed in [79] software reliability measures are valuable to a software engineer, manager or user in the following ways:

1. *System Engineering*: Making trade-offs among components w.r.t. reliability quantitatively.
2. *Technology*: Evaluating software engineering technologies (both new and old) quantitatively.
3. *Development Status*: Objective reliability measures for e.g. failure intensity derived from test data can be used to determine the development status during the testing phases.
4. *Managing Change*: Monitor operational performance of software and control addition of new features, design changes while adhering to reliability objective.
5. *Quality*: Quantitative measures of the quality of the software product and the factors influencing and affected by it provides new insights on the development processes which can be leveraged in making more informed decisions.

An analogous list can be developed in the context of software security engineering. We discuss Software Reliability Engineering Growth Models (SREGMs) we used in this study in Chapter- 4.

## 2.4 Non-Operational Profiles

There are two extreme cases of non-operational software use practices: “layer” and “functional groups” models [103]. Most SRE models assume problem detection based on the operational profiles of the system. This assumption is violated during, for example unit and integration testing phases, and during cyber-attacks. Therefore, the use of traditional SRE models in such situations may be difficult and possibly also misleading unless special precautions are taken. Figure 2.2 illustrates the layered model. The principle is that of sampling with replacement where an input activates a hierarchy of functions. As the result more basic functionalities are exercised more often due to



Execution of a "function" involves synthesis of general and specific functions provided by the program

Figure 2.2: The *Sandwich* or *Layered* testing model [103]

their repeated inclusion in multiple test or attack case paths. In such cases, classical failure intensity decay models may be accurate in measuring the effectiveness of the testing or attacking process.

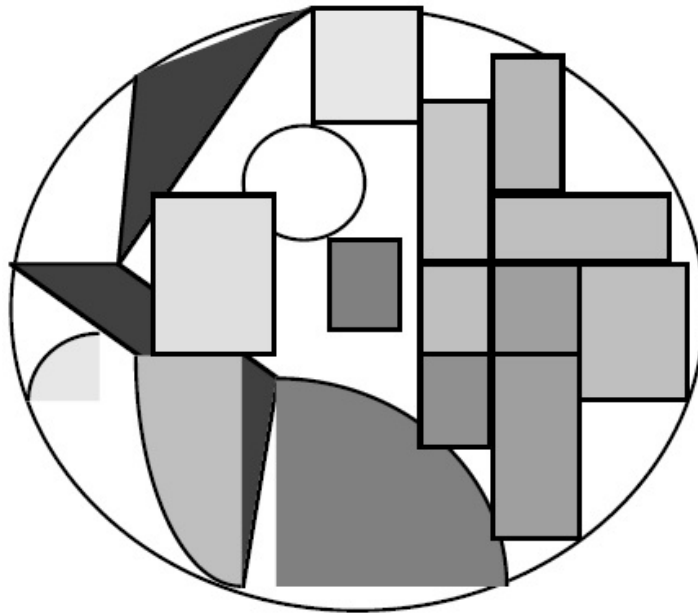


Figure 2.3: The *Functional Groups* or *Disjoint* Testing Model [103]

The other extreme is the functional groups model shown in Figure 2.3 in which the components or features of the program are decoupled or disjointed to an extent where they are completely independent of each other. Each test or attack case in such a model exercises a completely disjoint set of functions and the failure intensity may not follow the “*classical*” decay profile, but may exhibit modes instead. The differential equations that describe such a model give a Weibull failure intensity in the most general case, a special case of which being the Rayleigh model [103]. In the extreme case, this model results in sampling without replacement, and is more akin to coverage-based models. In those situations the process is describable using Hypergeometric distribution [102, 90].

It can be shown[103, 102, 90] that fault removal growth offered by a non-operational profile, can be successfully modeled. Furthermore, it is also possible to establish the point of diminishing returns. This may be helpful in decisions such as when to change strategy, or what is “good enough” [94].

### **2.4.1 Attack Profiles**

While a “regular” operational profile does take into account the environmental conditions that the system operates in, it assumes regular usage alone. Hence, testing for security based solely on operational profile is not sufficient to catch attackers who are known to limit the number of resources that they are ready to spend on trying to compromise any system. Attack sequences on the other hand, will be focused on real or imagined vulnerabilities, often applying sampling without replacement i.e. attack cases are not repeated [91]. A question of interest is how often are security problems found in non-attack situations, and can classical SRE models be used in those situations, and if so, what can one say about the security quality of the product based on that? Previous work by Anbalagan [40, 41, 42, 43] and Da Young [64] seem to indicate this is possible.

## **2.5 Epistemic Versus Aleatory**

As already mentioned, there are two types of errors humans make we are interested in, in the context of injection of security problems - aleatory and epistemic. Aleatory variability may be defined as “*the natural randomness in a process*” [39]. It is also the statistic uncertainty which is representative of unknowns that differ each time we run the same experiment [36]. Epistemic uncertainty is the scientific uncertainty in the model of the process. It is due to limited data and knowledge [39]. It is the systematic uncertainty which is due to things we could in principle know but do not practice [36]. It is a reducible uncertainty in the sense that improved understanding by means of thorough research, or having access to more relevant data can reduce the epistemic uncertainty to the natural aleatory variability of the entity being analyzed.

Abrahamson [39] distinguishes these two concepts as an example of an unknown die where,

based on 4 rolls of the die and its corresponding outcomes, one could develop different models of the die, some purely based on theory of a die and some including empirical information. For example, if a die is loaded our experimental data may indicate different than expected (uniform) frequency at which different faces come up. The uncertainty that gives rise to different models represents the epistemic constituent. With more rolls of the die, i.e., more data, we may be able to determine whether the die is biased (loaded) and which models fit the data thus reducing our epistemic uncertainty. However, the aleatory variability does not become zero. In this case, the outcome of the roll of a die constitutes the aleatory component, i.e., if there are six faces on a die, the number that comes up should always range between 1 and 6.

In the context of software security engineering interests, we note that epistemic errors, (such as errors due to lack of knowledge, poor software practices, or incorrect specifications) probably can be avoided through better software and security engineering processes, more robust algorithms, additional training, and so on. On the other hand, aleatory or random errors (e.g., typos, errors due to stress, lack of sleep, distraction, or accidents) are hard to avoid, they need to be eliminated after the fact (but before release of the product) through appropriate verification, validation and testing.

### **2.5.1 Uncertainty in Software Security Engineering**

Lee et al. in their study [65] explain how current software practices appear to detect and remove security vulnerabilities more by luck which constitutes aleatory variability in the process as opposed to employing knowledge driven i.e. epistemic methods. Our interest is in analyzing security problems to determine if their cause is due to an aleatory event or due to an epistemic event. The good part about epistemic problems is that they can be detected, avoided or tolerated using *reactive* defense engineering strategies based on known attack vectors, vulnerability categories, etc. In the case of aleatoric security problems, current practices are proven insufficient in detecting them as they do not conform to previously known patterns, hence ZERO DAY exploits. As mentioned in [84] being prepared for the unknown constitutes the *dynamic* component of security which is an ongoing battle between the continuously evolving teams of attackers and defenders. Only strong scientific analyses will help us in distinguishing one from the other and reduce the uncertainty in the security reliability of the system.

## **2.6 Classifying Security Problems**

*The first step in wisdom is to know the things themselves; this notion consists in having a true idea of the objects; objects are distinguished and known by classifying them methodically and giving them appropriate names. Therefore, classification and name-giving will be the foundation of our science.*

Classification of an observable fact enables organized study of the fact. In particular, a taxonomy of security problems based on the uncertainty factor that we described in Section 2.5 above will help us understand what kind of counter-measures may be most effective in preventing, finding and mitigating such problems. For example, during construction of non-operational testing aimed at security problems. In the modern resource-constrained development environments, having focus is a key to minimizing resource wastage and curtailing costs.

The RISOS (Research in Secure Operating Systems) project [37], of the mid 1970s, presented case studies of operating system software in an attempt to recognize the basis of security problems. It was among the first pieces of work that aimed at classifying or categorizing security problems exhibited by operating systems. They note that the study of real instances is superior to theoretical penetration exercises as they are recurrent, taking place in-situ and the most importantly, "*embody the rational as well as unpredictable human behavior under natural stress*". In our case the "real" data come from Bugzilla, NVD, CWE and other vulnerability repositories. That is, all data part of this thesis have been obtained from The Common Weakness Enumeration(CWE), the RedHat Bugzilla bug repository, the Common Vulnerabilities and Exposures(CVE), the National Vulnerability Database(NVD) and the Open Source Vulnerability Database(OSVDB). These are discussed in detail in Chapter- 3.

Landwehr in his report[63] describes how security is mostly a "penetrate and patch" process; finding computer program flaws is an almost unending activity as more flaws always seem to appear. This experience has motivated researchers to seek better ways of building systems that would be inherently secure. This in fact motivates to organize security flaws into the categories that allow differentiation between random ones that can perhaps be removed using elimination processes, and epistemic ones which, in theory, are avoidable. This philosophy is at the core of the current study. Landwehr's taxonomy focuses on 3 main aspects of a security flaw: its genesis, time of introduction into the system and its location. The benefit is that designing tools to detect security flaws from such specific descriptions becomes easier. Since our initial goal was to identify whether most field security problems are subsets of problems that should have been avoided, we needed a higher level of abstraction in the taxonomy and this scheme of classification was not fulfilling the required purpose. However, we cannot discount its application in the future in order to build tools aimed at eliminating these subset of problems from the software.

Du and Mathur in [56] critique to the schemes of classifying a security problem into a single category. They believe that placing an error into one single category which closely matches most of its features will seek abstraction at the risk of throwing away essential information about the problem. For example, if a testing technique  $T$  has 95% efficiency in uncovering security problems, but most of these problems may have an attribute  $E$  which is abstracted by the classification which consists of categories  $A$ ,  $B$ ,  $C$  and  $D$  as they seemed to implicitly incorporate  $E$ . Assigning errors

to one of *A* to *D* in this case might bias estimates of the testing efficiency of a particular strategy and the authors suggest keeping more information in categories to avoid such biases. They thus propose an approach based on the *Cause —Impact —Fix* sequence of operations to capture all pieces of information that concern a security error. They make use of Landwehr's scheme [63] to classify problems by genesis by considering the *inadvertent* errors. The primary goal of their paper was to be able to avail of such a classification to evaluate effectiveness of different testing strategies in uncovering security errors and required eliminating factors that may bias the results of such an evaluation.

Classifying security problems will help us in constructing our attack test cases to include tests for problems that are not represented in the normal operational test profile of our system. In short classification describes what we should be looking for. But this information alone is insufficient in the construction of a non-operational test suite. We also need information on how many test cases we may additionally require to target these types of security problems. This information can be provided to us from classical Software Reliability Growth Models(SRGM) as described in Chapter- 4.

## **2.7 Approaches to Non-Operational Testing**

As discussed in [90], in classical testing approaches, learning is said to occur if testers dynamically improve the efficiency of their testing as they progress through the testing phase. But, given the constrained environments in which software is developed today, testing activity is often analogous to *sampling without replacement*. In each new phase, the primary and sometimes the only objective is to *cover* previously uncovered functionalities. This unguided constrained testing, will yield software of poorer quality than when testing is based on operational profiles which represent sampling with replacement. However, sampling with replacement may place unreasonable demands on budgets and schedules and hence in order to achieve better quality software, one must conduct a guided constrained testing process, where the testing efficiency tends to improve in consecutive phases.

This chapter, presents a discussion of the data used in conducting the empirical analyses. Section 3.1 gives an overview of the repositories. Section 3.2 and Section 3.3 present a discussion of the proposed security problem classification scheme which is exemplified using cases from Fedora software in Section 3.4. Sections 3.5 and 3.6 discuss the results of our analyses.

## **3.1 Data Sources**

The Common Weakness Enumeration (CWE) [50], Red Hat Bugzilla [3], National Vulnerability Database (NVD)[27], Common Vulnerabilities and Exposures (CVE) [15] and the Open Source Vulnerability Database(OSVDB)[29] have been the primary data sources in this study. We briefly discuss the purpose and information provided by each of these repositories in the following sections.

### **3.1.1 Common Weakness Enumeration (CWE)**

The Common Weakness Enumeration (CWE) is a project sponsored by the National Cyber Security Division of the US Department of Homeland Security to classify security problems. It assigns a unique number to weakness types such as buffer overflows or cross-site scripting bugs. The audience for CWE comprises of both the development community as well as the community of security practitioners. It is a formal list or dictionary of software weaknesses(problems) that can occur in software's architecture, design, code or implementation that can lead to exploitable security vulnerabilities. Its main goal is to stop vulnerabilities at the source by educating software designers,

architects, developers and testers on how to avoid the most common mistakes before delivering the final software.

The CWE/SANS Top 25 [50], a collaborative initiative between the CWE, MITRE and the SANS institute is a collection of 25 most common and dangerous software security problems that affect the software industry which can result in loss of information from or access to the application altogether. These security problems are often easy to find and easy to exploit. The top 25 list enables focus on issues plaguing the industry during present times. Yet, the issues persist 10 or more years after the list has first appeared. A glaring example is the recent heartbleed vulnerability [97, 18] and exploit [34] in OpenSSL[30] that turned out to be lack of proper bounds checking (buffer overflow) - a textbook example of how not to write code.

### **3.1.2 Bugzilla for RedHat Fedora**

Empirical data discussed in this thesis come from data found in Red Hat Bugzilla. Red Hat Bugzilla is the Red Hat bug-tracking system and is used to submit and review problems that have been found in Red Hat distributions [3]. In this section, we describe the steps used to collect data for Red Hat sponsored Fedora from the Bugzilla database.

#### **Relationship between Fedora and Red Hat Enterprise Linux**

Fedora is sponsored by Red Hat, which invests in Fedora to encourage collaboration and incubate new innovative free software technologies i.e. Fedora is used as an open research and development lab that is shared with the developer community to test new technologies. The goals of Fedora and Red Hat Enterprise Linux(RHEL) are fundamentally opposed in that Fedora strives to achieve innovation(and change) while RHEL is intended as a more stable platform that extends support to its customers over a long period of time. This is also one of the primary reasons why RHEL is charged while Fedora is the free version of Red Hat's open source Linux platforms. Fedora is upstream for RHEL as shown in Figure 3.1 which is a commercial enterprise operating system that has its own set of test phases including alpha and beta releases which are separate and distinct from Fedora development [100].

#### **Steps used for Generating Reports in Bugzilla**

1. Go to `https://bugzilla.redhat.com`.
2. Under the *Most Common Actions* Panel on the top left click on *Summary Reports and Charts*.
3. Under the *Current State* section, select *Tabular Reports*.
4. (a) Select *Version* from the drop-down menu for *Horizontal Axis*.

Table 3.1: 2011 CWE/SANS Top 25 Most Dangerous Software Errors

Rank	ID	Name
[1]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command (SQL Injection)
[2]	CWE-78	Improper Neutralization of Special Elements used in an OS Command (OS Command Injection)
[3]	CWE-120	Buffer Copy without Checking Size of Input (Classic Buffer Overflow)
[4]	CWE-79	Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)
[5]	CWE-306	Missing Authentication for Critical functionality
[6]	CWE-862	Missing Authorization
[7]	CWE-798	Use of Hard-coded Credentials
[8]	CWE-311	Missing Encryption of Sensitive Data
[9]	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	CWE-250	Execution with Unnecessary Privileges
[12]	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory (Path Traversal)
[14]	CWE-494	Download of Code Without Integrity Check
[15]	CWE-863	Incorrect Authorization
[16]	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	CWE-732	Incorrect Permission Assignment for Critical Resource
[18]	CWE-676	Use of Potentially Dangerous Function
[19]	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[20]	CWE-131	Incorrect Calculation of Buffer Size
[21]	CWE-307	Improper Restriction of Excessive Authentication Attempts
[22]	CWE-601	URL Redirection to Untrusted Site (Open Redirect)
[23]	CWE-134	Uncontrolled Format String
[24]	CWE-190	Integer Overflow or Wraparound
[25]	CWE-759	Use of a One-Way Hash without a Salt

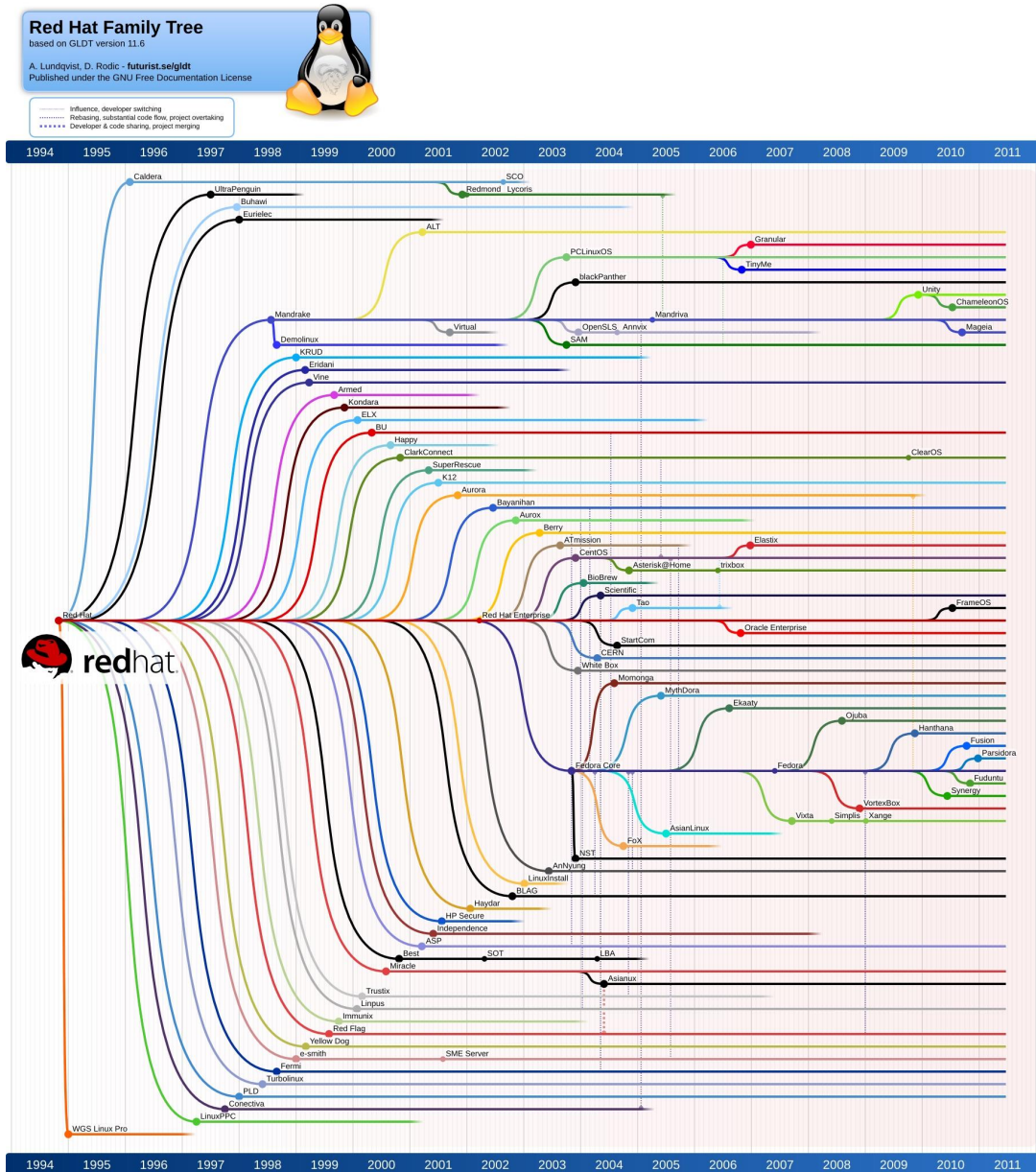


Figure 3.1: The Red Hat Family Tree [4]

- (b) Select *Classification* from the drop-down menu for *Vertical Axis*.
- (c) Select *Fedora* from the list for *Classification*.
- (d) Select *Fedora* from the list for *Product*.
- (e) Click the *Refresh Components/Versions/Releases/Milestones* button.
- (f) Select *CLOSED* from the list for *Status*.

- (g) Select *CURRENTRELEASE*, *RAWHIDE*, *ERRATA*, *UPSTREAM*, *NEXTRELEASE* from the list for *Resolution*.
  - (h) Under *Detailed Bug Information*, select *Contains any of the words* from the drop down menu for *Keywords* and then select *Security* and *SecurityTracking* for the next field.
  - (i) Select the version number(s) from the list for *Version*.
  - (j) Click on the *Generate Report* button.
5. Click on the hyperlinked number to get the entire report of security bugs filed for one version of Fedora

### **Rationale Behind Selection**

Two parameters in the above report generation method are of prime importance in current work—*Status* and *Resolution* of bugs. In order to ensure the validity of the assumption that each security problem is unique and independent of the others and that we do not generate a new problem by fixing an existing one, we need to carefully select only those security bugs reported in Fedora that have been *closed* and resolved in either the current or the next release or have been fixed upstream [64]. The selection of the specific criteria mentioned in point #4 (f) and (g) above, were based on the life cycle of the bugs reported against Fedora [31]

### **3.1.3 CVE, NVD and OSVDB**

The Common Vulnerabilities and Exposures (CVE) is a collection of publicly known information security vulnerabilities and exposures. CVE is not a vulnerability database. CVE does not contain information such as risk, impact, fix information, or detailed technical information. CVE only contains the standard identifier number with status indicator, a brief description, and references to related vulnerability reports and advisories. The goal of CVE is to make it easier to share data across separate vulnerability capabilities (tools, repositories, and services) with this “common enumeration”[15] The National Vulnerability Database (NVD) is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP). This data enables automation of vulnerability management, security measurement, and compliance. NVD includes databases of security checklists, security related software flaws, mis-configurations, product names, and impact metrics[27]. While CVE and NVD are sponsored by the Department of Homeland Security’s National Cyber Security Division, the Open Source Vulnerability Database(OSVDB), is an independent and open sourced web-based vulnerability database created for the security community. The goal of the project is to provide accurate, detailed, current, and unbiased technical information on security vulnerabilities [29].

Similar to Anbalagan studies [43] security problems used in the current study are unique CVE numbers which are cross-referenced in the NVD as well as the OSVDB. The findings in this study

are based on information retrieved from these three main vulnerability databases and the external references cited therein.

## 3.2 Classifying the CWE Top 25

In this section, we present an analysis of the Top 25 problems according to the uncertainty categories (aleatoric and epistemic) described in Chapter- 2 Section 2.5. In this sub-section we first provide, as examples, analyses, with respect to uncertainty categories, of several top 25 problems to illustrate. This is followed by summary of the findings for all top 25 errors.

### # 1 CWE - 89 Improper Neutralization of Special Elements used in an SQL Command (SQL Injection)

Injection attacks occur when unvalidated input is embedded in an instruction stream and cannot be distinguished from valid instructions. When a language having a parser or an interpreter and the input stream can be confused for instructions in the language or if it interferes with the way the language is applied, in such cases the language is vulnerable to injection attacks. SQL injection takes advantage of insecure code resident on systems that are connected to the Internet to pass commands directly to the underlying database systems resulting in considerable damage. In the case of SQLi, the software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component [23]

According to the IBM X-Force 2013 Mid-Year Trend and Risk Report SQLi was the most common breach paradigm. As McDonald mentions in his article [73] SQL injections are neither a “dark art” nor are they new. They have been known to exist for the past 14 years [44]. They represent a serious threat to any database-driven site. The means to execute such an attack are not out of reach of a lay man. There are plenty of resources available online to carry out SQL injection attacks without the use of sophisticated tools.

SQL injections become a reality when the programmer ignores the first principle of security —*Input is evil!*. The mitigation strategies are well known and simple. It is an input validation problem and the first step as a programmer would be to accept only known good input —reject input that does not conform or transform it into something that does. Thus, use of white-list of acceptable inputs is preferred over using black-lists. Use of libraries that enforce structure on the application for e.g. frameworks like Hibernate can provide significant protection against SQLi if used properly. Processing SQL queries using stored procedures and prepared statements as opposed to constructing queries in an ad-hoc manner is a basic practice that must be followed.

Presence of SQLi bugs in code indicates an absence of established organizational programming standards. Further, this error in most cases relates to lack of knowledge or poor practice in the part of the programmer/organization and contributes to epistemic bugs. One cannot however discount the presence of a slight aleatory variability in that an unescaped character could be a genuine typo. But for most part, SQLi bugs are epistemic in nature.

### **# 2 CWE - 78 Improper Neutralization of Special Elements used in an OS Command (OS Command Injection)**

The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component [22]. Notice the similarity in the definitions for SQLi and OS command injection? Once, again this is a case of insufficient input validation. CWE describes two variants of this bug:

1. The application uses user supplied input as arguments to execute an intended fixed program under its own control. The attacker may use command separate to execute their own program after the application program has completed execution. In this case, clearly the programmer is trusting the user input and must ideally validate it.
2. The application makes use of user supplied input in deciding which program and what set of commands to execute and hands down this information to the operating system. In such a situation the attacker may execute arbitrary commands if under his/her control. Here the programmer intended to hide the command from the attacker but has not air-gapped all paths sufficiently.

Mitigations for the CWE#2 include those for SQLi as at their lowest level they are input validation bugs and additionally ensure proper escaping and encoding of special characters in both the input as well as output sequences. Further, the programmer must adopt the design principle of running applications with the *principle of least privilege* which grants permissions to applications on a “need to know” basis. The input validation and encoding mechanisms provide *defense - in - depth* but clear demarcations of the trust spheres makes it one step difficult for the attacker to get into your operating system and take control of everything.

Similar to #1, OS command injection also falls into the epistemic variety of software bugs, which continue to exist due to ignorance or sloppiness on the part of programmers, as well as poor design decisions of architects and security practitioners.

### **# 3 CWE - 120 Buffer Copy without Checking Size of Input (Classic Buffer Overflow)**

The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

The simplest type of error, and the most common cause of buffer overflows is the “classic” case in which the program copies the buffer without restricting how much is copied. The existence of a classic overflow strongly suggests that the programmer is not considering even the most basic of security protections [19].

Existence of buffer overflows can be partially attributed to the lack of input validation. The least the programmer could and should do is consider the length of the input. Mitigations for the CWE#3 include input validation, usage of static analysis tools that detect these kind of bugs, usage of frameworks, languages or libraries that do not allow or make it difficult for this weakness to occur.

The “classic” buffer overflow is predominantly an epistemic security problem as its existence only signifies negligence, poor processes or standards or lack of security knowledge.

### # 20 CWE - 131 Incorrect Calculation of Buffer Size

In unmanaged languages such as C, memory management is at the mercy of the programmer. This increases the possibility of introducing problems in the code. While CWE # 3 is lapse on the part of the programmer, often despite having conditions for bounds check of the input parameters in place, a program may be faced with a buffer overflow situation. This happens when the size of the buffer is calculated incorrectly which can happen for a variety of different reasons, some of which include:

- simple math errors
- not accounting for size differences when transforming one input to another format
- not accounting for the sentinel in calculations for total memory to be allocated while allocating memory that uses sentinels to mark the end of a data structure

For example, consider the code snippet below [20]:

```
1  int i;
2  unsigned int numWidgets;
3  Widget **WidgetList;
4  numWidgets = GetUntrustedSizeValue();
5  if ((numWidgets == 0) || (numWidgets > MAX_NUM_WIDGETS))
6  {
7      ExitError("Incorrect number of widgets requested!");
8  }
9  WidgetList = (Widget **)malloc(numWidgets * sizeof(Widget *));
10 printf("WidgetList ptr = %p\n", WidgetList);
11 for(i = 0; i < numWidgets; i++)
12 {
13     WidgetList[i] = InitializeWidget();
14 }
```

```
15  WidgetList[numWidgets] = NULL;
16  showWidgets(WidgetList);
```

The intention is to allocate memory for a maximum number of *widgets*. It takes as input the number of widgets from the user and checks to ensure that the request is not for too less or too many. Next, it initializes the elements of the array `WidgetList` by calling the function `InitializeWidget()`. Since the number can vary between 1 and `MAX_NUM_WIDGETS`, the `NULL` character is used to mark the end of the list.

There is an off-by-one error in this piece of code. It allocates just enough memory for the total number of widgets, forgetting about the `NULL` pointer. If a request is made for the `MAX_NUM_Widget` number of widgets, allocation of the `NULL` pointer at the end might corrupt other data in memory (depending on the environment).

Although, this indicates an oversight on the programmer's part (which is epistemic), it could also be a genuine random mistake committed due to several reasons such as high stress environments, lack of sleep, distractions etc. Thus, there is a partial aleatory variability associated with this weakness in addition to the epistemic component.

Mitigations for this problem typically exist during the implementation phases. Replacing unbounded copy functions with analogous functions that support length arguments, such as `strcpy` with `strncpy` or creating these if they are not available, identifying and resolving any inconsistencies between the size field and the actual size of the data when processing structured incoming data containing a size field followed by raw data, etc. These strategies however may only let us control problems that arise due to poor practices. The ones that are accidental may still persist.

### 3.2.1 Summary - CWE Top 25

All security problems give in the CWE/SANS Top 25 list in table 3.1 are primarily epistemic in nature. This means that those problems can be avoided consciously by practices such as more security training, secure development and similar. The CWE itself provides a comprehensive list of mitigation strategies right from the architecture and design stages to the implementation and operation phase of a software. Therefore in our classification scheme Figure 3.2 discussed in Section 3.3, the CWE/SANS Top 25 feature as a subcategory under epistemic issues.

## 3.3 Classification Scheme

We randomly selected 60 security field problem reports for each of the Fedora software versions 15, 16, 17 and 18.

It turns out that most of the problems encountered by the 40th week after release in a Fedora release were epistemic in nature but did not necessarily fall into the top 25 categories [99]. Instead,

they were instances of more process oriented incomplete analysis and design issues. One could argue that issues listed on the CWE top 25 may themselves be cases of incomplete analysis or design issues, however, we decided to keep the top 25 intact to serve as a good reference point in estimating percentage of security problems that can be eliminated using an automated test generation strategy. Security problems resulting from misinterpreted or incorrect requirements or functional specifications were classified as problems of incomplete analysis. Problems that arose due to violation of secure software development design principles were classified as design flaws. The latter two categories of security problems can be mitigated by incorporating security into every decision in the software development life-cycle (for example, enumerating the threats posed by a functional requirement, the security implications of a design choice, etc.). Examples of security problems that classified in these subcategories shall be discussed in the next section.

Security problems caused due to typographical errors, state or environment changes, incorrect fixes, inherent weaknesses in standards or due to complexity of the problem being handled were classified as aleatoric problems. As discussed in [105] when building real world models, one would like to reduce the epistemic uncertainty to aleatoric uncertainty but it may be the exact opposite in the case of security. It is easier to identify epistemic (knowledge based) biases and handle them as equivalency classes but it may be much harder to target aleatoric security issues.

In the following sections we present examples of security problems reported during normal operational use of Fedora releases 15, 16, 17 and 18 classified under each of these categories. For some security problem reports, it was hard to judge whether their cause was an epistemic or aleatoric event and hence, we classified them as “borderline” issues. Lastly, for some problem reports, due to lack of sufficient information we were unable to classify them into any one particular category and left them as “unclassified”.

## **3.4 Examples in Fedora**

### **3.4.1 Epistemic Security Problems**

#### **CWE/SANS Top 25**

#### **Fedora v15 Bug Bug 742654 - CVE-2011-3869 puppet: k5login can overwrite arbitrary files as root**

“Puppet is a declarative, model-based approach to IT automation, helping you manage infrastructure throughout its life-cycle, from provisioning and configuration to orchestration and reporting. Using Puppet, you can easily automate repetitive tasks, quickly deploy critical applications, and proactively manage change, scaling from 10s of servers to 1000s, on-premise or in the cloud” [11].

The k5login type is typically used to manage a file in the home directory of a user; the explicit purpose of the files is to allow access to other users. The code responsible for this issue is presented

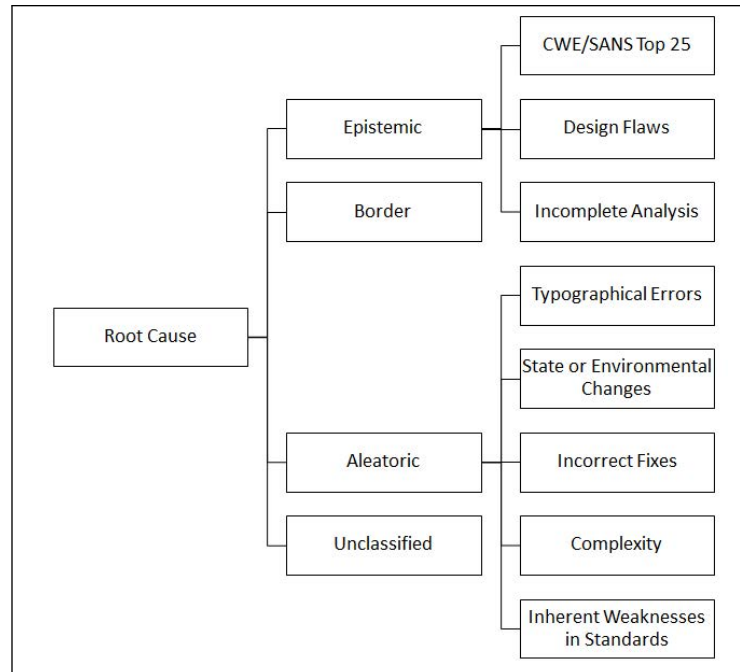


Figure 3.2: The Classification Scheme

below [86]:

```
1 File.open(@resource[:name], "w") { |f| f.puts value.join("\n") }
```

The issue was that if a user's `.k5login` file was a symlink, Puppet could overwrite the link's target when managing that user's login file with the `k5login` resource type. This could allow local privilege escalation by linking a user's `.k5login` file to root's `.k5login` file, which could result in escalation to root the next time Puppet ran.

The problem here is that puppet writes to the target file directly, without doing anything to secure it. Insecure file operations give an attacker the ability to read confidential information, perform a denial of service attack, take control of an application, or even take control of the entire system. The patch for the above code snippet is as follows [86]:

```
1 Puppet::Util.secure_open(@resource[:name], "w") do |f|
2 f.puts value.join("\n")
```

The `secure_open()` function is a utility function that opens files in a secure fashion checking for Time-Of-Check-To-Use (TOCTOU) and symbolic link attacks. Thus, we consider this to be a case of CWE #15 which is Incorrect Authorization (while opening the file) which leads to CWE #11 i.e. Execution With Unnecessary Privileges. Therefore this security problem was classified as an Epistemic "Top 25" security problem.

### **Fedora v15 Bug 752703 - CVE-2011-4128 gnutls: possible DoS due to buffer overflow**

GnuTLS is a secure communications library implementing the SSL, TLS and DTLS protocols and technologies around them. It provides a simple API in C to access the secure communications protocols [33]. This bug was reported to flag a possible Denial-of-Service vulnerability that existed in the way the function responsible for session resumption handled an input argument [95].

The MITRE CVE dictionary describes this issue as [6]:

Buffer overflow in the `gnutls_session_get_data` function in `lib/gnutls_session.c` in GnuTLS 2.12.x before 2.12.14 and 3.x before 3.0.7, when used on a client that performs non-standard session resumption, allows remote TLS servers to cause a denial of service (application crash) via a large SessionTicket.

The lines of code responsible for this security problem are given below [72]:

```
1 *session_data_size = psession.size;
2 if (psession.size > *session_data_size)
3 {
4   ret = GNUTLS_E_SHORT_MEMORY_BUFFER;
5   goto error;
6 }
7 if (session_data != NULL)
8   memcpy (session_data, psession.data, psession.size);
```

As we can see, the check for possible buffer overflow in line 2 is made after an assignment is made in line 1. Ideally, line 1 should be executed only after line 6. Thus, this piece of code results in a “classic” buffer overflow i.e. CWE #3 vulnerability. This application layer security problem in Fedora version 15 was therefore classified as an Epistemic problem of the Top 25 flavor.

### **Fedora v16 Bug 827365 - CVE-2012-2661 rubygem-activerecord: SQL injection when processing nested query paramaters**

The MITRE CVE dictionary describes this issue as:[16]

The Active Record component in Ruby on Rails 3.0.x before 3.0.13, 3.1.x before 3.1.5, and 3.2.x before 3.2.4 does not properly implement the passing of request data to a `where` method in an ActiveRecord class, which allows remote attackers to conduct certain SQL injection attacks via nested query parameters that leverage unintended recursion, a related issue to CVE-2012-2695.

The vulnerable code directly passes the request parameters to the WHERE clause of a query i.e. constructs the query on the fly without sanitizing the input thus opening doors to the CWE #1 or a SQL injection attack [32]. *intend to add one example for each of the top 25*

## **Incomplete Analysis**

### **Fedora v16 Bug 883063 - CVE-2012-5621 ekiga: DoS (crash) after receiving call from other party with not UTF-8 valid name**

Ekiga is a VoIP, IP Telephony, and Video Conferencing application that allows you to make audio and video calls to remote users with SIP or H.323 compatible hardware and software. It supports many audio and video codecs and all modern VoIP features for both SIP and H.323 [25].

A denial of service flaw was found in the way Ekiga processed information from certain OPAL connections (UTF-8 strings were not verified for validity prior showing them) [66, 8]. A remote attacker (other party with a not UTF-8 valid name) could use this flaw to cause ekiga executable crash. While the fix for this issue available at [55] does seem like an input validation vulnerability, its origins seem to be at the requirements level as the function should have been able to handle inputs of the remote machine participating in the call when they were not UTF-8 encoded. It is a case on incomplete analysis of the specifications for the expected input of the function which could be due to a communication error or because the requirements were not analyzed sufficiently. Therefore this application layer security problem in Fedora 16 has been classified as an epistemic issue resulting from incomplete analysis of the requirements.

### **Fedora v15 Bug 712774 - CVE-2011-2203 kernel: hfs\_find\_init() sb->ext\_tree NULL pointer dereference**

The MITRE CVE dictionary describes this issue as:

The `hfs_find_init` function in the Linux kernel 2.6 allows local users to cause a denial of service (NULL pointer dereference and Oops) by mounting an HFS file system with a malformed MDB extent record.

This security problem reported in Fedora 15 was due to a NULL pointer dereference in the Linux kernel's Hierarchical File System (HFS). If a specially crafted HFS file system, which had a corrupted Master Directory Block (MDB) extent record was mounted on a machine, it could result in a Denial-of-Services on the machine. The context of this bug report suggests that there was a misunderstanding regarding the operations performed by another function [101]. Due to lack of analysis of the functions being used, a wrong assumption was made which resulted in this security problem. Therefore, this problem in the Fedora 15 core substrate was classified as an instance of incomplete analysis.

## Design Flaws

### **Fedora v15 Bug 773025 - CVE-2012-0035 emacs: CEDET global-edo-mode file loading vulnerability**

GNU Emacs is a free, portable, extensible text editor. CEDET is a Collection of Emacs Development Environment Tools written with the end goal of creating an advanced development environment in Emacs. CEDET is a project which implements several advanced features developers have come to expect from an Editor [68]. The MITRE CVE dictionary describes this issue as:

Untrusted search path vulnerability in EDE in CEDET before 1.0.1, as used in GNU Emacs before 23.4 and other products, allows local users to gain privileges via a crafted Lisp expression in a Project.ede file in the directory, or a parent directory, of an opened file [51].

As per [83] EDE can store various information about a project, such as how to build the project, in a file named Project.ede in the project directory tree. When the minor mode 'global-edo-mode' is enabled, visiting a file causes Emacs to look for Project.ede in the file's directory or one of its parent directories. If Project.ede is present, Emacs automatically reads and evaluates the first Lisp expression in it.

This design, violates the principle of compartmentalization by not isolating components from each other based on trust boundaries, exposing EDE users to the danger of loading malicious code from one file (Project.ede), simply by visiting another file in the same directory tree.

The patch written to fix this security problem prevents EDE from loading Project.ede files, except in directories explicitly designated as "safe" by the user via the new list variable 'ede-project-directories'.

### **Fedora v17 bug 841671 - CVE-2012-3378 at-spi2-atk: insecure temporary file handling [54]**

The *Assistive Technologies Service Provider Interface* (AT-SPI) in the *Accessibility ToolKit* is a toolkit-neutral way of providing accessibility facilities in applications [13]. The at-spi2-atk supports "accessibility" over Linux platforms by acting as the "glue" between applications and assistive technology programs which enable people with disabilities to browse text over the Internet, read contents, etc. The package makes it easier for developers to support *assistive technologies*(AT) by making calls to native accessibility APIs [82].

The MITRE CVE dictionary describes this issue as:

The register\_application function in atk-adaptor/bridge.c in GNOME at-spi2-atk 2.5.2 does not seed the random number generator and generates predictable temporary file names, which makes it easier for local users to create or truncate files via a symlink attack on a temporary socket file in /tmp/at-spi2 [17].

The NVD has assigned CWE-310: Cryptographic Issues as the vulnerability type for this security problem. The lines of code that were problematic are given below [2]:

```
1      /* could this be better, we accept some amount of race in
      getting the temp name */
2      /* make sure the directory exists */
3      mkdir ("/tmp/at-spi2/", S_IRWXU|S_IRWXG|S_IRWXO|S_ISVTX);
4      chmod ("/tmp/at-spi2/", S_IRWXU|S_IRWXG|S_IRWXO|S_ISVTX);
5      app->app_bus_addr = g_malloc(max_addr_length * sizeof(char));
6      #ifndef DISABLE_P2P
7      sprintf (app->app_bus_addr,
              "unix:path=/tmp/at-spi2/socket-%d-%d", getpid(), rand());
```

This is undoubtedly an epistemic error. There is a race between the time that the directory's existence is being checked and appropriate permissions being assigned/checked. The return values of these functions are also not checked. The security of the directory depends on the unpredictability of the directory name which is achieved by appending the random number to the end of the directory's name. However, not seeding the random number generator opens up all doors to brute-force methods to predict directory names eventually allowing a malicious user to create symbolic links, change permissions on the directory, etc. The fundamental design problem here is not separating or compartmentalizing the resource according to privilege. This is supported by the fact that the fix for this security problem makes use of the XDG\_RUNTIME\_DIR which is a Unix specification that defines the base directory relative to which *user-specific* non-essential runtime files and other file objects such as sockets should be stored. As described in [46] the directory MUST be owned by the user, and he MUST be the only one having read and write access to it. Thus, instead of creating the directory in world-readable /tmp directory, it is now created in the user's specific runtime directory.

### 3.4.2 Aleatoric Security Problems

#### **Fedora v15 bug 754509 - bind: Remote denial of service against recursive servers via logging negative cache entry**

Bind is an implementation of the Domain Name System(DNS) protocols. The name "BIND" stands for *Berkeley Internet Name Domain*, because the software originated in the early 1980s at the University of California at Berkeley. The DNS protocols are part of the core Internet standards. They specify the process by which one computer can find another computer on the basis of its name. 'An implementation of DNS protocols' means that the software distribution contains all of the software necessary for asking and answering name service questions.

The BIND software distribution has 3 main parts: A Domain Name System server, a Domain Name System resolver library and Software tools for testing servers. This particular bug caused the BIND 9 resolvers to crash after logging an error in query.c. According to the Internet Systems

Consortium(ISC), organizations across the Internet reported crashes that interrupted service on BIND 9 nameservers which performed recursive queries.[74] It was found that a network event caused BIND 9 resolvers to cache an invalid record and subsequent queries for that record crashed the resolvers with an assertion failure. The common DNS query message format has a fixed length 12-byte header and a variable portion reserved for *question, answer, authority* and *additional* DNS resource records [24].

According to the Supplemental Information on this issue given by the ISC, the crash was triggered by an *accidental* operational error that exposed a *previously unknown* bug in BIND, causing an internal inconsistency.[75] The nameservers that were compromised due to this operational error crashed after caching information in the *Additional* section of the DNS message inappropriately. This bug is described as a *potential zero-day exploit with no workaround* [49].

Considering the above pieces of information that were obtained regarding this bug in the applications substrate of Fedora 15, which was also assigned the CVE identifier CVE-2011-4313, this security problem was classified as an aleatoric issue.

### **Fedora v15 bug 734679 - Remove DigiNotar CA cert from RHEL**

Communication over the web today mostly takes place over the Hyper Text Transfer Protocol Secure(HTTPS)—a result of layering the stateless HTTP over SSL/TLS which adds the security capabilities of the SSL/TLS transport layer protocols to HTTP. Hence HTTPS relies heavily on TLS which uses long term public keys and secret keys to generate short term session keys for encrypted and secure communication between 2 entities over a computer network. In order to ascertain that one is talking to the entity that one intended to communicate with, X.509 certificates are used which necessitates the need for certificate authorities, and a public key infrastructure(PKI) in order to verify the digital certificate, the identity of its owner, its administration etc.[57]

*Certificate Authorities* are responsible for validating the identity of a web server. However, root authorities seldom sign the browser trusted certificates. Rather they sign intermediate certificates which gives them the flexibility of storing their keys offline as well as delegating their signing rights to other third parties. Thus when we trust a certificate signed by a root authority the trust holds a transitive relationship between the user and all other authorities that have signed the certificate.

Thus, there is a web of trust that is established and clearly, security of the certificate is only as strong as its weakest certificate authority. If any certificate authority is compromised or has malicious intent, fraudulent certificates could be generated giving rogues the power to eves-drop into various communication channels and carry out man-in-the-middle attacks.

On August 29, 2011 something similar happened when Google first noticed a rogue \*.google.com certificate issued by DigiNotar to several users in Iran. DigiNotar B.V., a company that provides digital certificate services, hosting a number of certificate authorities in Iran, was victim to an infrastructure compromise by a hacker having the fingerprint *Janam Fadaye Rahbar* who generated

false certificates and intercepted all private communications and searches of Iranian Google users for over two months.[87, 93]

The information discussed above highlights one of the characteristics of the Internet that make it vulnerable—*delegation of trust*.[58] This is clearly an inherent weakness in the system. One may argue that this exists due to limited knowledge and should classify as an epistemic issue, however, from the point of view of a organization conforming to existing standards, this clearly is an aleatoric issue.

### **Fedora v17 bug 872391 - catdoc: buffer overflow flaw**

Catdoc is a MS Word file decoding tool that doesn't attempt to analyze file formatting (it just extracts readable text), but is able to handle all versions of Word and convert character encodings [14]. A Debian bug report [45] reported a buffer overflow bug in catdoc's src/xlsparse.c file. Specifically the code that triggered the overflow was:

```
1 for (i=0;i<NUMOFDATEFORMATS; i++);  
2  FormatIdxUsed[i]=0;
```

Because of the ';' at the end of the for loop, the loop counter variable 'i' was being set to the constant NUMOFDATEFORMATS resulting in an *out-of-bounds* access in the 'FormatIdxUsed' array.[53]

Although, it is hard to judge whether this is accidental or sheer carelessness on the part of the programmer, we chose to consider it as a case of a random typographical error and classified it as an aleatoric security issue.

### **3.4.3 Security problems classified on the “Border” of epistemic and aleatoric problems or left “Unclassified”**

#### **Fedora v15 bug 786988 - CVE-2012-0830 php: remote code exec flaw introduced in the CVE-2011-4885 hashdos fix**

This bug was a result of an incorrect fix to a previous security issue in PHP.[52] PHP 5.3.9 was updated to include a patch to prevent DoS attacks using hash collisions. The developers accomplished this by limiting the maximum possible input parameter to 1000 in the php\_variables.c file by using max\_input\_vars. Due to mistakes in the implementation of this patch, hackers could intentionally inject and execute code resulting in a more critical security issue.

A diff of the file [98] in which changes were made to fix this problem shows that they forgot to free the memory allocated initially before returning from the function. Several proof-of-concept exploits on the Internet show that using brute-force an attacker could rewrite the return address of this function to run attacker injected code.[85] One could argue that this is a case of incomplete analysis on part of the developers, however this could have been a genuine mistake considering the fact that the initial intension was to fix another security issue i.e. CVE-2011-4885. Thus, while there

is an epistemic component to this problem, we cannot rule out the presence of natural tendency of a human to make this mistake. Hence this security problem was classified to be on the 'border'.

### **Fedora v15 Bug 791185 - CVE-2011-3026 thunderbird: libpng: Heap-buffer-overflow in png\_decompress\_chunk**

libpng is the official Portable Network Graphics(PNG) reference library. It is a platform-independent library that contains C functions for handling PNG images. It is used by several programs, including web browsers and potentially server processes.[10]

The MITRE CVE dictionary [5, 96] describes this issue as:

Integer overflow in libpng, as used in Google Chrome before 17.0.963.56, allows remote attackers to cause a denial of service or possibly have unspecified other impact via unknown vectors that trigger an integer truncation.

The National Vulnerability Database(NVD) [7] classifies this problem into Numeric Errors i.e. CWE-189 whose description states that weaknesses in this particular category arise due to miscalculations or conversions of numbers [21]. The vulnerable lines of code are presented below is present in the png\_decompress\_chunk() function in the pngutil.c file.[1, 9]

```
1         png_size_t expanded_size = png_inflate(png_ptr,
          (png_bytep)(png_ptr->chunkdata + prefix_size), chunklength
          - prefix_size, 0/* output */, 0/* output size */);
2         /* some code removed */
3         png_charp text = png_malloc_warn(png_ptr, prefix_size +
          expanded_size + 1);
4         /* some code removed */
5         png_memcpy(text, png_ptr->chunkdata, prefix_size);
```

png\_inflate can return an arbitrarily large number in expanded\_size and therefore the addition on line# 3 will overflow the size attribute which is declared as a 32-bit type resulting in faulty malloc and heap-based buffer overflow. The bytes at chunkdata and prefix\_size are attacker controlled. This security problem behaves as a regular integer overflow case in 32-bit systems while in 64-bit systems the problem exists since the png\_malloc\_warn() function in line 3 above uses a 'png\_uint\_32' for the size argument resulting in truncation of 64-bit argument. Thus we clearly see that this bug is affected by changes in environment as well as due to incomplete analysis and has both epistemic and aleatoric components and hence is classified as 'on the border' or a case where it is hard to say if the problem occurred either due to limited knowledge or was accidental.

### 3.5 Results

“A *sample* is a subset of observations obtained from a larger set, termed a *population*”. “A *random* sample of a specified size is one for which every possible sample of that size has the same chance of being selected from the population” [92]. In this case, the *population* is the set of all problems that are reported against one version of open source Fedora software. Empirical data discussed in this section form a random sample of 60 “security” problems obtained from this population, representing roughly 20-30% of the total number of security problems reported for one version of the product. A web application for generating random numbers [61] was used to randomly select 60 problem reports from each version. Selecting randomly ensured that we retained the time-dependent nature of arrival of security problems in our sample, making it representative of the original population which is also illustrated in Table 3.2. The objective of such exploratory analyses was to answer the question “are the root causes of security-related uncertainties epistemic or aleatory?”. As discussed in Section 3.3 in the security context, it is easier to identify epistemic (knowledge based) biases and handle them as equivalency classes but it may be much harder to target aleatoric security issues. Hence, confirming that the root cause behind *most* security problems is indeed “epistemic” or knowledge driven enabled us to proceed with our analyses. Classification of security problems in this manner is not conventional so that we could deploy software to search for keywords or patterns and therefore each of the 60 problems sampled for each of the four versions of Fedora (i.e., a total of 240 problems) were analyzed manually. While automating the process could have analyzed all security problems reported against one version of a product, the goal was only to assert that our perception of the nature of security problems was correct.

In Fedora 15 [Table A.1], out of 60 randomly selected hand-analyzed security problems, 11 are in the Fedora Core, and rest are in Fedora Applications substrate (layer). 10 core problems appear to be epistemic, while 36 out of 49 Application layer problems are epistemic. From Table A.1 we can see that out of the 60 sampled security problems in Fedora release 15, 30 Application layer Epistemic problems are of the “Top 25” variety, the rest are design flaws an incomplete analysis issues. The kernel in Fedora 15 follows a similar distribution with 5 of the 11 security bugs of the “Top 25” flavor. A graphical representation of the same is shown in Graph 3.3. Similar patterns are observed in Fedora version 16 through 18 as depicted in Table A.1 , Table A.1 and Table A.1 respectively and represented graphically by Figure 3.4, Figure 3.5 and Figure 3.6 respectively.

### 3.6 Time Distribution

For each set of the 60 randomly sampled problems of Fedora releases 15 through 18 we also clustered the security problems by the week that the particular security bug was reported. This gives us the distribution of the problems among the vulnerability categories prior to and after 20 weeks from the

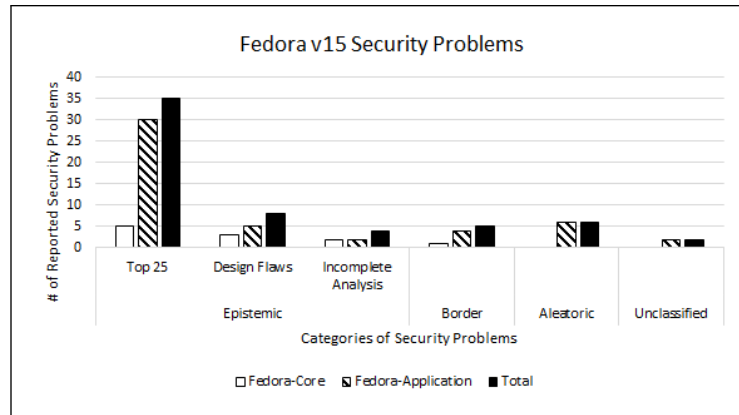


Figure 3.3: Fedora v15 Security Problems Categorical Distribution

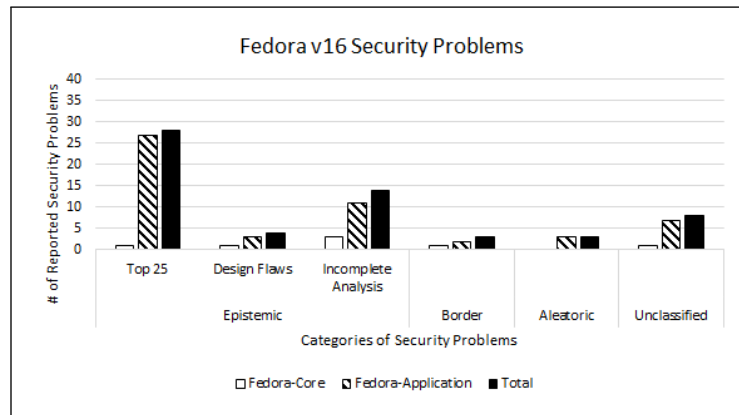


Figure 3.4: Fedora v16 Security Problems Categorical Distribution

release date of the software, which is roughly somewhere halfway through the release time period. Note that the numbers in Tables A.5, A.6, A.7 and A.8 and Graphs 3.7 through 3.14 are assumed to be representative of the original dataset.

### 3.6.1 Fedora Core Substrate

In Fedora 15 through 17, the core layer follows very similar distribution in that most or all security problem reports occur from week 20 onwards. The numbers in the actual reports are also similar and given in Table 3.2. An interesting deviation occurs in Fedora 18 where most problems in the core are reported prior to week 20.

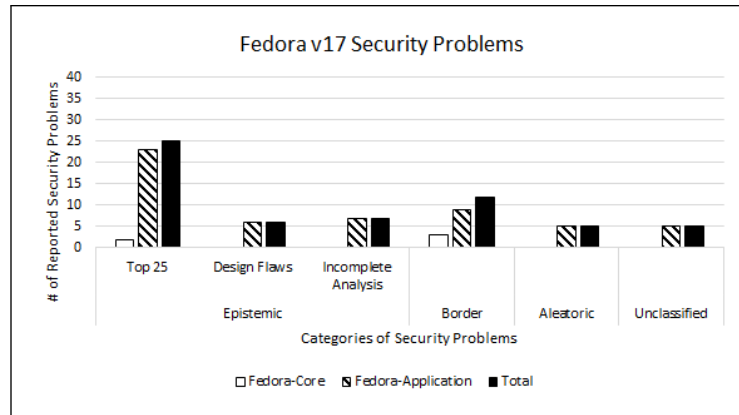


Figure 3.5: Fedora v17 Security Problems Categorical Distribution

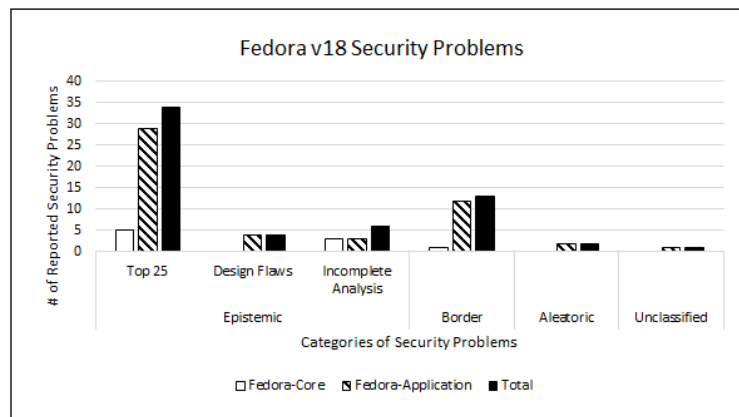


Figure 3.6: Fedora v18 Security Problems Categorical Distribution

Table 3.2: Fedora Core (Kernel) Security Problems (Actual Data)

Fedora Version	Total # of Kernel Security Problems Reported	# of Problems Reported before Week 20	# of Problems Reported After Week 20
15	51	3	48
16	29	7	22
17	7	0	7
18	46	39	7

The distribution of the security problems in the Core substrate is similar across versions 15 through 18 and is given in Table A.5, Table A.6, Table A.7 and Table A.8. A graphical representation of these distributions is given in Figure 3.7, Figure 3.8, Table 3.9 and Table 3.10 respectively. The solid bars give a count of security bugs in a particular category that were reported prior to the 20th week post release while the striped bars denote security bugs reported on or after the 20th week post release. The distribution is similar across the versions and there are no major fluctuations.

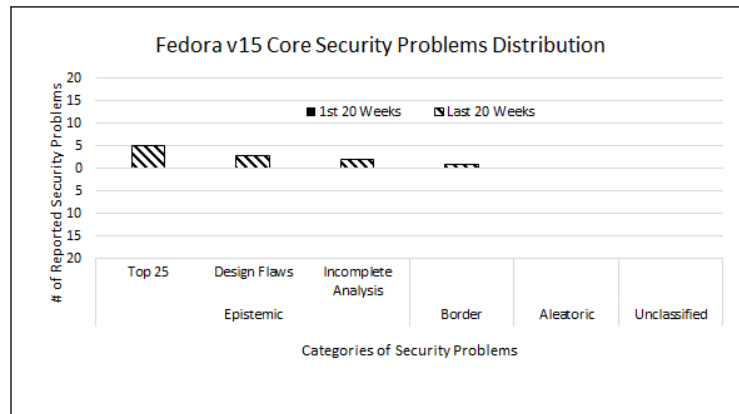


Figure 3.7: Fedora v15 Core Security Problems Time Distribution

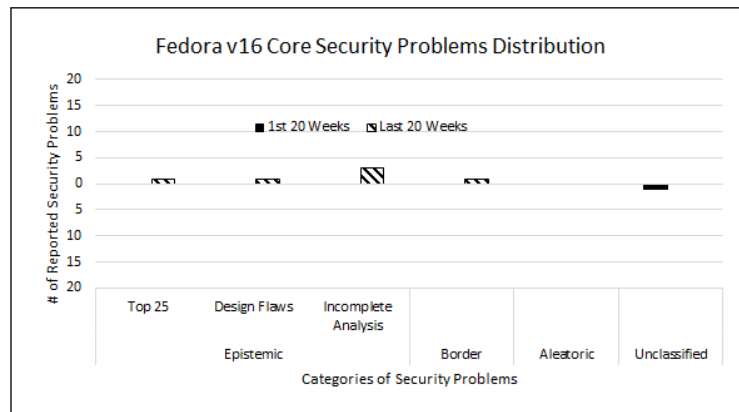


Figure 3.8: Fedora v16 Core Security Problems Time Distribution

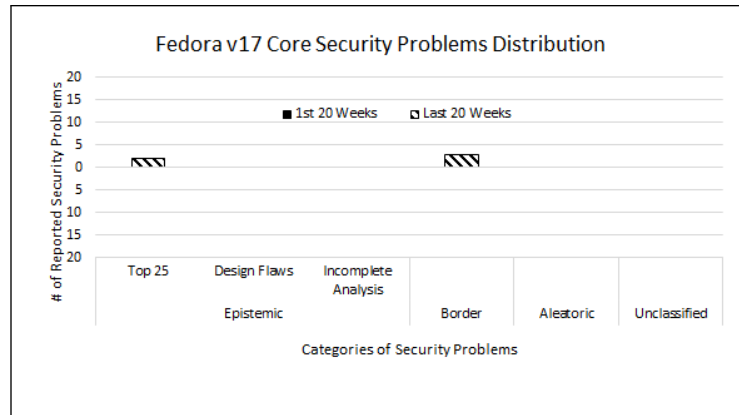


Figure 3.9: Fedora v17 Core Security Problems Time Distribution

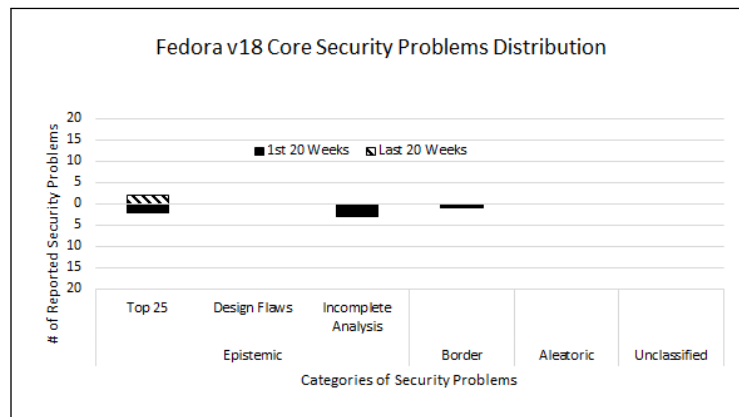


Figure 3.10: Fedora v18 Core Security Problems Time Distribution

### 3.6.2 Fedora Applications Substrate

In the applications substrate for Fedora version 15 through 17, the distribution of security problems that have been reported prior to and after the 20th week from the release date appears to be roughly symmetrical, especially for the security problems classified as “Epistemic” as suggest by Table A.5, Table A.6 and Table A.7. This is indicative of roughly constant vulnerability discovery rate during normal field use.

The solid bars in Figure 3.11, Figure 3.12, Figure 3.13 and Figure 3.14 give the count of the security problems in a category that were reported prior to week 20 while the red bars denote the security problems that were reported on or after week 20 from release. For Fedora version 18 however, the distribution is skewed to prior to week 20. In the original bug report at [3] of 323 Application layer security problems, only 68 were reported on or after week 20 from release. Thus the same in Table A.8

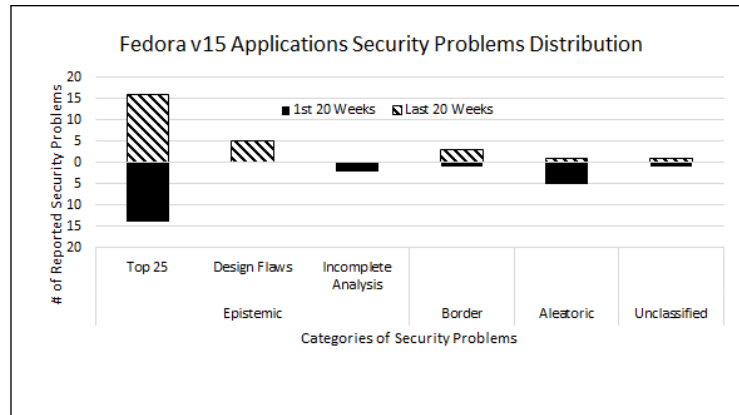


Figure 3.11: Fedora v15 Applications Security Problems Time Distribution

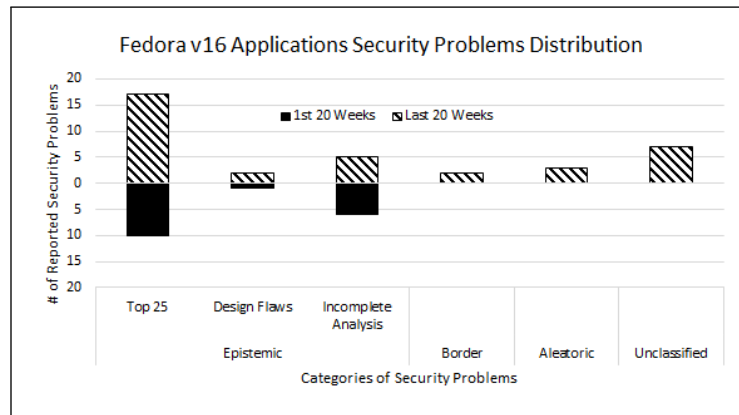


Figure 3.12: Fedora v16 Applications Security Problems Time Distribution

and Figure 3.14 are representative of the original dataset. Across the four versions under study, 60% to 70% of the reported security faults in the applications substrate both, prior to as well as after the 20th week are of the “Top 25” flavor and only a tiny fraction of these problems are instances of design flaws or incomplete analysis.

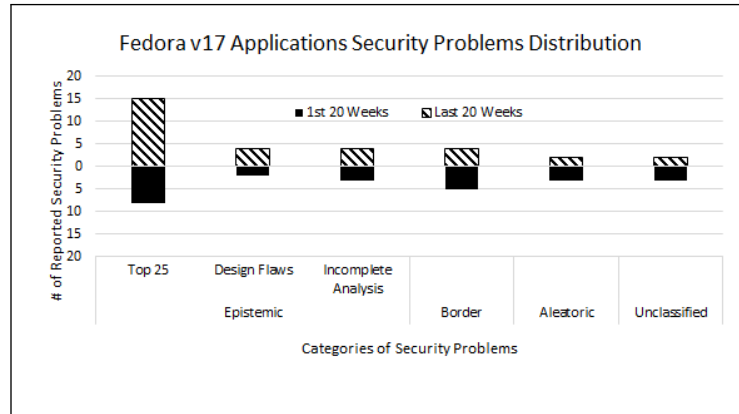


Figure 3.13: Fedora v17 Applications Security Problems Time Distribution

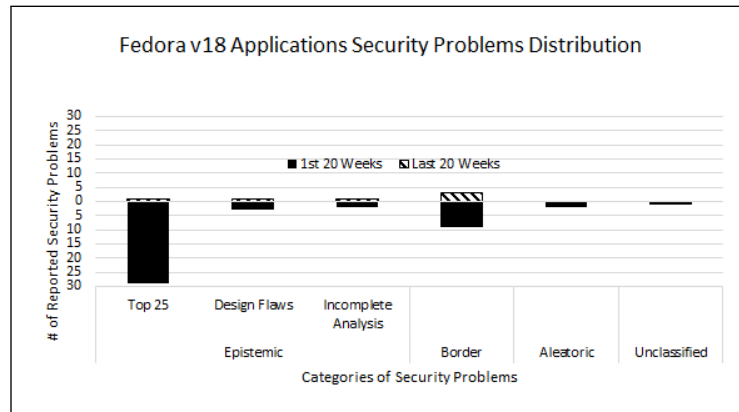


Figure 3.14: Fedora v18 Applications Security Problems Time Distribution

This chapter illustrates how a Software Reliability Growth Model (SRGM) can be used in the security context. Such models could be used to both describe and perhaps predict the behavior of security properties of software such as the number of residual security faults. Input data are Fedora releases 15, 16, 17 and 18 operational problem reports analyzed for calendar time trends.

## 4.1 The Yamada Delayed S-Shaped Software Reliability Growth Model

This *delayed* s-shaped SRGM is a modified version of the Goel-Okumuto model [60, 106]. The model parameters of such stochastic SRGMs can be estimated by the *Maximum Likelihood* method. We use the model not with field failures, but with unique field security problem reports (or vulnerabilities).

### 4.1.1 Model Assumptions [70]

1. The cumulative number of problem reports by time  $t$ ,  $M(t)$ , follows a Poisson process, with mean value function  $\mu(t)$ . The mean value function is of the form  $\mu(t) = \alpha[1 - (1 + \beta t)e^{-\beta t}]$   $\alpha, \beta > 0$ . This is a bounded, non-decreasing function of time with  $\lim_{t \rightarrow \infty} \mu(t) = \alpha < \infty$ , that is, it is a finite failure model. This assumption holds most of the time [64].
2. The time between problem reports of the  $(i - 1)$ st and the  $i$ th depends on the time to failure of the  $(i - 1)$ st.
3. When a failure occurs, the fault which caused it is immediately removed and no other faults are introduced. This particular assumption holds in a delayed fashion. Since we are analyzing

unique closed problem reports against the day when they were reported, this assumption holds in an indirect fashion but is consistent and we believe valid in the context of security problems.

#### 4.1.2 Notation

The notation is similar to [107] and is described below:

$\{N(t), t \geq 0\}$	Counting process representing the cumulative number of closed problem reports up to time $t$ .
$H(t)$	Statistically expected value of $N(t)$ or the <i>mean value</i> function. $H(t)$ is assumed to be non-decreasing in time $t$ with boundary values $H(0) = 0$ and $H(\infty) = a$ where $a$ is the statistically expected initial error content of a software.
$poim(\bullet; \mu)$	Poisson pmf with mean $\mu$ .
$b$	Error detection rate per error in the steady state.
$d(t)$	Error detection rate per error (per unit time) at exposure time $t$ .

#### 4.1.3 Model Form

The SRGM based on NHPP is given by:

$$Pr\{N(t) = n\} = poim(n; H(t)) \quad (n = 0, 1, 2, \dots) \quad (4.1)$$

Where  $H(t)$  for the delayed s-shaped model is given by:

$$H(t) = M(t) = a[1 - (1 + bt)e^{-bt}] \quad b > 0 \quad (4.2)$$

The problem detection rate per error at exposure time  $t$  for the delayed s-shaped SRGM is given by:

$$d(t) = \frac{\frac{d}{dt}(H(t))}{a - H(t)} \quad (4.3)$$

From Eq. 4.2 we have,

$$d(t) \equiv \frac{b^2 t}{1 + bt} \quad (4.4)$$

Equations 4.4 above shows that  $d(t)$  is non-decreasing in  $t$ ,  $t > 0$ , and hence from *Definition 1* in [106]  $H(t)$  is an Increasing Problem Detection Rate (IPDR) function. This means that the delayed S-Shaped SRGM describes an error detection process in which the “*detectability of an error increases with progress in the testing of the software*”.

## 4.2 The CASRE Tool

The Computer Aided Software Reliability Estimation (CASRE) tool [71] is an easy to use software reliability modeling tool developed to provide a systematic and automatic application of software reliability modeling for real-world projects. It is built on top of commonly used SRE models such as Bayesian Jelinski-Moranda Model [67], the Littlewood non-homogeneous Poisson Process Model(LNHPP) [38], the Yamada Delayed S-Shaped Model [107], etc. While this tool is copyrighted by NASA, it is distributed along with [70]. It provides complete automation to software reliability modeling and estimation with a graphical user interface. In Section 4.2.2 and Section 4.2.3 we illustrate how we used this tool on the security data that was randomly sampled from [3] for Fedora versions 15 to 18.

### 4.2.1 Available Models

The CASRE tool takes input data in the form of ASCII based text files with a .dat extension having one of the two formats:

1. *Time Between Failures*: failure#, number of natural or time units since last failure, failure severity class. In our examples, time between closed problem reports.
2. *Failure Counts*: interval#, # of failures in interval, duration of interval, failure severity class. In the current case, failure count is actually count of reported security problem reports.

Based on the type of data that these models work well with, we can classify them into two groups as follows [59], [71]:

1. Time Between Failures Models:
  - (a) Geometric
  - (b) Jelinski-Moranda
  - (c) Littlewood-Verall
  - (d) Musa-Basic
  - (e) Musa-Okumuto
  - (f) NHPP
2. Failure Counts Models:
  - (a) Generalized Poisson
  - (b) NHPP
  - (c) Schneidewind

- (d) Shick-Wolverton
- (e) Yamada Delayed S-Shaped

Since we analyzed and organized security problems reported for Fedora on a weekly basis, we prepared our input data for models that work well with failure counts and modeled the reliability using the Yamada Delayed S-Shaped model which is discussed in Section 4.1. Other models could be used. In fact, Anbalagan has show that Musa's models may be applicable as well [43]. The format of the .dat file used as input to the CASRE tool is given in Appendix B in Section B.1

#### 4.2.2 Using the tool

In this section, we describe a step-by-step process to obtain model estimates from the CASRE 3.x tool:

1. Open the .dat file containing input data as specified in Appendix-A, Table B.1 on page 87.
2. On the plot window select *Cumulative Failures->From model start point* from the *Display* tab.
3. Click on the *Select data range...* option under the *Model* tab on the menu bar.
  - (a) Specify the first data point to be considered for input to the model.
  - (b) Specify the last data point to be considered for input to the model.
  - (c) Specify the last data point to be considered for estimating the parameters of the model. Note that this number will determine the predictive accuracy of the model.
4. Click on the *Select and run models...* option under the *Model* tab on the menu bar.
  - (a) Click and highlight the Yamada-S Shaped model from the left hand box and click the *Add >>* button. You will see Yamada S-Shaped on the right hand side box now.
  - (b) Click on the *Run models* button to run the Yamada S-Shaped model on your input data selected in step 1 above.
5. Select the *Select model results...* option from the *Results* tab on the plot window.
  - (a) If your model had successfully executed, you will see "Yamada S-Shaped" under the *Models Executed* box in the dialog box that opens by performing the previous step. Select and highlight this model.
  - (b) Click on the *Add >>* button. You will see "Yamada S-Shaped" now under the *Results to display* box.
  - (c) Click on OK to display the results.

### 4.2.3 CASRE Fits for Fedora Security Data

#### CASRE fits for Fedora 15 security data

We organized the sampled security problems from Fedora 15 that we manually analyzed, according to the input specification as described in Appendix-A in Table B.1 and selected to run the Yamada S-shaped model on the same. The specifications for the model are given below in Table 4.1.

Table 4.1: CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Security Data

Parameter	Value
Starting Point	1
End Point	22
End point for initial parameter estimation	21
Cumulative number of security problem reports at Week 40	
- Actual	275
- 95% confidence bounds	(123,656)

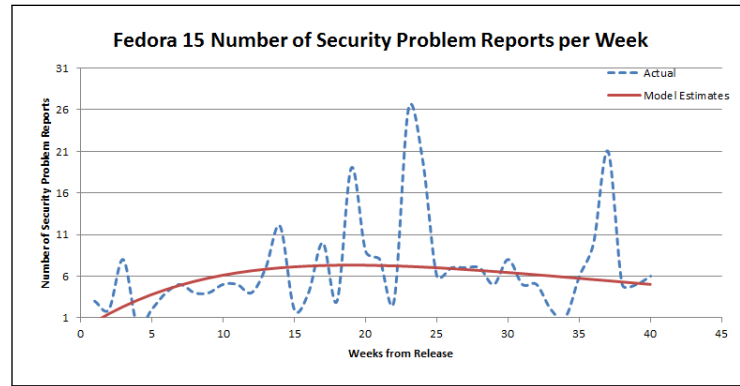


Figure 4.1: Fedora v15 Number of Security Problem Reports Per Week - Yamada fit

Figure 4.1 shows the number of security problem reports per week on the  $Y$  axis versus the calendar weeks from release of Fedora 15 on the  $X$  axis. The broken (blue) line shows the actual number of security problems reported for a week while the smooth (red) line gives the model's estimate of the same.

Figure 4.2 shows the cumulative number of security problems reported on the  $Y$  axis against the (calendar) weeks from release of Fedora 15. Note that the vertical axis in Figure 4.2 is logarithmic. At week 22, which is assumed roughly halfway through the life-cycle, the total number of security

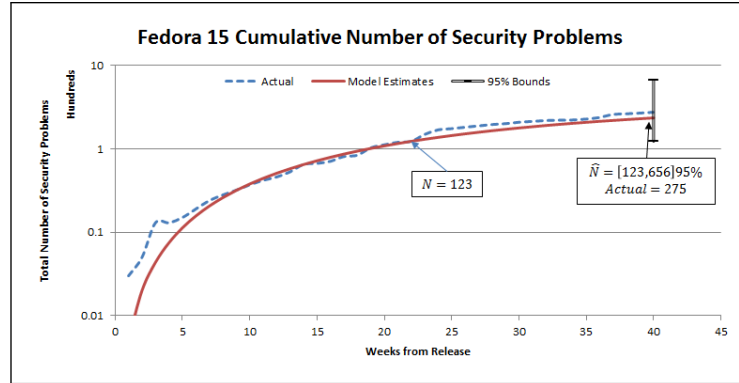


Figure 4.2: Fedora v15 Cumulative Number of Security Problem Reports - Yamada fit

problem reports( $N$ ) is 123. We start projecting from week 22. The model estimates the total number of security problems( $\hat{N}$ , 95% confidence bounds) at week 40 to be between 123 and 656. The actual cumulative number of problems reported from week 1 to week 40 is 275.

#### CASRE fits for Fedora 16 security data

For Fedora 16, since sample data set from week 12 onwards was more representative of the behavior of the software, therefore, we chose that as the starting point for the model. The results of the tool are illustrated in Figure 4.3 and Figure 4.4.

Table 4.2: CASRE: Yamada S-Shaped Model Specifications for Fedora 16 Security Data

Parameter	Value
Starting Point	12
End Point	33
End point for initial parameter estimation	32
Cumulative number of security problem reports at Week 55	
- Actual	296
- 95% confidence bounds	(246,650)

Figure 4.3 shows the number of security problem reports per week on the Y axis versus the calendar weeks from release of Fedora 16 on the X axis. The broken(blue) line shows the actual number of security problems reported for a week while the smooth(red) line gives the model's estimate of the same.

Figure 4.4 shows the cumulative number of security problems reported on the Y axis against the (calendar) weeks from release of Fedora 16. Note that the vertical axis in Figure 4.4 is logarithmic.

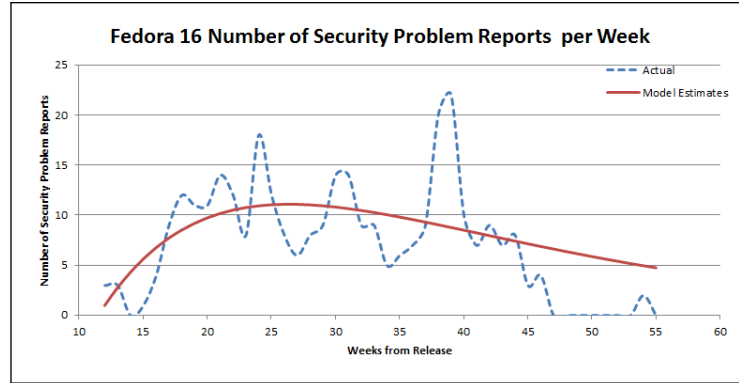


Figure 4.3: Fedora v16 Number of Security Problem Reports Per Week - Yamada fit

At week 33, which is assumed roughly halfway through the period that our selected sample spans over (i.e. 20 weeks from week 12), the total number of security problem reports ( $N$ ) is 195. We start projecting from week 33. The model estimates the total number of security problems ( $\hat{N}$ , 95% confidence bounds) at week 55 (i.e. the 40th week from week 12) to be between 246 and 650. The actual cumulative number of problems reported from week 12 to week 55 is 296.

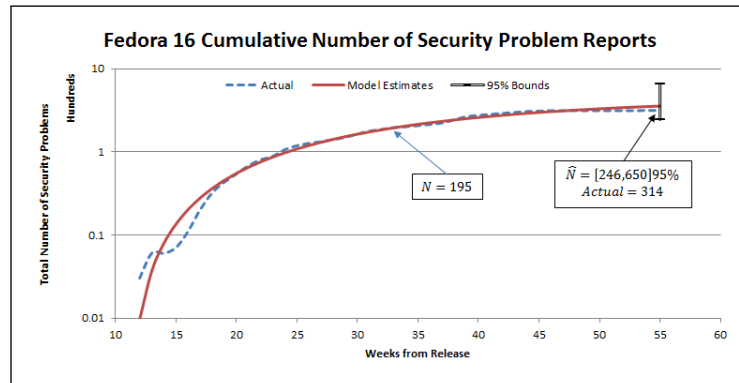


Figure 4.4: Fedora v16 Cumulative Number of Security Problem Reports - Yamada fit

### CASRE fits for Fedora 17 security data

For Fedora 17, the specifications of the model are given in Table 4.3 below. The results are graphically represented by Figure 4.5 and Figure 4.6.

Figure 4.5 shows the number of security problem reports per week on the Y axis versus the calendar weeks from release of Fedora 17 on the X axis. The broken(blue) line shows the actual

Table 4.3: CASRE: Yamada S-Shaped Model Specifications for Fedora 17 Security Data

Parameter	Value
Starting Point	10
End Point	34
End point for initial parameter estimation	33
Cumulative number of security problem reports at Week 56	
- Actual	216
- 95% confidence bounds	(249,728)

number of security problems reported for a week while the smooth(red) line gives the model's estimate of the same.

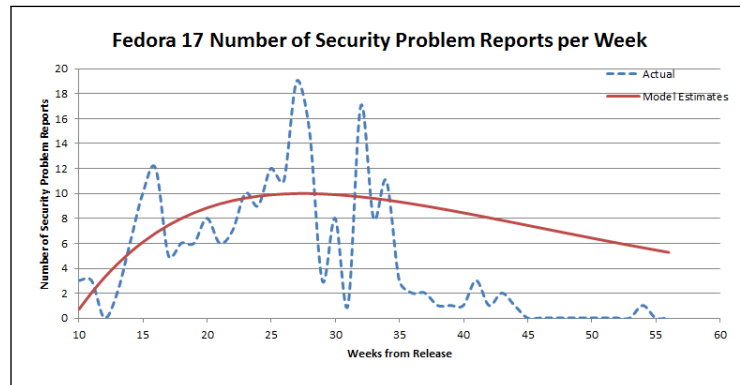


Figure 4.5: Fedora v17 Number of Security Problem Reports Per Week - Yamada fit

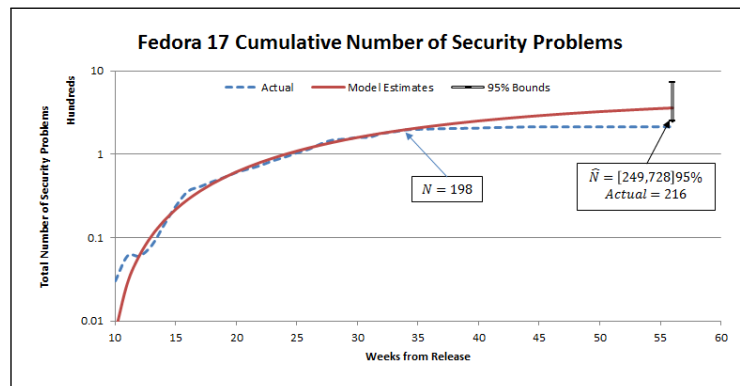


Figure 4.6: Fedora v17 Cumulative Number of Security Problem Reports - Yamada fit

Figure 4.6 shows the cumulative number of security problems reported on the  $Y$  axis against the (calendar) weeks from release of Fedora 17. Note that the vertical axis in Figure 4.6 is logarithmic. At week 34, which is assumed roughly halfway through the period that our selected sample spans over, the total number of security problem reports( $N$ ) is 198. We start projecting from week 34. The model estimates the total number of security problems ( $\hat{N}$ , 95% confidence bounds) at week 56 (i.e. the 47th week from week 10) to be between 249 and 728. The actual cumulative number of problems reported from week 10 to week 56 is 216.

#### CASRE fits for Fedora 18 security data

The results of running the Yamada S-Shaped model on security data sampled for Fedora 18 are graphically represented by Figure 4.7 and Figure 4.8. A key difference is the actual rate of security problems reported per week decreases post week 20 in Fedora 18.

Table 4.4: CASRE: Yamada S-Shaped Model Specifications for Fedora 18 Security Data

Parameter	Value
Starting Point	1
End Point	40
End point for initial parameter estimation	20
Cumulative number of security problem reports at Week 40	
- Actual	338
- 95% confidence bounds	(404,762)

Figure 4.7 shows the number of security problem reports per week on the  $Y$  axis versus the calendar weeks from release of Fedora 18 on the  $X$  axis. The broken(blue) line shows the actual number of security problems reported for a week while the smooth(red) line gives the model's estimate of the same.

Figure 4.8 shows the cumulative number of security problems reported on the  $Y$  axis against the (calendar) weeks from release of Fedora 18. Note that the vertical axis in Figure 4.8 is logarithmic. At week 20, which is assumed roughly halfway through the lifecycle, the total number of security problem reports( $N$ ) is 291. We start projecting from week 20. The model estimates the total number of security problems( $\hat{N}$ , 95% confidence bounds) at week 40 to be between 404 and 762. The actual cumulative number of problems reported from week 1 to week 40 is 338.

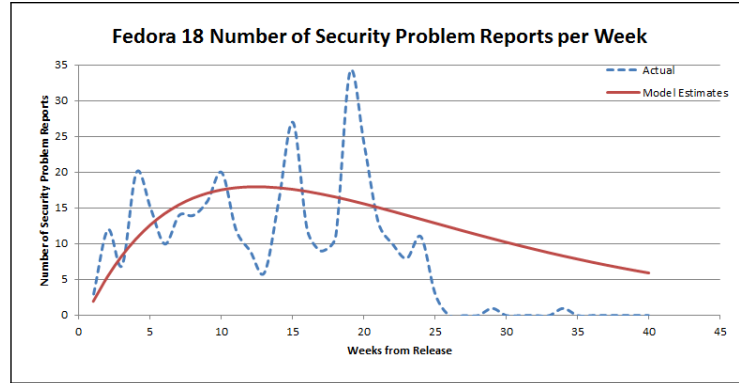


Figure 4.7: Fedora v18 Number of Security Problem Reports Per Week - Yamada fit

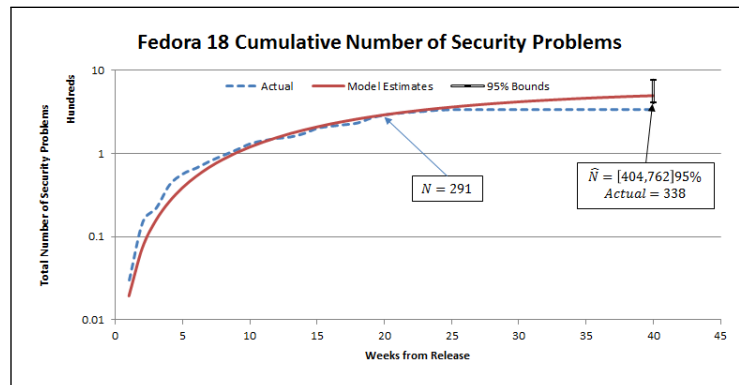


Figure 4.8: Fedora v18 Cumulative Number of Security Problem Reports - Yamada fit

**Effect of parameter estimation endpoint on predictive accuracy - Fedora 15 data**

In this section we present results of varying the parameter estimation endpoint on the predictive accuracy of the model in terms of the High and Low 95% bounds on the cumulative number of failures. Note that the vertical axis in Figures 4.9, 4.10 and 4.11 is logarithmic.

In Figure 4.9 the endpoint for estimating the parameters of the model were the 12th week (see Table 4.5). The model's estimates are off by a great margin from the actual number of security problems reported by week 40.

In Figure 4.10 There is a sharp oscillation in the 95% upper bound on the total number of security problem reports due the corresponding oscillation in the number of reports made in week 22 (3 problems) and week 23 (26 problems). However, in Figure 4.11 we see that the availability of more data (an additional 12 weeks, see Table 4.7) has made the bounds more stable and reduced it by an order of magnitude from the order of  $10^5$  in Figure 4.10 to the order of  $10^2$  respectively. The actual total number of security problems reported by Week 40 for Fedora version 15 is 275.

Table 4.5: CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Security Data Using 12 Weeks Data

Parameter	Value
Starting Point	1
End Point	12
End point for initial parameter estimation	11
Cumulative number of security problem reports at Week 12	46(Actual)
Cumulative number of security problem reports at Week 40	
- Actual	275
- 95% confidence bounds	(46,118)

Table 4.6: CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Security Data Using 24 Weeks Data

Parameter	Value
Starting Point	1
End Point	24
End point for initial parameter estimation	23
Cumulative number of security problem reports at Week 24	169(Actual)
Cumulative number of security problem reports at Week 40	
- Actual	275
- 95% confidence bounds	(169,130573)

Table 4.7: CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Security Data Using 36 Weeks Data

Parameter	Value
Starting Point	1
End Point	36
End point for initial parameter estimation	35
Cumulative number of security problem reports at Week 36	238(Actual)
Cumulative number of security problem reports at Week 40	
- Actual	275
- 95% confidence bounds	(303,539)

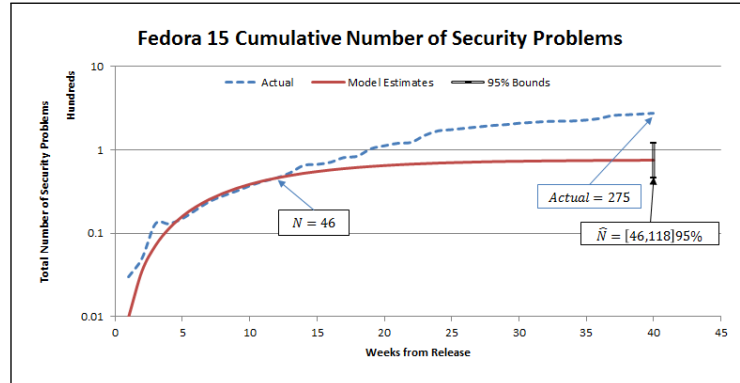


Figure 4.9: Fedora v15 Cumulative Number of Security Problem Reports - 95% Bounds Using 12 Weeks Data

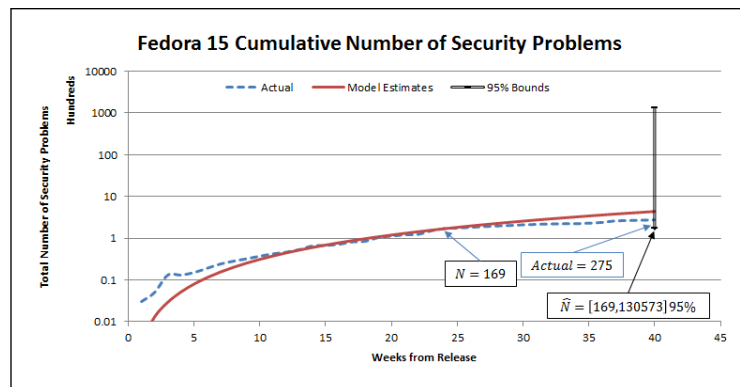


Figure 4.10: Fedora v15 Cumulative Number of Security Problem Reports - 95% Bounds Using 24 Weeks Data

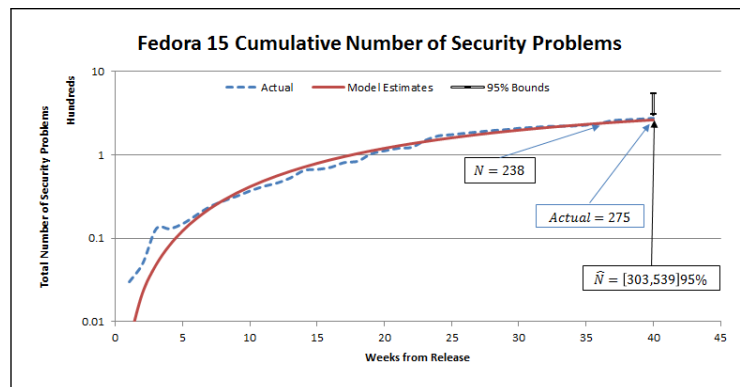


Figure 4.11: Fedora v15 Cumulative Number of Security Problem Reports - 95% Bounds Using 36 Weeks Data

#### 4.2.4 CASRE fits for Fedora 15 non-security data

Unlike Figure 4.1 and Figure 4.2 where there appears to be no real security reliability growth for Fedora 15, Figure 4.12 shows a decreasing rate of non-security problems reported for Fedora 15 over calendar time. Due to the observed growth in reliability, the fit obtained for Fedora 15 non-security data illustrated in Figure 4.13 appears to be better. Note that the vertical axis in Figure 4.13 is logarithmic. In week 28 the actual total number of non-security problems reported for Fedora 15 is 6513. The model predicts there to be anywhere between 6513 and 6824 problems cumulatively in the software at the end of 40 weeks. This gives us an upper bound of around 311 residual problems between week 28 and week 40. The actual number of problems reported between those times is 297.

Table 4.8: CASRE: Yamada S-Shaped Model Specifications for Fedora 15 Non-Security Data

CASRE Options	Selected Value
Starting Point	1
End Point	29
End point for initial parameter estimation	28
Cumulative number of security problem reports at Week 40	
- Actual	6810
- 95% confidence bounds	(6513,6824)

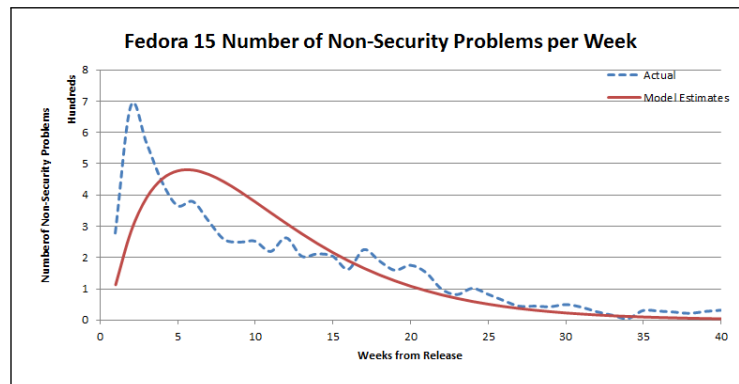


Figure 4.12: Fedora v15 Number of Non-Security Problem Reports Per Week - Yamada fit

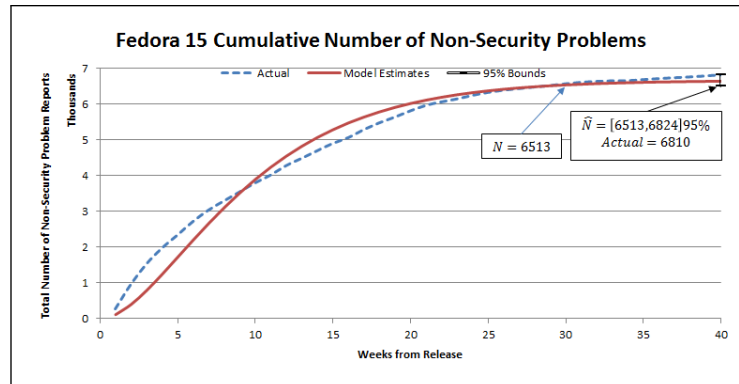


Figure 4.13: Fedora v15 Cumulative Number of Non-Security Problem Reports - Yamada fit

Similar results were obtained for non-security problems reported for releases 16, 17 and 18.

### 4.3 Summary

From the preceding discussion we find that due to the oscillations in security problems reported every week in Fedora 15, the model estimates tend to fluctuate by orders of magnitude and are not stable, perhaps due to the rarity of security problems (which are seen to arrive in bursts). For releases 16, 17 and 18 there were less fluctuations in problem reports per week. In Fedora 16 the problem reports from week 18 onwards were more representative of the entire dataset which spanned over 60 weeks. For release 18, as indicated earlier, more problems were reported in the first 20 weeks, possibly due to several updates in the kernel from 3.6.x to 3.8.x within the initial weeks. When viewed with reference to calendar time, growth in security reliability of a release is not apparent.

For Fedora 15, an upper bound on the number of residual security problems at the end of 40 weeks can be estimated to be  $656 - 123 = 533$  (with projections made at week 22), where the actual cumulative number of problem report till week 22 was 123. This is a very large number. For releases 16, 17 and 18 the number appears to be similar at 455, 530 and 471 respectively.

Thus we see that classical SRE models like [106, 107] can be used to describe security properties such as the count of residual number of security problems of a software (Fedora) under normal operational usage, given the assumption of constant vulnerability discovery rate. The estimate of residual or yet “to be found” security problems could be used to estimate the order of magnitude of additional attack cases that one may need to find the remaining security problems and subsequently improve the growth in security reliability of a Fedora release. The next chapter discusses this idea.

---

## Operational Profile Discovery and Adjustment

---

The sensitivity of current testing processes (which are based on operational profiles) to security problems is not sufficient. Non-security related problems appear to be removed at a higher rate than security problems, as seen in Figure 1.1. Similar results were obtained for Fedora releases 16, 17 and 18 (See Appendix A for figures). We believe that a combination of classical operational testing combined with epistemic and “corner case” analysis is needed. Chapter- 3 presented a description of how to analyze and define the attack profile space using our classification scheme. This gave us the “*what*” component of the attack test cases. We still needed to know “*how many*” more attack test cases would be needed to achieve the desired security intensity objective in our product. Chapter- 4 explained how to employ Software Reliability Growth Models(SRGMs) to estimate the “order of magnitude” of these additional attack test cases. The process of combining these two pieces of information to construct the non-operational profile with a focus on security problems is discussed in this chapter.

### 5.1 Hybrid Approach

We believe that a hybrid approach to vulnerability elimination based on a combination of “classical” and some non-operational “bounded” high-assurance testing along the lines discussed in [70, 90, 103, 104, 102] will yield better results than pure operational testing, or decoupled security testing. Consider the process diagram in Figure 5.1. We propose a combined approach to testing software where we begin by setting the reliability objective (for both security as well as non-security problems) and constructing the operational and some non-operational (attack) profiles for our system. Initially

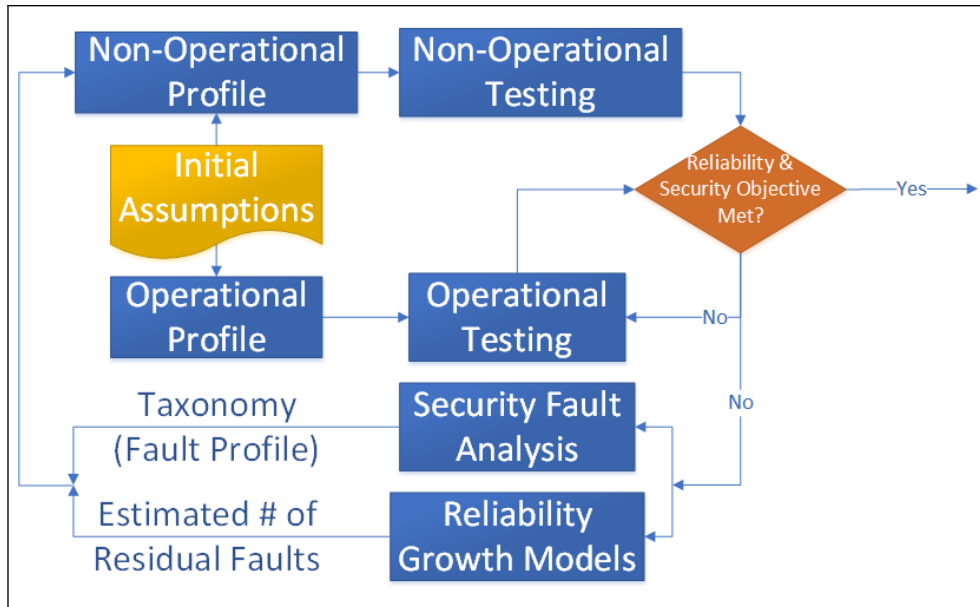


Figure 5.1: The Hybrid Approach

classical SRE methods, metrics and models may be used to track both non-security and security problem detection under normal operational profile. We then model the reliability growth, if any, and estimate the number of residual faults by estimating the lower and upper bounds on the total number of faults of a certain type. This tells us the order of magnitude of the attack test-cases we might still need to generate to target security problems. To define the attack profile space (attack surface) we analyze detected security faults and come up with a fault profile or the vulnerability categories. We then generate those test cases and track security reliability growth. Based on this taxonomy, the estimate of residual faults, and a risk analysis (relative severity of security breaches), we then build non-operational test-suite i.e. we focus our testing efforts by analyzing and generating additional attack test cases for issues that reflect our risk analysis. We then conduct operational and non-operational testing in parallel to accelerate the process of vulnerability identification and removal. After each iteration, we compare the resultant reliability with the initial objective and refine the non-operational test profile if a) the objective hasn't been achieved for the current release, or b) if we observe new problems or higher risk for new or existing problems in the current release. Type b) information can also be used to adjust the test profile for the next release of the product. This sort of continuous learning and refinement could perhaps yield better security reliability than traditional operational testing or decoupled security testing.

## 5.2 Non-Operational Profile Generation

Consider Figure 4.2, which shows a classical SRE fit to F15 cumulative non-security and security data. This fit was obtained using CASRE tool distributed with the Lyu handbook [70]. Fits appear to be good. If one conducts operational testing (as opposed to field use shown here), one may thus be able to, say in week 20, review security problems and try to estimate how many there may still be left in the software. If the projection is made in week 20, then within the 95% confidence bound there appear to be  $640 - 112 = 528$  security problems still to be reported by week 40. That is a large number. It is very likely there is no security reliability growth in the classical sense (visualization of the rate (Figure 1.1) appears to confirm this). That also means that the number of “to be found” security problems may be unbounded in calendar time. Since the security problem discovery hazard rate is not reducing fast enough (or at all) with exposure (testing, time), the number of problems remaining may not be a stable estimate yet. Our results indicate that most of the security problems encountered by that time are epistemic in nature, but do not necessarily fall into the top 25 categories.

To accelerate the process, one might at that point generate a suite of at least 533 “corner” high-impact special attack cases, and continue with both operational and non-operational testing. Non-operational test case generation would be driven by the taxonomy of known security vulnerabilities (in general and collected so far in the process), security fault coverage criteria, failure mode analysis using for example fault trees, and brainstorming and enumerating worst “corner” cases. In fact, operational and non-operational testing could (and should) be done in parallel from the start. The question is whether this type of hybrid testing would improve product security and yield a satisfactory security problem discovery rate.

For example, Figure 5.2 shows a draft profile for Fedora 15 security problems. We manually analyzed 60 randomly selected security problems and divided them into two substrates—Fedora Core (Green row) and Fedora Applications (Orange row). Horizontally we have classified the problems using Top 25, Design Flaw, Incomplete Analysis, Borderline (not sure whether it is an accident or a knowledge issue), Random (aleatoric), and “could not tell”. An attack profile is then formulated based on observed categories and residual fault estimate (about 528 cases). We see that some vulnerabilities may be detectable by direct testing (Pink Columns) while some may need a review of the processes (Gray columns). Similar analyses for Fedora versions 16, 17 and 18 are presented in Figure 5.3, Figure 5.4 and Figure 5.5 respectively.

Note that in the case the when the value in a particular cell is zero, for example, Figure 5.2 there are no aleatoric security problems encountered in the core, we preassign one test case. This is to ensure that we do not rule out the possibility that that operation (in this case attack) can be invoked during test execution [77]. We then assign the remaining test cases to each category proportionately.

Security Fault Categories		Epistemic			Border	Aleatoric	Unclassified	Total
		Top 25	Design Flaws	Incomplete Analysis				
								60
Substrate	Fedora-Core	5	3	2	1	0	0	11
	Attack New Non-Op	44	26	17	9	1	1	98
	Fedora-Application	30	5	2	4	6	2	49
	Attack New Non-Op	266	44	18	36	53	18	435
		Testing	Requirements and Design Review		Testing			

Figure 5.2: Fedora v15: A Draft Attack Profile and Remedy

Security Fault Categories		Epistemic			Border	Aleatoric	Unclassified	Total
		Top 25	Design Flaws	Incomplete Analysis				
								60
Substrate	Fedora-Core	1	1	3	1	0	1	7
	Attack New Non-Op	7	7	23	7	1	7	53
	Fedora-Application	27	3	11	2	3	7	53
	Attack New Non-Op	205	23	83	15	23	53	402
		Testing	Requirements and Design Review		Testing			

Figure 5.3: Fedora v16: A Draft Attack Profile and Remedy

Security Fault Categories		Epistemic			Border	Aleatoric	Unclassified	Total
		Top 25	Design Flaws	Incomplete Analysis				
		2	0	0	3	0	0	5
Substrate	Fedora-Core	16	1	1	24	1	1	44
	Attack New Non-Op	23	6	7	9	5	5	55
	Fedora-Application	203	53	62	80	44	44	486
	Attack New Non-Op		Testing	Requirements and Design Review	Testing			

Figure 5.4: Fedora v17: A Draft Attack Profile and Remedy

Security Fault Categories		Epistemic			Border	Aleatoric	Unclassified	Total
		Top 25	Design Flaws	Incomplete Analysis				
		5	0	3	1	0	0	9
Substrate	Fedora-40Core	38	1	23	7	1	1	71
	Attack New Non-Op	29	4	3	12	2	1	51
	Fedora-Application	227	31	24	94	16	8	400
	Attack New Non-Op		Testing	Requirements and Design Review	Testing			

Figure 5.5: Fedora v18: A Draft Attack Profile and Remedy

### 5.3 Accounting for Risk

It was shown previously [77] that operational profiles constructed for the first release of a system often evolve with the system and may change after deployment when actual data becomes available about usage. Similarly, as we have indicated in Figure 5.1, we need to be able to incorporate new information regarding the non-operational profile of the system.

For example, If we consider Figure 5.2 as the initial non-operational profile but either know or later find out that aleatoric security problems tend to be 10 times more dangerous than Epistemic problems, 4 times as much as security problems classified as “on the border” and 100 times more than any unclassified problems, we would want our test profile to emphasize the same. Traditionally Risk is given by:

$$\text{Risk} = \text{Probability} \times \text{Impact} \quad (5.1)$$

Where Probability is the likelihood that the risk will occur and Impact is the resulting loss incurred if the risk occurred. Keeping this in mind, we could adjust the non-operational profile to account for this risk as follows:

1. Assign the weights (factors) to the (new) categories by giving the most risky category the highest factor. This may depend on cost, loss of lives, etc. and hence is subject to variability from one system to another.
2. Change row #2 in Table 5.2 to show the ratios/fraction of test cases assigned to each category.
3. Multiply the ratios by the weights assigned to their respective categories.
4. Add the above ratios. The sum of these is going to be different from 1.
5. Divide the ratios by the new sum to normalize their values so that they sum up to 1.
6. Preassign test cases to categories who's ratio falls below a certain threshold i.e. ensure at least one test case is assigned to less frequentl occurring problems.
7. Assign the remaining test cases to the other categories proportionately.

We illustrate the above step by step for the draft profile constructed in Figure 5.2. The above scenario is only a hypothetical case. We don't have evidence that says that one category is more dangerous than the other. While we did have the CVSS scores [28] for Fedora 15 problems, which gave us the statistics in Table 5.1, they are not generalizable. But, in the case one wishes to emphasize on a particular security problem, which has proven to be most dangerous in previous releases of their software, then we illustrate one way of how this could be done.

Table 5.1: Statistics for CVSS Scores Assigned to Fedora 15 Security Problems

Category	Mean	Standard Deviation	Median	Max	Min
Epistemic	6.41	2.28	6.8	10.0	2.1
Border	6.88	0.99	7.5	7.8	5.8
Aleatoric	6.37	1.40	6.3	7.8	5.0

**Step One**

Figure 5.6 shows the assignments of weights to the categories in Row #1 enclosed in the round brackets. We have also converted the test cases assigned to each category of problems to their corresponding ratios.

Security Fault Categories		Epistemic			Border	Aleatoric	Unclassified	Total	
		Top 25	Design Flaws	Incomplete Analysis				60	
Substrate	Fedora Core	Attack New Non-Op	44	26	17	9	1	1	98
		Ratios	0.449	0.265	0.174	0.092	0.010	0.010	1
	Fedora Applications	Attack New Non-Op	266	44	18	36	53	18	435
		Ratios	0.612	0.101	0.041	0.083	0.122	0.041	1

Figure 5.6: Accounting for Risk in Fedora v15 - Step One

**Step Two**

In Figure 5.7 we have multiplied the ratios generated in the previous step by the weights assigned to each category. Now the sum of these will be different than 1.

Security Fault Categories			Epistemic (0.1)			Border (0.25)	Aleatoric (1)	Unclassified (0.01)	Total
			Top 25	Design Flaws	Incomplete Analysis				60
Substrate	Fedora Core	Attack New Non-Op	44	26	17	9	1	1	98
		Ratios	0.0449	0.0265	0.0174	0.0229	0.0100	0.0001	0.1218
	Fedora Applications	Attack New Non-Op	266	44	18	36	53	18	435
		Ratios	0.0612	0.0101	0.0041	0.0207	0.1218	0.0004	0.2183

Figure 5.7: Accounting for Risk in Fedora v15 - Step Two

### Step Three

In Figure 5.8 we normalized the ratios generated in step two above so that they sum up to 1. This is in conformance to the process discussed in [77].

Security Fault Categories			Epistemic (0.1)			Border (0.25)	Aleatoric (1)	Unclassified (0.01)	Total
			Top 25	Design Flaws	Incomplete Analysis				60
Substrate	Fedora Core	Attack New Non-Op	44	26	17	9	1	1	98
		Ratios	0.369	0.218	0.142	0.188	0.082	0.001	1
	Fedora Applications	Attack New Non-Op	266	44	18	36	53	18	435
		Ratios	0.280	0.046	0.019	0.095	0.558	0.002	1

Figure 5.8: Accounting for Risk in Fedora v15 - Step Three

### Step Four

We preassigned at least one test case to categories which had a low ratio as discussed in Section 5.2. For example, problems that remained unclassified in both the Core as well as the Applications

substrates. The remaining test cases were then assigned proportionately among the other categories. Figure 5.9 illustrates this.

Security Fault Categories		Epistemic (0.1)			Border (0.25)	Aleatoric (1)	Unclassified (0.01)	Total	
		Top 25	Design Flaws	Incomplete Analysis				60	
Substrate	Fedora Core	Problems	5	3	2	1	0	0	11
		Attack New Non-Op	36	21	14	18	8	1	98
	Fedora Applications	Problems	30	5	2	4	6	2	49
		Attack New Non-Op	122	20	9	41	242	1	435

Figure 5.9: Accounting for Risk in Fedora v15 - Step Four

## 5.4 Summary

As mentioned earlier (in Section 5.3), the proposal for adjusting the non-operational profile, to emphasize on a particular category of vulnerabilities has not been confirmed empirically using real data and test cases. However, SRE risk-based analyses by [78, 79, 77] seem to confirm that this type of approach works for software in general. We expect that our adaptation of this method to security problems could, in theory at the least, help improve the rate of detecting security issues. A risk based testing strategy offers the advantages of reduced resource consumption and improved quality by focussing on more critical functions of a program [48].

The empirical data used for analyses are obtained from field operations i.e. once the software has been deployed. The question to be asked is whether observations about the field behavior of one release can drive testing of the next release. From Figure 5.2 and Figure 5.3 above, it is seen that the distribution of security problems reported in the “Core” and “Applications” substrates for Fedora releases 15 and 16 are similar. Likewise, the distribution of the security problems in each (sub-)category for both releases is similar. Theoretically, it appears that information obtained for Fedora 15 field usage could have been used to drive testing in Fedora 16 release. Since information about pre-release testing activity (for example, the number of test cases) for Fedora 16 was not available, we constructed our non-operational profiles based on the field information for Fedora 16

and think that it is reflective of the pre-release testing profile.

Based on the symmetry observed in the distribution of the security problems reported in the field for Fedora releases 15, 16, 17 and 18 under normal operational usage we find two applications for this approach:

- in production, one could simply count the vulnerabilities and project over the next period assuming constant vulnerability discovery rate
- in testing phase, to accelerate the process, one might leverage collected vulnerability information to generate non-operational test-cases aimed at vulnerability categories

The observed distributions of security problems reported under normal “operational” usage appear to support the above approach - i.e., what is learned say in the first  $x$  weeks can then be leveraged in selecting test cases in the next stage. Similarly, what is learned about a product  $y$  weeks after its release may be very indicative of its vulnerability profile for the rest of its life given the assumption of constant vulnerability discovery rate. That operational and non-operational testing could (and should) be done in parallel from the start, is intuitive. The question is whether this type of hybrid testing would improve product security and yield a satisfactory security problem discovery rate.

---

## Conclusions and Future Work

---

The goal of this thesis was to assess whether “classical” software reliability engineering (SRE) models can be used in the security context to provide useful information with respect to the security quality of a software product. We proposed a hybrid security problem elimination approach based on risk-driven non-operational testing. In several ways this work just opens up a number of questions regarding practical effectiveness and efficiency of the approach.

### 6.1 Conclusions

- Fedora security problems were re-classified, based on the type of uncertainty (epistemic or aleatory) involved in our assessment of their introduction into the system. Such a taxonomy, as discussed in Chapter- 3 could help us understand what kind of counter-measures may be most effective in preventing, finding and mitigating these problems. Knowing the distribution of problems based on this taxonomy could also help in constructing our attack test cases to include tests for problems that are not represented in the normal operational test profile of our system. For example, our analyses of Fedora releases 15, 16, 17 and 18 show that nearly 75% of all field security problems are epistemic in nature, 60% to 75% of which are of the “Top 25” variety. This suggests that an automated repeatable strategy of generating test cases based on these Top 25 problems could help to considerably reduce their occurrence in the field.
- The use of “classical” SRGMs [106, 107] in the security context was described in Chapter- 4. The input data for the model were field security problems reported for Fedora releases 15, 16, 17 and 18 under “normal” operational usage. The predictive ability of the model was

demonstrated for Fedora 15 both, security and non-security problem reports. It was shown that these models could be used to estimate the residual number of security problems in Fedora releases 15, 16, 17 and 18.

- The results of the analyses obtained by categorizing the security problem reports and estimating the residual problems were used in constructing a security oriented, risk-based non-operational testing profile for Fedora releases 15, 16, 17 and 18. This was based on the observation that the security problem discovery rate for open source Fedora releases 15, 16, 17 and 18 was roughly constant in calendar time. A hybrid approach to testing software based on combined operational and non-operational testing, with a focus on security problems to improve the security problem discovery rate was proposed. The observed distributions of field usage data seem to support the proposed approach. The application of such an approach during the testing phases to select non-operational attack cases aimed at the observed vulnerability categories as well as in production to drive the testing of the next stage/release is briefly discussed. However, the efficiency of this process in discovering security problems has not been verified empirically and provides a possible direction for future work.

## 6.2 Future Work

- The data used in our analyses are obtained from normal operational use of Fedora releases 15, 16, 17 and 18. Inferences about the pre-release testing profile are drawn based on this field usage data. Further, we suggest that operational and non-operational testing be done in parallel from the start. The question is whether this type of hybrid testing would improve product security and yield a satisfactory security problem discovery rate. This is one of the main aspects where further work is needed. Hence, confirmatory experiments may need to be conducted to:
  - observe effectiveness of the new (hybrid) process in improving the sensitivity of the test profile towards security problems (both for the current release as well as to drive testing of a later release), and
  - evaluate the costs and benefits of:
    - collecting and maintaining data required by SRE models
    - integrating non-operational testing techniques
- The Yamada Delayed S-Shaped model discussed in Chapter- 4 in Section 4.1 assumes that the total number of failures experienced by a software over infinite time is finite. In the context of security problems, this assumption may not necessarily hold true. Hence, an investigation of infinite failure models such as Musa's Logarithmic Poisson Execution Time (LPET) model

([80]) as discussed by Anbalagan [43] must ensue, and they must be revisited for verifying the underlying assumptions in the context of software security [64].

- To prove the validity of this process, this sort of analytical study could be extended to other open source security data such as Ubuntu [35], Android [12], etc. since they are widely used and their problem reports and information about their usage are readily available. The generality of these results outside the open source domain, on proprietary products by vendors like Microsoft, Apple, Cisco, etc. must also be checked.

## REFERENCES

- [1] Code Search. <https://code.google.com/p/chromium/>.
- [2] Proposed patch for bug #678348. <https://bugzilla.gnome.org/attachment.cgi?id=216971&action=diff>.
- [3] Red Hat Bugzilla. <https://bugzilla.redhat.com/>.
- [4] Red Hat Family Tree. [http://upload.wikimedia.org/wikipedia/commons/a/a3/Redhat\\_family\\_tree\\_11-06.png](http://upload.wikimedia.org/wikipedia/commons/a/a3/Redhat_family_tree_11-06.png).
- [5] CVE-2011-3026. <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3026>, August 2011.
- [6] CVE-2011-4128 gnutils: buffer overflow in Overflow\_session\_get\_data() (GNUTLS-SA-2011-2). <https://access.redhat.com/security/cve/CVE-2011-4128>, November 2011.
- [7] Bug 790737 - CVE-2011-3026 libpng: Heap buffer overflow in png\_decompress\_chunk. <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3026>, February 2012.
- [8] CVE-2012-5621 ekiga: DoS (crash) after receiving call from other party with not UTF-8 valid name. <https://access.redhat.com/security/cve/CVE-2012-5621>, 2012.
- [9] Issue 112822:Security: Heap-buffer-overflow in png\_decompress\_chunk. <https://code.google.com/p/chromium/issues/detail?id=112822>, February 2012.
- [10] libpng: Multiple vulnerabilities. <http://www.gentoo.org/security/en/glsa/glsa-201206-15.xml>, June 2012.
- [11] What is Puppet? <http://puppetlabs.com/puppet/what-is-puppet>, May 2013.
- [12] Android Operating System. <http://www.android.com/>, May 2014.
- [13] Assistive Technology Service Provider Interface. [http://en.wikipedia.org/wiki/Assistive\\_Technology\\_Service\\_Provider\\_Interface](http://en.wikipedia.org/wiki/Assistive_Technology_Service_Provider_Interface), April 2014.

- [14] catdoc. <http://freecode.com/projects/catdoc>, May 2014.
- [15] Common Vulnerabilities and Exposures. <http://cve.mitre.org/about/faqs.html>, May 2014.
- [16] CVE-2012-2661 rubygem-activerecord: SQL injection when processing nested query parameters. <https://access.redhat.com/security/cve/CVE-2012-2661>, May 2014.
- [17] CVE-2012-3378 at-spi2-atk: insecure temporary file handling. <https://access.redhat.com/security/cve/CVE-2012-3378>, May 2014.
- [18] CVE-2014-0160 openssl: information disclosure in handling of TLS heartbeat extension packets. <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160>, April 2014.
- [19] CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow'). <http://cwe.mitre.org/data/definitions/120.html>, May 2014.
- [20] CWE-131: Incorrect Calculation of Buffer Size. <http://cwe.mitre.org/data/definitions/131.html>, May 2014.
- [21] CWE-189: Numeric Errors. <http://cwe.mitre.org/data/definitions/189.html>, May 2014.
- [22] CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'). <http://cwe.mitre.org/data/definitions/78.html>, May 2014.
- [23] CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). <http://cwe.mitre.org/data/definitions/89.html>, May 2014.
- [24] DNS Protocol. [http://technet.microsoft.com/en-us/library/dd197470\(v=ws.10\).aspx](http://technet.microsoft.com/en-us/library/dd197470(v=ws.10).aspx), May 2014.
- [25] Ekiga. <http://ekiga.org/>, May 2014.

- [26] Fedora Operating System. [http://en.wikipedia.org/wiki/Fedora\\_\(operating\\_system\)](http://en.wikipedia.org/wiki/Fedora_(operating_system)), May 2014.
- [27] National Vulnerability Database. <http://nvd.nist.gov/>, May 2014.
- [28] NVD Common Vulnerability Scoring System Support v2. <http://nvd.nist.gov/cvss.cfm>, May 2014.
- [29] Open Source Vulnerability Database. <http://osvdb.org/about>, May 2014.
- [30] OpenSSL. <https://www.openssl.org/>, April 2014.
- [31] Red Hat Bugzilla - Field Descriptions.  
url<https://bugzilla.redhat.com/page.cgi?id=fields.html>, May 2014.
- [32] SQL Injection Vulnerability in Ruby on Rails. <https://groups.google.com/forum/#!original/rubyonrails-security/dUai00GWL1k/35L8J8amLfwJ>, May 2014.
- [33] The GnuTLS Transport Layer Security Library. <http://gnutls.org/>, May 2014.
- [34] The Heartbleed Bug. <http://heartbleed.com/>, April 2014.
- [35] Ubuntu. <http://www.ubuntu.com/>, May 2014.
- [36] Uncertainty Quantification. [http://en.wikipedia.org/wiki/Uncertainty\\_quantification](http://en.wikipedia.org/wiki/Uncertainty_quantification), May 2014.
- [37] RP Abbott, JS Chin, JE Donnelley, WL Konigsford, S Tokubo, DA Webb, and TA Linden. Security analysis and enhancements of computer operating systems: The RISOS project. Technical report, Technical Report NBSIR 76-1041, Institute for Computer Sciences and Technology, National Bureau of Standards, US, 1976.
- [38] Abdallah A. Abdel-Ghaly, PY Chan, and Bev Littlewood. Evaluation of competing software reliability predictions. *Software Engineering, IEEE Transactions on*, (9):950–967, 1986.

- [39] Norman A Abrahamson. Aleatory Variability and Epistemic Uncertainty. <http://www.ce.memphis.edu/7137/PDFs/Abrahamson/C05.pdf>.
- [40] P. Anbalagan and M. Vouk. On Reliability Analysis of Open Source Software - FEDORA. In *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, pages 325–326, Nov 2008.
- [41] P. Anbalagan and M. Vouk. Towards a Unifying Approach in Understanding Security Problems. In *Software Reliability Engineering, 2009. ISSRE '09. 20th International Symposium on*, pages 136–145, Nov 2009.
- [42] P. Anbalagan and M. Vouk. Towards a Bayesian Approach in Modeling the Disclosure of Unique Security Faults in Open Source Projects. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 101–110, Nov 2010.
- [43] Prasanth Anbalagan. *A study of software security problem disclosure, correction and patching processes*. PhD thesis, North Carolina State University, 2011.
- [44] Alex Baker. 14 Years of SQL Injection and still the most dangerous vulnerability. <https://www.netsparker.com/blog/sql-injection-vulnerability-history/>, May 2014.
- [45] Nick Bane and Olly Betts. catdoc: Extra ';' turns for loop into a buffer overflow. <https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=692076>, 2012.
- [46] Waldo Bastian, Ryan Lortie, and Lennart Poettering. XDG Base Directory Specification. <http://standards.freedesktop.org/basedir-spec/basedir-spec-latest.html>, November 2012.
- [47] Matt Bishop and David Bailey. A critical analysis of vulnerability taxonomies. Technical report, DTIC Document, 1996.
- [48] Barry W Boehm. *Tutorial: software risk management*. IEEE Computer Society Press, 1989.

- [49] Richard Chirgwin. BIND security update protects against serious server crash - Attacks may already be underway. [http://www.theregister.co.uk/2011/11/16/bind\\_in\\_a\\_bind\\_again/](http://www.theregister.co.uk/2011/11/16/bind_in_a_bind_again/), May 2014.
- [50] Steve Christey, M Brown, D Kirby, B Martin, and A Paller. CWE/SANS Top 25 Most Dangerous Software Errors. <https://cwe.mitre.org/top25>, 2011.
- [51] Vincent Danen. Bug 773025 - CVE-2012-0035 emacs: CEDET global-edo-mode file loading vulnerability. [https://bugzilla.redhat.com/show\\_bug.cgi?id=773025](https://bugzilla.redhat.com/show_bug.cgi?id=773025), 2012.
- [52] Vincent Danen. Bug 786988 - CVE-2012-0830 php: remote code exec flaw introduced in the CVE-2011-4885 hashdos fix. [https://bugzilla.redhat.com/show\\_bug.cgi?id=786988](https://bugzilla.redhat.com/show_bug.cgi?id=786988), February 2012.
- [53] Vincent Danen. Bug 872390 - catdoc: buffer overflow flaw. [https://bugzilla.redhat.com/show\\_bug.cgi?id=872390](https://bugzilla.redhat.com/show_bug.cgi?id=872390), 2012.
- [54] Vincent Danen. Bug 841671 - CVE-2012-3378 at-spi1-atk: insecure temporary file handling, February 2013.
- [55] Eugen Dedu. Validate UTF-8 strings before showing them. <https://git.gnome.org/browse/ekiga/commit/?id=7d09807257>, May 2014.
- [56] Wenliang Du and Aditya P Mathur. Categorization of software errors that led to security breaches. In *21st National Information Systems Security Conference*, pages 392–407, 1998.
- [57] Zakir Durumeric, James Kasten, Michael Bailey, and J Alex Halderman. Analysis of the HTTPS certificate ecosystem. In *Internet Measurement Conference*, 2013.
- [58] Peter Eckersley. Iranian hackers obtain fraudulent HTTPS certificates: How close to a Web security meltdown did we get? *Electronic Frontier Foundation, March*, 2011.
- [59] BH Far. SENG 521 Software Reliability & Software Quality, Software Reliability Tools (Chapter 12).

- [60] A.L. Goel and K. Okumoto. Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. *Reliability, IEEE Transactions on*, R-28(3):206–211, Aug 1979.
- [61] Dr Mads Haahr. True Random Number Generator. `random.org`, May 2014.
- [62] Pete Herzog. Open-source security testing methodology manual. *Institute for Security and Open Methodologies (ISECOM)*, 2003.
- [63] Carl E Landwehr, Alan R Bull, John P McDermott, and William S Choi. A taxonomy of computer program security flaws. *ACM Computing Surveys (CSUR)*, 26(3):211–254, 1994.
- [64] Da Young Lee, M. Vouk, and L. Williams. Using software reliability models for security assessment - Verification of assumptions. In *Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium on*, pages 23–24, Nov 2013.
- [65] Da Young Lee, Mladen Vouk, and Laurie Williams. Uncertainty in Software Security Engineering, March 2013.
- [66] Jan Lieskovsky. Bug 790737 - CVE-2011-3026 libpng: Heap buffer overflow in png\_decompress\_chunk. [https://bugzilla.redhat.com/show\\_bug.cgi?id=883058](https://bugzilla.redhat.com/show_bug.cgi?id=883058), December 2012.
- [67] Bev Littlewood and Ariela Sofer. A Bayesian modification to the Jelinski-Moranda software reliability growth model. *Software engineering journal*, 2(2):30–41, 1987.
- [68] Eric M. Ludlam. Collection of Emacs Development Environment Tools All (Fundamental). <http://cedet.sourceforge.net/>, May 2014.
- [69] Michael R Lyu. Software reliability engineering: A roadmap. In *2007 Future of Software Engineering*, pages 153–170. IEEE Computer Society, 2007.
- [70] Michael R Lyu et al. *Handbook of software reliability engineering*, volume 222. IEEE computer society press CA, 1996.

- [71] M.R. Lyu and A. Nikora. CASRE: a computer-aided software reliability estimation tool. In *Computer-Aided Software Engineering, 1992. Proceedings., Fifth International Workshop on*, pages 264–275, Jul 1992.
- [72] Nikos Mavrogiannopoulos. gnutls\_session\_get\_data: Fix possible buffer overflow. <http://git.savannah.gnu.org/gitweb/?p=gnutls.git;a=commitdiff;h=190cef6eed37d0e73a73c1e205eb31d45ab60a3c>, 2011.
- [73] Stuart McDonald. SQL Injection: Modes of attack, defense, and why it matters. *White paper, GovernmentSecurity.org*, 2002.
- [74] Michael McNally. CVE-2011-4313: BIND 9 Resolver crashes after logging an error in query.c. <https://kb.isc.org/article/AA-00544/74/CVE-2011-4313%3A-BIND-9-Resolver-crashes-after-logging-an-error-in-query.c.html>, November 2011.
- [75] Michael McNally. CVE-2011-4313 FAQ and Supplemental Information. <https://deephought.isc.org/article/AA-00549>, February 2012.
- [76] J.D. Musa. Operational profiles in software-reliability engineering. *Software, IEEE*, 10(2):14–32, March 1993.
- [77] John D Musa. *Software Reliability Engineering More Reliable Software Faster Development and Testing*. McGraw-Hill New York, 1999.
- [78] John D Musa. *Software reliability engineering: more reliable software, faster and cheaper*. Tata McGraw-Hill Education, 2004.
- [79] John D Musa, Anthony Iannino, and Kazuhira Okumoto. *Software reliability*. McGraw-Hill New York, 1987.
- [80] John D Musa and Kazuhira Okumoto. Application of Basic and Logarithmic Poisson Execution Time Models in Software Reliability Measurement. In *Software System Design Methods*, pages 275–298. Springer, 1986.

- [81] David M. Nicol, William H. Sanders, William L. Scherlis, and Laurie A. Williams. Science of Security Hard Problems: A Label Perspective. Technical report, University of Illinois at Urbana-Champaign and Carnegie Mellon University and North Carolina State University, November 2012.
- [82] Oedipus. Accessibility. <http://www.linuxfoundation.org/collaborate/workgroups/accessibility>, May 2014.
- [83] Hiroshi Oota. [CEDET-devel] Security flaw in EDE. <http://sourceforge.net/p/cedet/mailman/message/28649762/>, 2012.
- [84] Dusko Pavlovic. On bugs and elephants: Mining for a science of security. *Developing a blueprint for a science of cybersecurity*, 19(2):23–29, 2012.
- [85] pilate. Simple proof of concept for PHP bug (CVE-2012-0830) described by Stefan Esser. <https://gist.github.com/pilate/1725489>, 2012.
- [86] Daniel Pittman. k5login can overwrite arbitrary files as root. <http://projects.puppetlabs.com/issues/9794>, 2011.
- [87] JR Prins and Business Unit Cybercrime. DigiNotar Certificate Authority breach ‘Operation Black Tulip’. *Fox-IT, November*, 2011.
- [88] Jane Radatz, Anne Geraci, and Freny Katki. IEEE standard glossary of software engineering terminology. *IEEE Std*, 610121990:121990, 1990.
- [89] JR RICHARD C. SCHAEFFER. National Information Assurance (IA) Glossary. Technical Report 4009, Committee on National Security Systems, 2010.
- [90] Anthony T Rivers and Mladen A Vouk. Resource-constrained non-operational testing of software. In *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on*, pages 154–163. IEEE, 1998.

- [91] Anthony T. Rivers, Mladen A. Vouk, and Laurie Williams. On Coverage-Based Attack Profiles. In *Software Security and Reliability (SERE), 2014 IEEE 8th International Conference on*, June 2014.
- [92] Thomas P Ryan. *Modern engineering statistics*. John Wiley & Sons, 2007.
- [93] Seth Schoen and Eva Galperin. Iranian Man-in-the-Middle Attack Against Google Demonstrates Dangerous Weakness of Certificate Authorities. *Electronic Frontier Foundation*, August, 2011.
- [94] Mary Shaw. When Is 'Good' Enough?: Evaluating and Selecting Software Metrics. *Software Metrics: An Analysis and Evaluation*, 5:251, 1981.
- [95] Huzaifa S. Sidhpurwala. Bug 752703 - CVE-2011-4128 gnutls: possible DoS due to buffer overflow. [https://bugzilla.redhat.com/show\\_bug.cgi?id=752703](https://bugzilla.redhat.com/show_bug.cgi?id=752703), November 2011.
- [96] Huzaifa S. Sidhpurwala. Bug 790737 - CVE-2011-3026 libpng: Heap buffer overflow in png\_decompress\_chunk. [https://bugzilla.redhat.com/show\\_bug.cgi?id=790737](https://bugzilla.redhat.com/show_bug.cgi?id=790737), February 2012.
- [97] Huzaifa S. Sidhpurwala. Bug 1084875 - CVE-2014-0160 openssl: information disclosure in handling of TLS heartbeat extension packets. [https://bugzilla.redhat.com/show\\_bug.cgi?id=1084875](https://bugzilla.redhat.com/show_bug.cgi?id=1084875), April 2014.
- [98] stas. fix UMR in php\_register\_variable\_ex, reported by Stefan Esser. [http://svn.php.net/viewvc/php/php-src/trunk/main/php\\_variables.c?r1=321827&r2=323007&pathrev=323007](http://svn.php.net/viewvc/php/php-src/trunk/main/php_variables.c?r1=321827&r2=323007&pathrev=323007), Feb 2012.
- [99] S. Subramani, M. Vouk, and L. Williams. Non-operational testing of software for security issues. In *Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium on*, pages 21–22, Nov 2013.
- [100] Rahul Sundaram. Red Hat Enterprise Linux. [https://fedoraproject.org/wiki/Red\\_Hat\\_Enterprise\\_Linux](https://fedoraproject.org/wiki/Red_Hat_Enterprise_Linux).

- [101] Eugene Teo. Bug 712774 - CVE-2011-2203 kernel: hfs\_find\_init() sb->ext\_tree NULL pointer dereference. [https://bugzilla.redhat.com/show\\_bug.cgi?id=712774](https://bugzilla.redhat.com/show_bug.cgi?id=712774), May 2014.
- [102] M Vouk and Anthony T Rivers. Construction of reliable software in resource-constrained environments. *Case Studies in Reliability and Maintenance*, pages 205–231, 2003.
- [103] MA Vouk. Using reliability models during testing with non-operational profiles. In *Proceedings of the 2nd Bellcore/Purdue workshop on issues in Software Reliability Estimation*, pages 103–111, 1992.
- [104] Mladen A Vouk and Kuo-Chung Tai. Some issues in multi-phase software reliability modeling. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering-Volume 1*, pages 513–523. IBM Press, 1993.
- [105] Mladen A Vouk and Laurie Williams. An Investigation of Scientific Principles Involved in Software Security Engineering. March 2012.
- [106] Shigeru Yamada, Mitsuru Ohba, and S. Osaki. S-Shaped Software Reliability Growth Models and Their Applications. *Reliability, IEEE Transactions on*, R-33(4):289–292, Oct 1984.
- [107] Shigeru Yamada, Mitsuru Ohba, and Shunji Osaki. S-Shaped Reliability Growth Modeling for Software Error Detection. *Reliability, IEEE Transactions on*, R-32(5):475–484, Dec 1983.

## APPENDICES

### A.1 Fedora Operational Usage Weekly Problem Reports

Figure A.1, Figure A.2 and Figure A.3 shows the history of Fedora non-security and security problem report rates for over some 40+ calendar weeks. Note that the vertical axis is logarithmic.

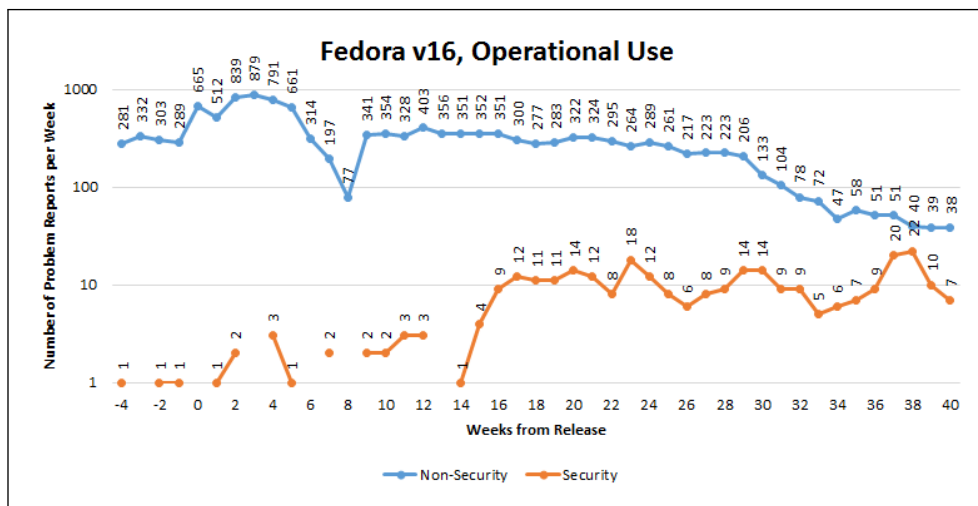


Figure A.1: Fedora v16 Weekly Problem Report Rates Under Operational Usage Over Calendar Time

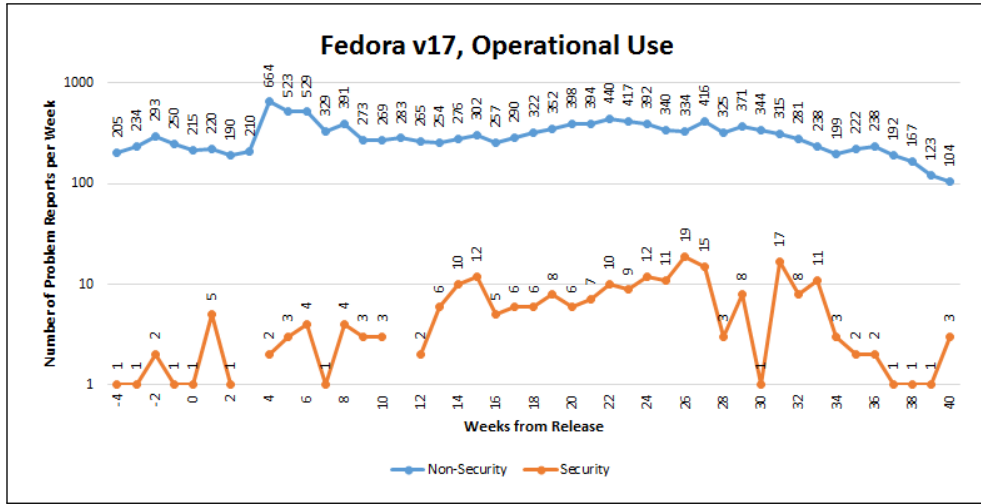


Figure A.2: Fedora v16 Weekly Problem Report Rates Under Operational Usage Over Calendar Time

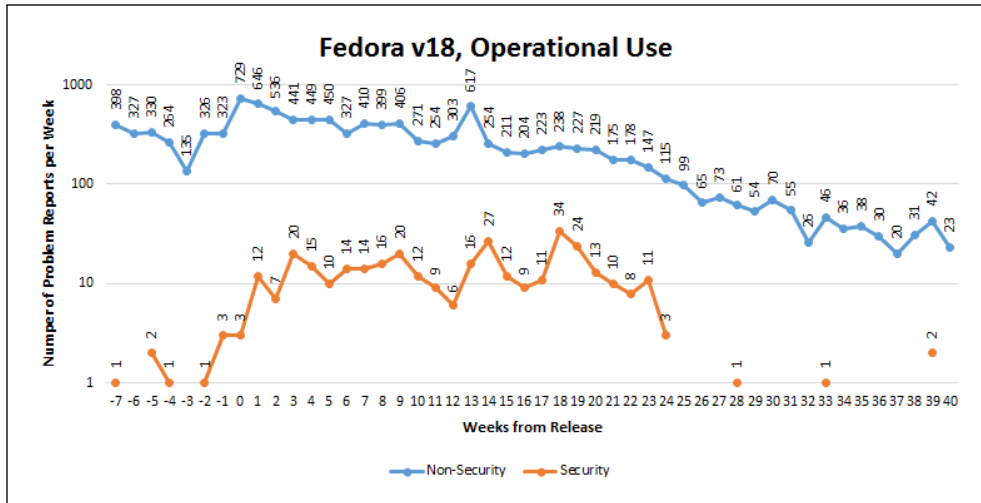


Figure A.3: Fedora v17 Weekly Problem Report Rates Under Operational Usage Over Calendar Time

## A.2 Categorical Distribution Results for Security Problem Reports in Fedora

Table A.1: Fedora v15 Security Problems Categorical Distribution

	Category of Problems	Epistemic			Border	Aleatoric	Unclassified
		Top 25	Design Flaws	Incomplete Analysis			
Substrate	Fedora-Core	5	3	2	1	0	0
	Fedora-Application	30	5	2	4	6	2

Table A.2: Fedora v16 Security Problems Categorical Distribution

	Category of Problems	Epistemic			Border	Aleatoric	Unclassified
		Top 25	Design Flaws	Incomplete Analysis			
Substrate	Fedora-Core	1	1	3	1	0	1
	Fedora-Application	27	3	11	2	3	7

Table A.3: Fedora v17 Security Problems Categorical Distribution

	Category of Problems	Epistemic			Border	Aleatoric	Unclassified
		Top 25	Design Flaws	Incomplete Analysis			
Substrate	Fedora-Core	2	0	0	3	0	0
	Fedora-Application	23	6	7	9	5	5

Table A.4: Fedora v18 Security Problems Categorical Distribution

	Category of Problems	Epistemic			Border	Aleatoric	Unclassified
		Top 25	Design Flaws	Incomplete Analysis			
Substrate	Fedora-Core	5	0	3	1	0	0
	Fedora-Application	29	4	3	12	2	1

### A.3 Time Distribution Results for Security Problem Reports in Fedora

Table A.5: Fedora v15 Security Problems Time Distribution

Category of Security Problems		Substrate			
		Core		Applications	
		1st 20 Weeks	Last 20 Weeks	1st 20 Weeks	Last 20 Weeks
Epistemic	Top 25	0	5	14	16
	Design Flaws	0	3	0	5
	Incomplete Analysis	0	2	2	0
Border		0	1	1	3
Aleatoric		0	0	5	1
Unclassified		0	0	1	1

Table A.6: Fedora v16 Security Problems Time Distribution

Category of Security Problems		Substrate			
		Core		Applications	
		1st 20 Weeks	Last 20 Weeks	1st 20 Weeks	Last 20 Weeks
Epistemic	Top 25	0	1	10	17
	Design Flaws	0	1	1	2
	Incomplete Analysis	0	3	6	5
Border		0	1	0	2
Aleatoric		0	0	0	3
Unclassified		1	0	0	7

Table A.7: Fedora v17 Security Problems Time Distribution

Category of Security Problems		Substrate			
		Core		Applications	
		1st 20 Weeks	Last 20 Weeks	1st 20 Weeks	Last 20 Weeks
Epistemic	Top 25	0	2	8	15
	Design Flaws	0	0	2	4
	Incomplete Analysis	0	3	3	4
Border		0	3	5	4
Aleatoric		0	0	3	2
Unclassified		0	0	3	2

Table A.8: Fedora v18 Security Problems Time Distribution

		Substrate			
		Core		Applications	
Category of Security Problems		1st 20 Weeks	Last 20 Weeks	1st 20 Weeks	Last 20 Weeks
Epistemic	Top 25	2	2	29	1
	Design Flaws	0	0	3	1
	Incomplete Analysis	3	0	2	1
Border		1	0	9	3
Aleatoric		0	0	2	0
Unclassified		0	0	1	0

---

CASRE Input Data Format

---

**B.1 Input Data Format**

For the Yamada S-Shaped model, the CASRE tool accepts input having the following format:  
 [1] Test Interval # ; [2] Number of Failures in Test Interval ; [3] Length of Test Interval ; [4] Program Fraction Tested ; [5] Last Interval to Include ; [6] Fraction of Cumulative Failures ; [7] Severity

Since we are dealing with field usage data under normal operational profile, we fill in the columns 4 and 6 by 1.0. Severity classes i.e. column 7 is also set to 1.

Table B.1, Table B.1, Table B.1 and Table B.1 give the input files used for analysis (using the CASRE tool) of Fedora 15, 16 17 and 18 security data respectively.

Table B.1: Fedora 15 Prepared Data (Security Problems)- fv15.dat

Weeks						
1	3	1	1	0	1	1
2	2	1	1	-1	1	1
3	8	1	1	-1	1	1
4	0	1	1	-1	1	1
5	2	1	1	-1	1	1
6	4	1	1	-1	1	1
7	5	1	1	-1	1	1
8	4	1	1	-1	1	1

*Continued on next page*

Table B.1 – *Continued from previous page*

<b>Weeks</b>						
9	4	1	1	-1	1	1
10	5	1	1	-1	1	1
11	5	1	1	-1	1	1
12	4	1	1	-1	1	1
13	7	1	1	-1	1	1
14	12	1	1	-1	1	1
15	2	1	1	-1	1	1
16	4	1	1	-1	1	1
17	10	1	1	-1	1	1
18	3	1	1	-1	1	1
19	19	1	1	-1	1	1
20	9	1	1	-1	1	1
21	8	1	1	-1	1	1
22	3	1	1	-1	1	1
23	26	1	1	-1	1	1
24	20	1	1	-1	1	1
25	6	1	1	-1	1	1
26	7	1	1	-1	1	1
27	7	1	1	-1	1	1
28	7	1	1	-1	1	1
29	5	1	1	-1	1	1
30	8	1	1	-1	1	1
31	5	1	1	-1	1	1
32	5	1	1	-1	1	1
33	2	1	1	-1	1	1
34	1	1	1	-1	1	1
35	6	1	1	-1	1	1
36	10	1	1	-1	1	1
37	21	1	1	-1	1	1
38	5	1	1	-1	1	1
39	5	1	1	-1	1	1
40	6	1	1	-1	1	1

Table B.2: Fedora 16 Prepared Data (Security Problems)- fv16.dat

Weeks						
1	0	1	1	0	1	1
2	1	1	1	-1	1	1
3	2	1	1	-1	1	1
4	0	1	1	-1	1	1
5	3	1	1	-1	1	1
6	1	1	1	-1	1	1
7	0	1	1	-1	1	1
8	2	1	1	-1	1	1
9	0	1	1	-1	1	1
10	2	1	1	-1	1	1
11	2	1	1	-1	1	1
12	3	1	1	-1	1	1
13	3	1	1	-1	1	1
14	0	1	1	-1	1	1
15	1	1	1	-1	1	1
16	4	1	1	-1	1	1
17	9	1	1	-1	1	1
18	12	1	1	-1	1	1
19	11	1	1	-1	1	1
20	11	1	1	-1	1	1
21	14	1	1	-1	1	1
22	12	1	1	-1	1	1
23	8	1	1	-1	1	1
24	18	1	1	-1	1	1
25	12	1	1	-1	1	1
26	8	1	1	-1	1	1
27	6	1	1	-1	1	1
28	8	1	1	-1	1	1
29	9	1	1	-1	1	1
30	14	1	1	-1	1	1
31	14	1	1	-1	1	1
32	9	1	1	-1	1	1

*Continued on next page*

Table B.2 – *Continued from previous page*

<b>Weeks</b>						
33	9	1	1	-1	1	1
34	5	1	1	-1	1	1
35	6	1	1	-1	1	1
36	7	1	1	-1	1	1
37	9	1	1	-1	1	1
38	20	1	1	-1	1	1
39	22	1	1	-1	1	1
40	10	1	1	-1	1	1
41	7	1	1	-1	1	1
42	9	1	1	-1	1	1
43	7	1	1	-1	1	1
44	8	1	1	-1	1	1
45	3	1	1	-1	1	1
46	4	1	1	-1	1	1
47	0	1	1	-1	1	1
48	0	1	1	-1	1	1
49	0	1	1	-1	1	1
50	0	1	1	-1	1	1
51	0	1	1	-1	1	1
52	0	1	1	-1	1	1
53	0	1	1	-1	1	1
54	2	1	1	-1	1	1
55	0	1	1	-1	1	1
56	0	1	1	-1	1	1
57	2	1	1	-1	1	1
58	1	1	1	-1	1	1
59	1	1	1	-1	1	1
60	0	1	1	-1	1	1
61	1	1	1	-1	1	1

Table B.3: Fedora 18 Prepared Data (Security Problems)- fv18.dat

Weeks						
1	3	1.0	1.0	0	1.0	1
2	5	1.0	1.0	-1	1.0	1
3	1	1.0	1.0	-1	1.0	1
4	0	1.0	1.0	-1	1.0	1
5	2	1.0	1.0	-1	1.0	1
6	3	1.0	1.0	-1	1.0	1
7	4	1.0	1.0	-1	1.0	1
8	1	1.0	1.0	-1	1.0	1
9	4	1.0	1.0	-1	1.0	1
10	3	1.0	1.0	-1	1.0	1
11	3	1.0	1.0	-1	1.0	1
12	0	1.0	1.0	-1	1.0	1
13	2	1.0	1.0	-1	1.0	1
14	6	1.0	1.0	-1	1.0	1
15	10	1.0	1.0	-1	1.0	1
16	12	1.0	1.0	-1	1.0	1
17	5	1.0	1.0	-1	1.0	1
18	6	1.0	1.0	-1	1.0	1
19	6	1.0	1.0	-1	1.0	1
20	8	1.0	1.0	-1	1.0	1
21	6	1.0	1.0	-1	1.0	1
22	7	1.0	1.0	-1	1.0	1
23	10	1.0	1.0	-1	1.0	1
24	9	1.0	1.0	-1	1.0	1
25	12	1.0	1.0	-1	1.0	1
26	11	1.0	1.0	-1	1.0	1
27	19	1.0	1.0	-1	1.0	1
28	15	1.0	1.0	-1	1.0	1
29	3	1.0	1.0	-1	1.0	1
30	8	1.0	1.0	-1	1.0	1
31	1	1.0	1.0	-1	1.0	1
32	17	1.0	1.0	-1	1.0	1

*Continued on next page*

Table B.3 – *Continued from previous page*

<b>Weeks</b>						
33	8	1.0	1.0	-1	1.0	1
34	11	1.0	1.0	-1	1.0	1
35	3	1.0	1.0	-1	1.0	1
36	2	1.0	1.0	-1	1.0	1
37	2	1.0	1.0	-1	1.0	1
38	1	1.0	1.0	-1	1.0	1
39	1	1.0	1.0	-1	1.0	1
40	1	1.0	1.0	-1	1.0	1
41	3	1.0	1.0	-1	1.0	1
42	1	1.0	1.0	-1	1.0	1
43	2	1.0	1.0	-1	1.0	1
44	1	1.0	1.0	-1	1.0	1
45	0	1.0	1.0	-1	1.0	1
46	0	1.0	1.0	-1	1.0	1
47	0	1.0	1.0	-1	1.0	1
48	0	1.0	1.0	-1	1.0	1
49	0	1.0	1.0	-1	1.0	1
50	0	1.0	1.0	-1	1.0	1
51	0	1.0	1.0	-1	1.0	1
52	0	1.0	1.0	-1	1.0	1
53	0	1.0	1.0	-1	1.0	1
54	1	1.0	1.0	-1	1.0	1
55	0	1.0	1.0	-1	1.0	1
56	0	1.0	1.0	-1	1.0	1
57	0	1.0	1.0	-1	1.0	1
58	0	1.0	1.0	-1	1.0	1
59	1	1.0	1.0	-1	1.0	1
60	1	1.0	1.0	-1	1.0	1
61	0	1.0	1.0	-1	1.0	1

Table B.4: Fedora 18 prepared Data (Security Problems)- fv18.dat

Weeks						
1	3	1.0	1.0	0	1.0	1
2	12	1.0	1.0	-1	1.0	1
3	7	1.0	1.0	-1	1.0	1
4	20	1.0	1.0	-1	1.0	1
5	15	1.0	1.0	-1	1.0	1
6	10	1.0	1.0	-1	1.0	1
7	14	1.0	1.0	-1	1.0	1
8	14	1.0	1.0	-1	1.0	1
9	16	1.0	1.0	-1	1.0	1
10	20	1.0	1.0	-1	1.0	1
11	12	1.0	1.0	-1	1.0	1
12	9	1.0	1.0	-1	1.0	1
13	6	1.0	1.0	-1	1.0	1
14	16	1.0	1.0	-1	1.0	1
15	27	1.0	1.0	-1	1.0	1
16	12	1.0	1.0	-1	1.0	1
17	9	1.0	1.0	-1	1.0	1
18	11	1.0	1.0	-1	1.0	1
19	34	1.0	1.0	-1	1.0	1
20	24	1.0	1.0	-1	1.0	1
21	13	1.0	1.0	-1	1.0	1
22	10	1.0	1.0	-1	1.0	1
23	8	1.0	1.0	-1	1.0	1
24	11	1.0	1.0	-1	1.0	1
25	3	1.0	1.0	-1	1.0	1
26	0	1.0	1.0	-1	1.0	1
27	0	1.0	1.0	-1	1.0	1
28	0	1.0	1.0	-1	1.0	1
29	1	1.0	1.0	-1	1.0	1
30	0	1.0	1.0	-1	1.0	1
31	0	1.0	1.0	-1	1.0	1
32	0	1.0	1.0	-1	1.0	1

*Continued on next page*

Table B.4 – *Continued from previous page*

<b>Weeks</b>						
33	0	1.0	1.0	-1	1.0	1
34	1	1.0	1.0	-1	1.0	1
35	0	1.0	1.0	-1	1.0	1
36	0	1.0	1.0	-1	1.0	1
37	0	1.0	1.0	-1	1.0	1
38	0	1.0	1.0	-1	1.0	1
39	0	1.0	1.0	-1	1.0	1
40	2	1.0	1.0	-1	1.0	1