Software Engineering Institute

Data-Driven Software Assurance: A Research Study

Mike Konrad
Art Manion
Andrew Moore
Julia Mullaney
William Nichols
Michael Orlando
Erin Harper

May 2014

TECHNICAL REPORT
CMU/SEI-2014-TR-010

Software Solutions Division

http://www.sei.cmu.edu

Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This report was prepared for the
SEI Administrative Agent
AFLCMC/PZM
20 Schilling Circle, Bldg 1305, 3rd floor
Hanscom AFB, MA 01731-2125

DM-0001039

Table of Contents

List of Figures

List of Tables

Acknowledgments

The authors of this report thank Julia Allen, Rich Caralli, Joji Montelibano, and Dave Zubrow for providing inspiration and encouragement for this research project.

Abstract

Software vulnerabilities are defects or weaknesses in a software system that if exploited can lead to compromise of the control of a system or the information it contains. The problem of vulnerabilities in fielded software is pervasive and serious. In 2012, Software Engineering Institute (SEI) researchers began investigating vulnerabilities reported to the SEI's CERT® Division and determined that a large number of significant and pernicious software vulnerabilities likely had their origins early in the software development life cycle, in the requirements and design phases. A research project was launched to investigate design-related vulnerabilities and quantify their effects. The Data-Driven Software Assurance project examined the origins of design vulnerabilities, their mitigations, and the resulting economic implications. Stage 1 of the project included three phases: 1) conduct of a mapping study and literature review, 2) conduct of detailed vulnerability analyses, and 3) development of an initial economic model. The results of Stage 1 indicate that a broader initial focus on secure design yields substantial benefits to both the developer and operational communities and point to ways to intervene in the software development life cycle (or operations) to mitigate vulnerabilities and their impacts. This report describes Stage 1 activities and outlines plans for follow-on work in Stage 2.

1 Introduction

Software vulnerabilities are defects or weaknesses in a software system that if exploited can lead to compromise of system control or the information a system contains. The problem of vulnerabilities in fielded software is pervasive and serious. A recent report from the U.S. General Accounting Office (GAO) shows that operational vulnerabilities have increased 780% over the past six years [GAO 2013].

A considerable amount of work has been done on various aspects of the vulnerability issue, but problems persist and new ones continue to emerge. Much of the research conducted so far has been somewhat narrow in focus. Secure coding standards, for example, have had high operational validity, but the solution is narrowly focused on the coding phase. Moreover, the knowledge gained through research that could prevent many types of vulnerabilities is significantly lagging in application. This gap is symptomatic of a focus on reductive approaches instead of investigations that use a systems perspective—that is, the focus is often on fixing defects once they are found in operational software instead of examining the entire software development lifecycle to prevent them from ever occurring.

In 2012, Software Engineering Institute (SEI) researchers began investigating vulnerabilities reported to the SEI's CERT® Division, which included more than 40,000 cases, and determined that a large number of significant, pernicious, and infamous software vulnerabilities likely had their origins early in the life cycle, in the requirements and design phases. A research project was designed to investigate design-related vulnerabilities and quantify their effects. The Data-Driven Software Assurance (DDSA) project was launched to examine the origins of design vulnerabilities, their mitigation, and the resulting economic implications.

The DDSA project bridged two distinct software communities—software developers and users of operational software—to facilitate research with a more broadly focused systems perspective aimed at identifying root causes instead of focusing on proximate cause. A bridge between these communities is critical for systems-based research because they are quite dissimilar in motivation and have different views of mitigation. The SEI designed the DDSA project as a joint effort between two of its research programs: the Software Engineering Process Management (SEPM) Initiative and the CERT Division. The SEPM Initiative provided data and expertise from the software development community, while the CERT Division provided data and expertise gained through interaction with the software user community.

The intent of DDSA was to align the concerns and decisions made during development with those in operations by better understanding how decisions made earlier in the software development life cycle affect the level of vulnerabilities released into production, where defects increase sustainment costs and risk to the mission.

The first stage of this research, Stage 1, has been completed, and the results are summarized in this report. A proposed follow-on research project, Stage 2, is also described.

® CERT® is a registered mark owned by Carnegie Mellon University.

2 Project Overview

2.1 Problem Description

Vulnerabilities in software affect everyone and are increasing at an alarming rate. Figure 1 shows the results of the analysis of US-CERT data: a 780% increase in incidents over the past six years. Examples of exploited vulnerabilities are abundant in news reports, and vulnerabilities have far-reaching effects. For example, many people experienced credit card recalls due to compromised user data from large companies, including TJ Maxx and Citibank.[1] In 2012, hackers breached a server at the Utah Department of Health and accessed thousands of Medicaid records—about 280,000 people had their Social Security numbers exposed, and another 350,000 people had sensitive data stolen, including names, birth dates, and addresses.[2] Sony suffered a massive breach in its video game online network that led to the theft of personal information, including the names, addresses, and possibly credit card data of 77 million user accounts.[3]

These kinds of problems are not unique to commerce. They exist everywhere computers are used and are worse where computers are connected to a network, including in the U.S. Department of Defense (DoD) and throughout key government agencies.

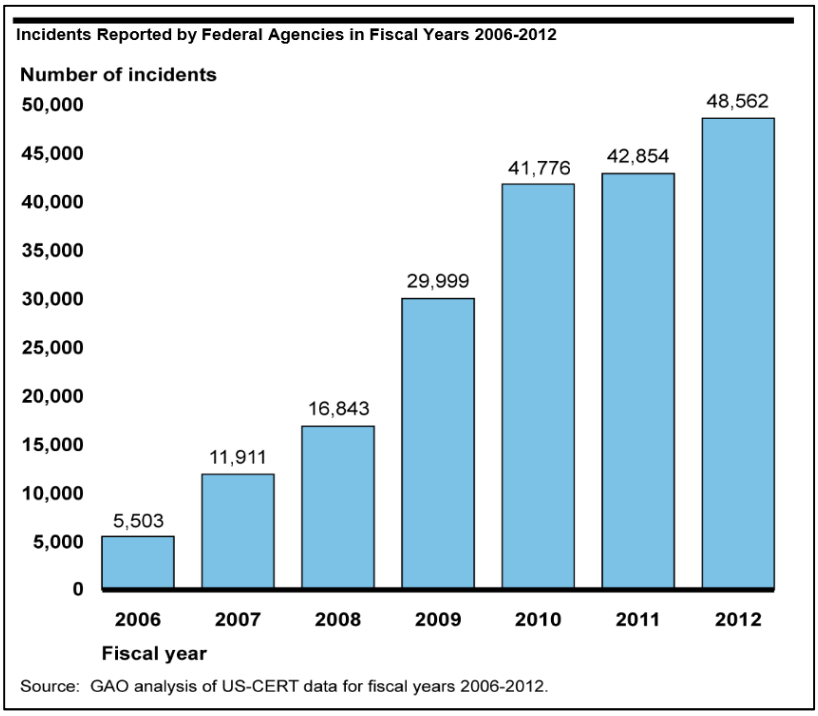


Figure 1: Number of Incidents Annually

[1] http://www.computerweekly.com/news/1280090525/TJMaxx-hacker-linked-to-Citibank-hack
[2] http://www.esecurityplanet.com/hackers/hackers-steal-thousands-of-medicaid-records.html
[3] http://www.reuters.com/article/2011/04/26/us-sony-stoldendata-idUSTRE73P6WB20110426

The economic consequences of vulnerabilities can be divided into two general types:

1. Harm caused – breaches are costly, causing mission failures, the theft of resources, and loss of security.
2. Fixing the problem – patching known vulnerabilities is expensive and the financial costs are growing. Growth trends show that unless we do something, there will not be sufficient skilled staff to even deploy needed patches in the future.

Coding-related vulnerabilities are preventable and detectable, but less attention has been paid to vulnerabilities arising from requirements and design defects. The DDSA project was charged with addressing this problem by identifying root causes of vulnerabilities in the requirements and design phases of the software development life cycle and introducing cost-effective changes in how software is built and validated.

The software development and operational software user communities are frequently disconnected from each other—often deliberately—by arms-length relationships and have many differences, represented in Table 1. Therefore, the DDSA project was staffed with representatives from both of these communities (found respectively in SEPM and CERT) to connect known software vulnerabilities to actionable practices early in the software development life cycle (SDLC).

Table 1: Differences in Two Software Communities

Community	**Software Development**	**Operations**
Motivation	Deliver complete product on time	The mission
Disruptions	Defects	Vulnerabilities
Costs of disruption	Rework, delay	Mission failure, cost of patching, diversion of attention and resources
Means to address disruptions	Process (prevention, early defect removal, testing, etc.)	Workarounds, patches
SEI Expertise	SEPM	CERT
Related SEI Data	Team Software Process (TSP) data	CERT Vulnerability Analysis data

Figure 2 illustrates how these two communities are currently connected (shown in black text next to arrows) vs. how they would ideally be connected (shown in light blue text next to arrows). The red dollar signs represent the costs of fixing issues at various stages in the software development life cycle. Note that smaller amounts of investment early in the process (represented by the size of the dollar sign) can prevent the largest cost: responding to exploits in operational software. Furthermore, some vulnerabilities can never be fully resolved because of their pervasiveness in large-scale releases of operational software (e.g., some early versions of Windows are still running unpatched). The two blue-dashed ovals in the figure relate developer and operational software user activities to the SEI initiatives having that expertise (SEPM and CERT).

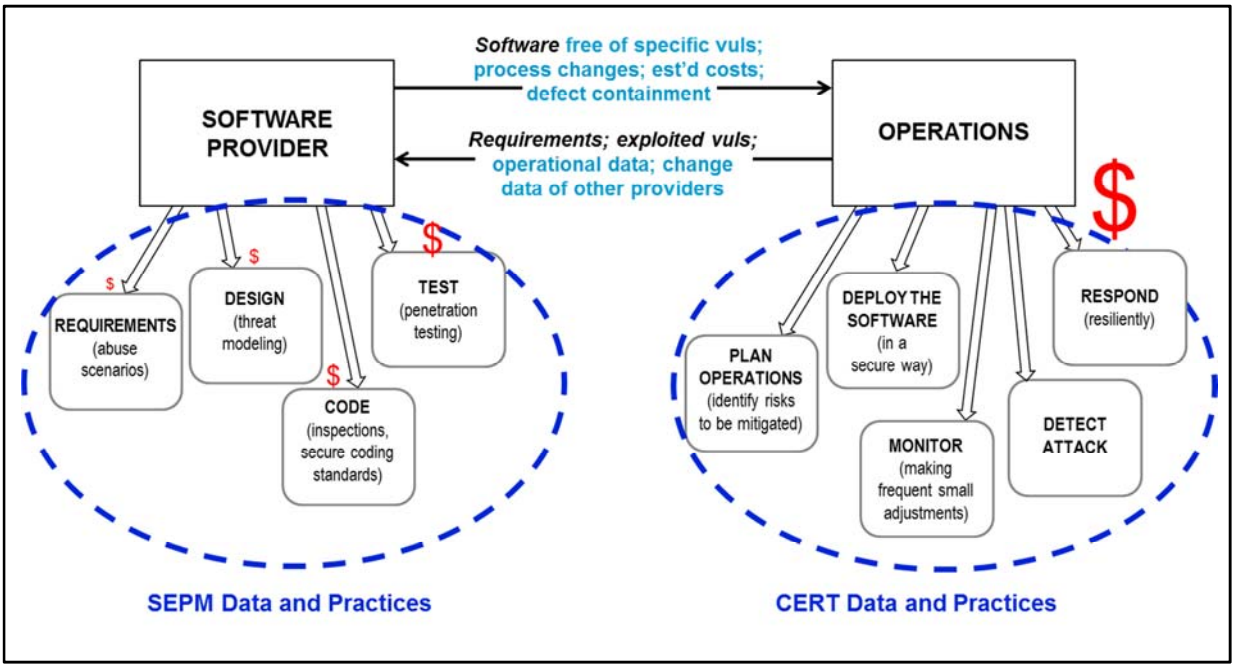


Figure 2: Bridging Development and Operations

2.2 Project Strategy

Table 2 shows the data available from the SEPM and CERT areas. The SEPM program maintains data on best practices and effort and develops models of effort and defect containment, but it has little information about the cost and other impacts of escaped vulnerabilities, which exists to a greater degree in the CERT Division.

Table 2: SEPM and CERT Division Data

SEPM (TSP) Data	CERT Division Data
Approximately 10,000 defect records	Over 40,000 uncategorized vulnerability cases
For each defect: • find and fix effort • phase injected • phase removed • ODC type • an explicit description of the change made • whether or not it was injected while fixing another defect	For each vulnerability: • descriptive title • tracking number • date when reported • quick first-order estimate of how many systems affected (impact) • list of vendors affected • Common Vulnerability Scoring System (CVSS) ratings • URLs with additional information about vulnera-bility, particularly those that are publicly known
Development effort for design	
Effort for design review	

The project identified the actions below to make the best use of the available data.

- Prioritize: Use the CERT vulnerability data as the primary source and prioritize vulnerabilities for further analysis.

- Conduct root cause analysis. Develop or adapt a method to trace vulnerabilities to root causes of defects and potentially actionable changes in the SDLC.
- Select practices: Model the effectiveness of proposed changes in the SDLC to select those that are the most promising to pilot. Use existing TSP process data, effort models, and defect containment models in the construction of change effectiveness models. (We later selected a system dynamics modeling approach.)
- Verify: Pilot and evaluate the effectiveness of the changes through the SDLC. Measure the cost and effectiveness of the specific practices at preventing or containing targeted defects.
- Validate: Close the loop in operations by evaluating how the changes affect the escape of operational vulnerabilities.

3 Orthogonal Defect Classification

One of the first focuses of the DDSA project was to complete an overview and comparison of root cause analysis techniques. Being able to relate vulnerabilities to their design origins was a critical capability for the project. Based on the literature review of root cause analysis, we created an orthogonal defect classification (ODC) system that reached deeply into the origins of a design vulnerability from an ecosystem perspective and that was incorporated into our detailed vulnerability analyses.

3.1 Overview of the IBM ODC

The orthogonal defect classification system created by IBM records attributes of defects as they are discovered and resolved [Chillarege 1992]. The attributes are shown in Figure 3 [IBM 1999]. At discovery, the known attributes include Activity, Trigger, and Impact. Other attributes are typically not known until resolution; these include Target, Type, Qualifier, Source, and Age. These attributes may contain additional levels of detail that vary according to where in the life cycle the defect is discovered or the resolution is applied. Interestingly, ODC includes the activity, but not the life-cycle phase (recall that the project was interested in understanding requirements and design phase-related origins of vulnerabilities).

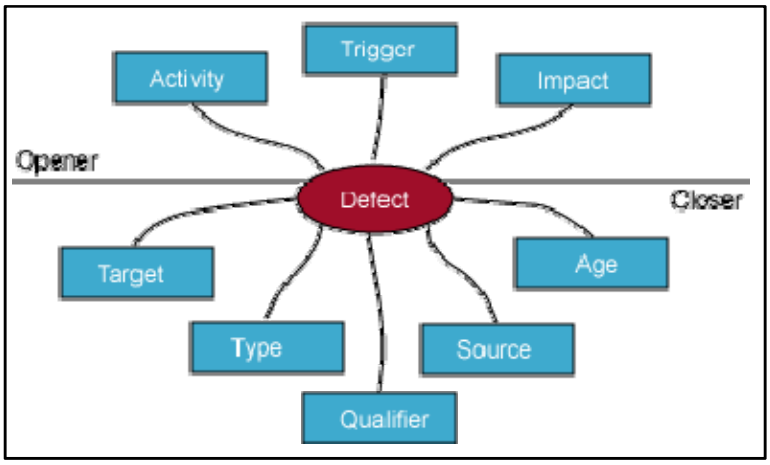


Figure 3: IBM ODC

The attributes in the IBM ODC can be summarized as follows:

1. "Activity" refers to the process step (i.e., code inspection, function test) when defects are discovered.
2. "Trigger" describes the environment or condition that had to exist to expose the defects.
3. "Impact" refers to either a perceived or real impact on users.
4. "Target" represents the high-level identity (e.g., design, code, ID) of the entity that was fixed.
5. "Type" represents the nature of the correction that was made.
6. "Qualifier" specifies whether the fix that was made was due to missing, incorrect, or extraneous code or information.
7. "Source" indicates whether the defect was found in code written in-house, reused from a library, ported from one platform to another, or outsourced to a vendor.
8. "Age" identifies the history of the target that had the defect.

These attributes are further categorized to another level of detail, listed below.

1. Activity
   - Requirements review
   - Requirements inspection
   - Design review
   - Design inspection
   - Code review
   - Code inspection
   - Unit test
   - Integration test
   - Function test
   - System test
   - Acceptance test
   - Product use
2. Trigger
   - Design review/code inspection triggers
     - Design conformance
     - Logic/flow
     - Backward compatibility
     - Internal document
     - Lateral compatibility
     - Concurrency
     - Language dependency
     - Side effect
     - Rare situation
   - Unit test triggers
     - Simple path
     - Complex path
   - Function test triggers
     - Coverage
     - Variation
     - Sequencing
     - Interaction
   - System test/acceptance test/field test triggers
     - Workload stress
     - Recovery/exception
     - Startup/restart
     - Hardware configuration
     - Software configuration
1. Impact
   - Installability
   - Serviceability
   - Standards
   - Integrity/security
   - Migration
   - Reliability
   - Performance
   - Documentation
2. Target
   - Requirements
   - Code
   - Build/package
   - Information development
   - National language support
3. Defect Type
   - Design/code
     - Assign/init
     - Checking
     - Alg/method
     - Func/class/object
     - Timing/serial
     - Interface/O-O messages
     - Relationship
   - Requirements
     - Correctness
     - Completeness
     - Consistency
     - Ambiguity/testability
4. Qualifier
   - Missing
   - Incorrect
   - Extraneous
5. Source
   - Developed in-house
   - Reused from library
   - Outsourced
   - Ported
6. Age
   - Base
   - New
   - Rewritten
   - Refixed

3.2 Overview HP Defect Origins, Types, and Modes

The Hewlett Packard (HP) Defect Origins, Types, and Modes model includes a classification scheme and a practical approach to root cause analysis [Grady 1992]. The Hewlett Packard Software Metrics Council developed the model in 1986 as a standard defect terminology that HP projects and labs could use to report and analyze defects, helping to focus efforts to eliminate defects and their root causes. The model is shown in Figure 4.

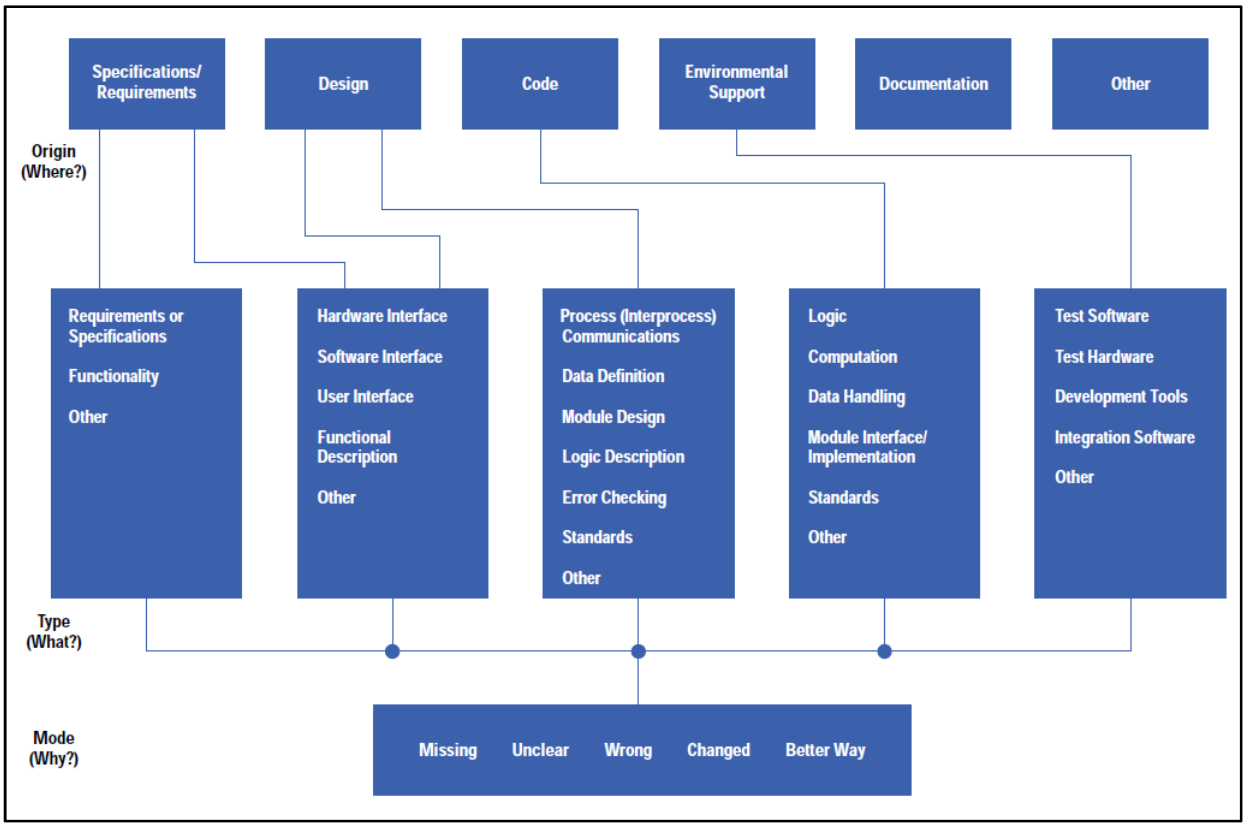


Figure 4: HP Defect Origins, Types, and Modes Model

"Origin" refers to the first activity in the life cycle where the defect could have been prevented, not where it was found. "Type" refers to the area, within a particular origin, that is responsible for the defect. "Mode" is a designator of why the defect occurred (this level of the model was not widely used within HP).

The root cause analysis includes three stages: pre-meeting, meeting, and post-meeting. The actions associated with each stage are listed below.

1. Pre-meeting
   - Identify the division's primary business goal.
   - Have the division champion and root cause facilitator analyze data.
   - Have the champion send out the meeting announcement and instructions to engineers, asking them to do the following:
     a. Pick two defects from their code related to the chosen defect categories.

b. Think of ways to prevent or find defects sooner.

2. Meeting
   - State the meeting's goal. Use insights gained from failure analysis data to improve development and support practices.
   - Perform issues selection (10 minutes).
   - Review the defects brought to the meeting (15 minutes).
   - Perform analysis (15 minutes).
   - Take a break (10 minutes).
   - Brainstorm solutions (10 minutes).
   - Test for commitment (10 minutes).
   - Plan for change (10 minutes).
3. Post-meeting
   - Have the division champion and root cause facilitator review meeting process.
   - Have the division champion capture software development process baseline data.

The process can be summarized as follows:

- Categorize defects using the HP model.
- Choose specific, high-risk defect areas as targets for process improvement.
- Engage in root cause analysis to determine the causes of these categories of defects.
- Generate a defect prevention plan, with action items, for major defect areas.
- Apply the Software Testing Focus model as described to determine testing focus for the next project.
- Determine action plans and assign owners.
- Execute the plans during the next project.
- At the end of the next project, categorize again and re-evaluate the defect trends.
- Document changes.

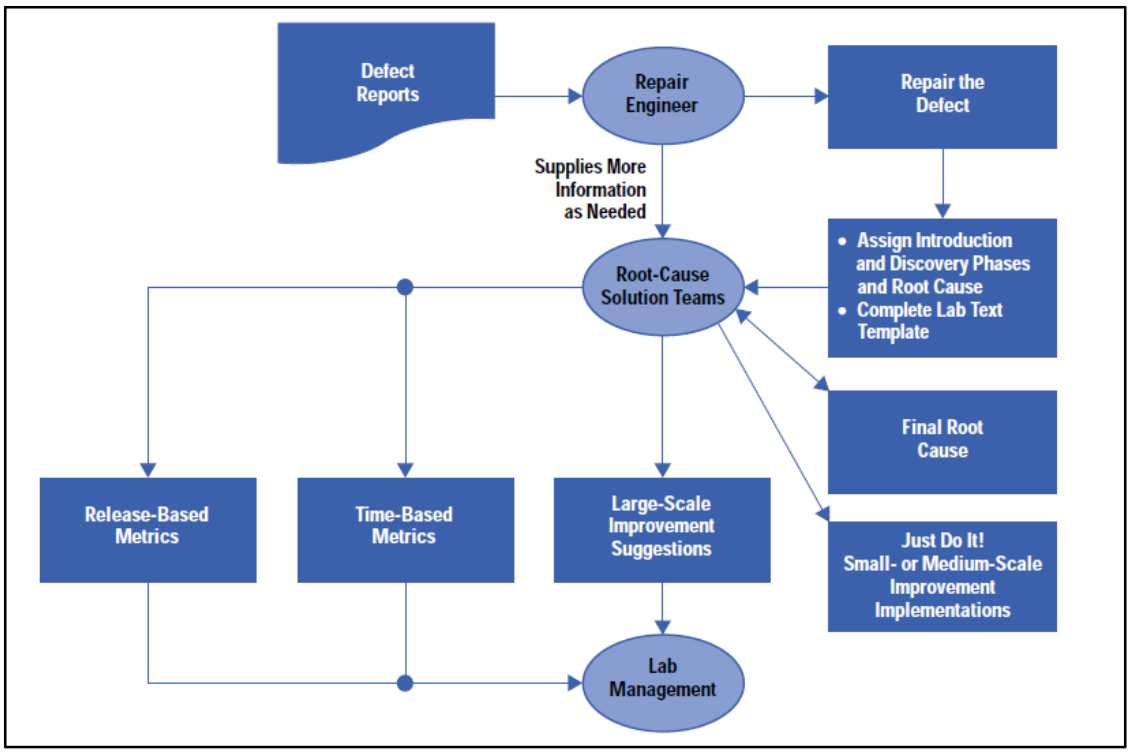


Figure 5: Root Cause Analysis Process

3.3 Evaluation of the Models

The IBM approach provides a thorough analysis of defect causes and effects throughout the life cycle. However, it requires

- substantial training
- substantial administrative and technical support
- substantial planning to implement
- the entire life cycle to be accessible to the investigators before it can be fully applied

Although this approach can be highly effective, it was not practical for the DDSA project given the limited resources and access.

The IBM ODC model may be more appropriate for an organization that has

- a dedicated resource
- engineers who are convinced that defect analysis will benefit their project
- sponsors willing to invest in the effort it takes to implement IBM ODC successfully throughout the SDLC

The HP model has several benefits, described below.

- The process for using the HP model is well documented and intuitive.

- The model is specifically designed to maximize the possibility for process improvement and incremental change. The HP model can be used by engineers who have never done defect analysis and taught in a short workshop, since each step in the process clearly suggests the next step. With a modest investment, it is possible to get an idea about defect trends and identify focus areas for future improvement.

These notable omissions may be observed from the IBM ODC approach:

- Design and architecture are not considered targets.
- Beta test is not listed as an activity.
- Type is focused on the design and code activities, with no explicit types for requirements. Moreover, design does not distinguish between detailed implementation designs and higher level design concerns such as architecture.
- Remediation cost is not considered.
- Impact in terms of cost to the customer is not considered.

A primary focus of the HP model is that all defects are not created equal. Therefore, the HP process puts emphasis on defect prevention and early defect detection in the front end of development (requirements and design). The emphasis is on what can be done to improve and better engineer the software development and testing in the future.

3.4 Conclusions and Recommendations

The IBM ODC model is designed for use throughout the life cycle of a product, while the HP model is focused on the requirements and design phases. The HP approach is aligned with the DDSA life-cycle analysis, but does not serve all of our project goals.

For our work, we determined that triggers and impacts are essential to the economic analysis, and we decided to identify the Target, Type, and Qualifier. Gathering Type information was important because there were a number of defects from different sources. Type was a way to categorize and sort defects as we looked for commonality. In the future, we may need to adapt to a Type taxonomy better suited to early life-cycle phases.

We included the following attributes in our ODC for our detailed analysis of selected vulnerabilities:

- **Impact** – used in the sense of the defect "symptom." Our focus was on defects leading to security vulnerabilities. This was used to determine the operational cost of the defect.
- **Operational cost** – included the operational cost of the defect, including patch deployment, down time, opportunity cost, or other damages. This cost is external to the development.
- **Source** – provided some insight into the origin and frequency of the defect/vulnerability prior to development. "Development" was used to mean coding to implement a software design and subsequent testing.
- **Origin** – used in the sense of a phase where a defect might have been recognized or prevented.
- **Activity** – indicated what work was being performed when the defect entered the system.
- **Type** – provided a classification scheme to identify classes of defects that might be addressed by process changes.

- **Target** – what had to be changed. We were concerned with pre-development products including requirements, standards, and preliminary designs. Target could include changes to software, procedures, documents, or standards.
- **Fix cost** – indicated the direct cost of identifying the defect and implementing and deploying a patch or fix. This cost is primarily borne internally by the project or organization.

4 Mapping Study

A systematic mapping study is a study of studies that adds unique knowledge by answering questions that cannot be answered by a single study, or even a subset of studies. Through a comprehensive examination of all published research on a topic, a variety of higher level questions can be explored in a reliable and repeatable manner, such as the identification of strengths, limitations, and gaps in the current research.

Mapping studies generally include these four basic stages:

1. Search – Identify the primary studies that may contain relevant research results.
2. Include/exclude – Select the appropriate primary studies from these after further examination.
3. Assess quality (bias/validity) – Where appropriate, perform a quality assessment of the selected studies.
4. Summarize – Affinitize and summarize results (typically along some dimension of interest).

The mapping study conducted for the DDSA project examined the published literature on the subject of predicting the presence or level of security vulnerabilities in software packages. We surveyed the literature to determine what was known about early indicators of vulnerabilities; that is, what is known when vulnerabilities become visible that might be actionable?

The research questions in the mapping study that we investigated and the motivations for each are listed in Table 3. A discussion of the results is provided in Section 4.3.

Table 3: Mapping Study Research Questions and Motivations

Research Questions	Motivation
Q1. In what forums does the research appear?	Identify the sources a researcher or practitioner should review to remain current in the field and a researcher should consult prior to a systematic literature review.
Q2. How easy is it to identify relevant vulnerability prediction studies in journals and conference proceedings?	Identify shortcomings of internet-based searches to identify software vulnerability prediction papers.
Q3. What research methods have been applied to vulnerability prediction studies?	Identify the scope of validation with respect to the research approaches used. If a particular method has dominated, are there opportunities to apply others productively? Could the data possibly be aggregated in meta-studies?
Q4. What sources of data have been used to develop and validate models?	Identify the scope of validation for the current research; identify gaps. For example, do the studies apply to a variety of different applications or only a few? Open source vs. closed source? Network vs. embedded systems?
Q5. What types of metrics and models have been constructed and tested?	Identify types of models and metrics that have been used, searching for trends as well as opportunities for new directions in the research.
Q6. What kinds of precision and recall have the studies indicated?	Identify the ranges of performance in vulnerability prediction models to determine scope for further study and improvement. Are models performing adequately, or is there need for additional study?
Q7. What are the most influential papers in the field?	Identify the papers and topics considered to be the most important.

4.1 Search and Selection of Publications

The principle criterion for including a journal article or conference presentation in this study was that it directly addressed research in predicting the presence or density of vulnerabilities in software products. Papers focused on defects or faults in general were only included if the paper also specifically predicted cybersecurity vulnerabilities. Papers evaluating methods for finding, identifying, or removing vulnerabilities were only included if they had estimates of vulnerability densities or vulnerable components prior to or after application of the methods. That is, the papers needed to estimate the effectiveness of the methods using metrics such as yield, false positives and false negatives, or precision and recall to be included.

Only papers that used "vulnerability" in the sense of a cybersecurity breach or potential breach were included. Papers that use the term "vulnerability" exclusively in the more general sense were excluded; this excluded all papers that did not expressly address software engineering concerns.

Some papers appeared to report the same or closely related studies in multiple venues. For our purpose these were all included because we did not intend to perform meta-statistical analysis on the data sets. While this choice may bias counts, we provided citation counts to guide others in deciding which papers and topics are of influence or importance.

In May 2013, we conducted a search using SciVerse Scopus for each of the keywords "software," "prediction," and either "vulnerability" or "vulnerabilities" in either the title or body of the text. The subject area excluded unrelated disciplines. A manual review identified individual authors and publication venues not relevant to the study. After these were added to the exclusion criteria, 48 results remained.

The next step of selection involved manually screening the title and abstract. We reasoned that if the primary focus of the paper was vulnerability prediction, it should be discernible from the abstract and title without a detailed reading of the paper. The title and abstract of each paper were examined for relevance using the inclusion and exclusion criteria. The effect was to remove papers that used "vulnerability" in a more general sense, failed to report some sort of prediction performance, or reported on non-software artifacts (e.g., hardware). This reduced the total count to 38 papers.

4.2 Collection and Classification

The data we collected fell into two broad categories: data associated with the publication or author and data associated with the research methods and models. For each of the candidate papers from the initial search, the data listed below were retrieved directly from SCOPUS. For the documents passing the initial cuts, references and citations were also extracted.

- document type (book, article, conference paper)
- conference name, date, and location (if applicable)
- source title, volume, issue
- page start and page end
- article title
- author keywords
- index keywords
- abstract
- link (if available)

Each of the papers was then read and coded using these broad categories:

- Prediction model/tool – described a method or a specific tool
- Requirements – evaluated requirements process or attributes
- Design – evaluated design process or attributes
- Vulnerability type – described the types of vulnerabilities found
- Defect levels – report defect density or prediction of defective components
- Model inputs
  - Static analysis – used static analysis (size, complexity, and so forth)
  - Developer behavior and motivation – used measures of developer's behavior
  - Development metrics – used development metrics (e.g., churn) in the prediction model
  - Component interactions – used component interactions in the prediction model
  - Mixed or novel techniques – used a mixture of inputs or estimates of usage

4.3 Discussion: Research Questions

This section discusses how the collected data address the research questions enumerated in Table 3. We do not include a meta-statistical aggregation because the studies in this report do not use a consistent set of predictor factors and models. The object of a meta-analysis is to apply statistical methods for the purpose of integrating the findings from multiple studies. That is, when considered separately, individual studies may lack statistical power to answer questions such as, "Does X affect Y?" and "By how much?" By aggregating results from similar studies, a meta-analysis might better answer these questions and perhaps address an additional question: "Are there other factors that explain the variation among the studies?" Aggregation, however, is valid only when homogeneous studies supply raw or statistical summary data and when the source of heterogeneity is limited to a well-understood subject partition [Pickard 1998].

The body of research includes many local and speculative ideas about the prediction of vulnerabilities, but until a more coherent and systematic base of research is conducted, our ability to generalize and systematically analyze the results is limited. Therefore, the following discussion of the research questions does not include meta-analysis. Additional detailed results from the mapping study are available in Appendix A.

4.3.1 Research Question 1: Forums

Q1: In what forums does the research appear?

The publications were examined to determine the country of origin, publication category (book, conference, or journal), and the names of the conferences or journals in which they were published. As shown in Figure 6, the majority, 17, of the publications appeared in conference proceedings, followed by 11 in journal articles, and 2 in books.

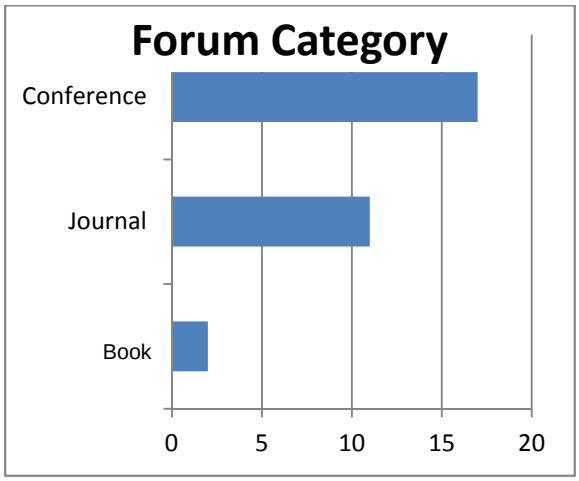


Figure 6: Forum Category: Conference, Journal, or Book

Among the journals, each paper appeared in a different journal. The most notable journals were *IEEE Transactions on Software Engineering* and *Empirical Software Engineering*. Table 6 in Appendix A lists all the journals in which the publications appeared, while Figure 21 shows the publications by country of origin.

Among conferences, only the International Symposium on Software Reliability Engineering and ACM Conference on Computer and Communications Security appeared more than once, with four and three publications, respectively. Nine conferences included only a single relevant paper.

The two major conferences were the forums for about 41% (7 of the 17) conference papers. Only in 2008 did multiple papers appear in the same conference (two each in the International Symposium on Software Reliability Engineering and the ACM Conference on Computer and Communications Security). These two conferences were most active in this subject; however, the publications appear broadly and without concentration. Table 7 in Appendix A lists all the conference and the counts of publications by conference.

Conclusion: No conference or journal can be considered a leading source. The lack of a focusing forum suggests that there may not yet be a research community concentrated on this subject.

4.3.2 Research Question 2: Identifying Studies

Q2: How easy is it to identify relevant vulnerability prediction studies in journals and conference proceedings?

A broad search using the term “vulnerability” has many false finds because the term also occurs in the context of security vulnerability, and the term “vulnerability” may not be used at all in the keywords. Alternate terms included “attack prone,” “thread,” and “risk.”

The keyword “vulnerability prediction” appeared in only seven of the publications, “vulnerability” alone in four, “software security” appeared in three, and “prediction” in two. Variations of “vulnerability” included “vulnerability prediction” (seven), “vulnerability measure” (two), “vulnerabilities” (two), “web application vulnerabilities” (two). In total, “prediction” in some form occurred in only nine of the publications.

Searching only the keywords for "vulnerability" or "prediction" is, therefore, inadequate to find the majority of the relevant publications. Figure 7 shows the frequency of the most common author-defined individual keywords. (Table 8 and Table 9 in Appendix A include the list of keyword phrases that occurred once or twice.)

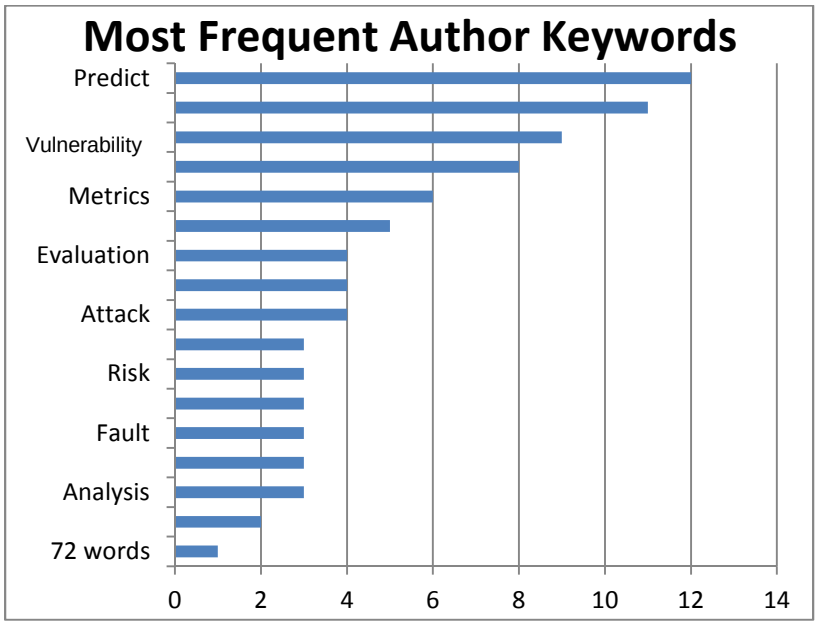


Figure 7: Frequency Count of Author-Defined Individual Keywords

Because "vulnerability" does not occur consistently among the keywords, searches must include the title and abstract, at a minimum. The problem is complicated by the large number of exclusions required in the SCOPUS search because the term "vulnerability" occurs in a number of subjects in a number of contexts. Surprisingly, only 25 of the 30 studies were included in the subject area "computer science" Use of "engineering," "social sciences," "mathematics," or "business, management, and accounting," further compound the search problem (see Figure 8).

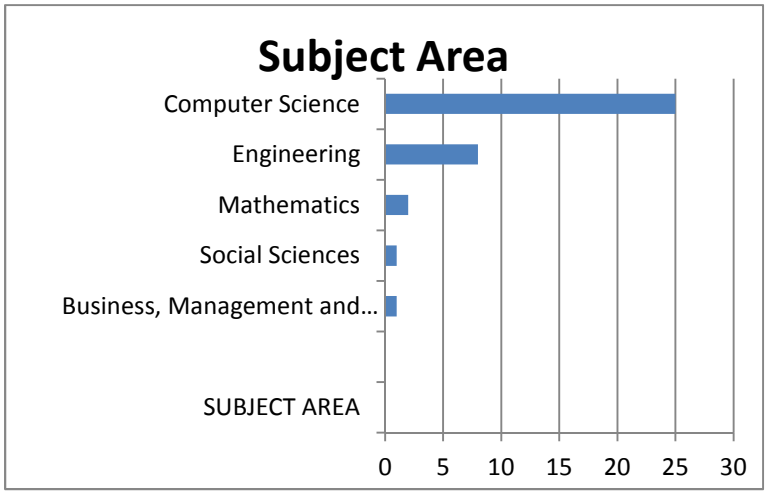


Figure 8: Counts by Subject Area Categories in Database

Conclusion: Anecdotal reports from colleagues suggested that identifying relevant studies is difficult. This mapping study indicates that this is true not only because the publication venues are diverse and the search cannot be concentrated in a few key venues, but also because keywords—an essential means of locating documents relevant to the topic— are not reliable for locating the studies. This indicates that the field lacks a common vocabulary, and that is likely to impede the ability of researchers to build on each other's work. This inference is also supported by the examination of the research question on influential papers in Section 4.3.7.

4.3.3 Research Question 3: Research Methods

Q3: What research methods have been applied to vulnerability prediction studies?

Although the term "experiment" occurs in several papers, the meaning appears to be that a method or tool was applied to a trial set of data. There was no randomization of subjects or use of control groups for comparison. These might be characterized as case studies with the application of quantitative methods. Of the papers in this study, four used formal hypothesis testing, including decision tables (or more commonly, precision and recall) and regression equations with statistical parameters.

Conclusion: Although a variety of approaches have been reported, there is little we can do to meaningfully compare or aggregate the results. To do so, more studies are needed with similar analysis approaches. Consistent and complete reporting of the statistical parameters is also needed to support meta-analysis. For example, precision and recall reports are of limited use without the actual counts or descriptions of how the independent variables are distributed among the components.

4.3.4 Research Question 4: Data and Validation

Q4: What sources of data have been used to develop and validate models?

Validation was performed with a handful of open-source products and a few closed-source commercial products. Among the open-source products, Mozilla Web Browser, Mozilla Javascript Engine, Linux Red Hat Kernel, and APACHE were most common. One study reported using a suite of open-source web applications. Among those using closed-source applications, authors reported using Cisco software system, Windows Vista, Internet Information Services (IIS), an unnamed large mobile application, and T. Three of the studies used the National Vulnerability Database to identify known vulnerabilities.

Conclusion: Open source is popular because it provides an available source of data. The open-source data, however, is of a limited kind, constraining the model constructs. That the same large open-source applications are often studied raises some generalizability questions. While the data sources are diverse, they are also limited in number and in type. Owners of closed-source applications are often reluctant to expose their source code or development process to scrutiny. The data available from these sources appear to constrain the types of models researched.

4.3.5 Research Question 5: Metrics and Models

Q5: What types of metrics and models have been constructed and tested?

Models used included regression, multiple regression, logistic regression decision tree, random forest, and Bayes. One model, SAVI, predicted defect density. One paper examined patterns of input sanitization to SQL. SAVD used static analysis as the predictor with the National Vulnerability Database. One study applied text analytics, another applied data mining and machine learning, while yet another surveyed developers regarding intended use of practices**. See Appendix B: Relevant Documents for more information about the papers referred to in this section.**

Neuhaus' paper examined existing vulnerability data to calibrate a tool, Vulture, that was applied to Mozilla. A key finding was that vulnerability history did not predict future vulnerabilities in the same component, but that similar patterns of imports and procedure calls did, suggesting repeated design implementation problems. Shin 2008 examined correlation between complexity and vulnerability in Mozilla and found weak correlation. Shin 2011 used a broader set of metrics. Zimmerman 2010 also applied a broad set of metrics to analyze Windows Vista, finding high precision but low recall with classic metrics, but dependencies predicted vulnerabilities with low precision and higher recall.

Code size, code or structural complexity, code churn (or change), complexity, and dependency analysis were featured prominently. Also included but less common was the history of defects and vulnerabilities reported post release. One study included measures of developer communication and network centrality.

Conclusion: Vulnerability density as a metric was uncommon in the reports studied. More common was the determination of a module as vulnerable or not vulnerable. Rather than reporting correlation coefficients, the most common results used decision tables, usually as recall and precision although some used true positive, false positive, true negative, and false negative. Vulnerabilities, the dependent variable, were problematic in that they are somewhat rare, including only 5-15% of the defects (in Shin and Williams 2011). Estimates of vulnerability density, however, vary widely and no reports on ranges or correlation between vulnerabilities and defects were included in this mapping study.

4.3.6 Research Question 6: Precision and Recall

Q6: What kinds of precision and recall have the studies indicated?

Studies using conventional metrics typically report precision >0.8 and recall <0.2. The exception was a study employing text analytics that found a much higher recall approaching 0.8.

Conclusion: This suggests that conventional metrics miss a very high portion of the vulnerabilities, but a text analytic approach applied to the source may improve results. This, however, must be considered with some caution because the studies are somewhat different and involve different source code. This is an area where more research may be useful.

4.3.7 Research Question 7: Influential Papers

Q7: What are the most influential papers in the field?

One measure of influence is the citation counts overall and within the group (see Figure 9 and Figure 10). Neuhaus 2007 ("Predicting Vulnerable Software Components") is most cited both overall and within the group, with 33 overall and 10 within the group. Toyssy 2006, which addresses smartphones, is second overall, but is not cited within the group. The within-group citations, shown in Figure 9, suggest a core group of Neuhaus, Shin, Zimmermann, Gegick, Chen, and Ngueyen as the lead authors. Williams appears frequently as a secondary author on the cited papers and first overall in authored papers. **See Appendix B: Relevant Documents for more information about the papers referred to in this section.**

Among this core group, the most cited papers are Neuhaus 2007 and Shin 2000. Zimmerman 2010 and Shin 2011 each received four in-group citations.

Conclusion: Citation analysis has been argued to have some value in measuring quality and to be useful in identifying invisible colleges that form around specialty subjects [Lindsey 1989]. However, citation counts as a measure of influence have been criticized for a many reasons [Macroberts 2010]. These problems include but are not limited to not actually citing the true influences, bias in selection of citations, citation of secondary sources rather than the primary source, the "Matthew Effect" whereby well-known works and authors are over represented, and not citing informal influences (for example discussions between an advisor and a student). The information we gathered should be viewed with these arguments in mind.

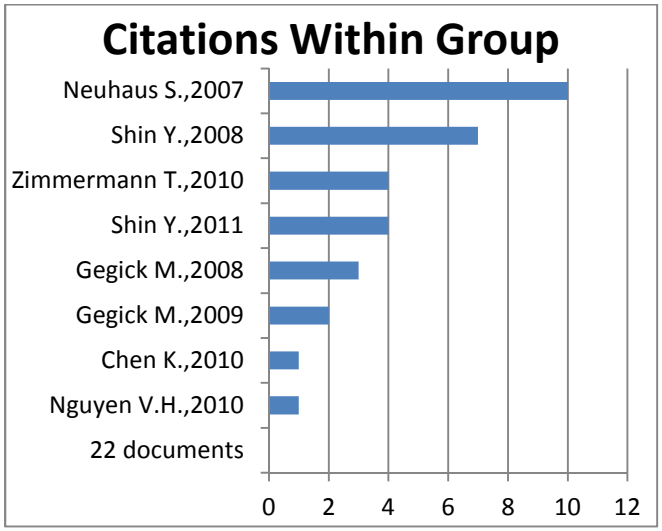


Figure 9: Frequency Count of Publication Citations Within the Group

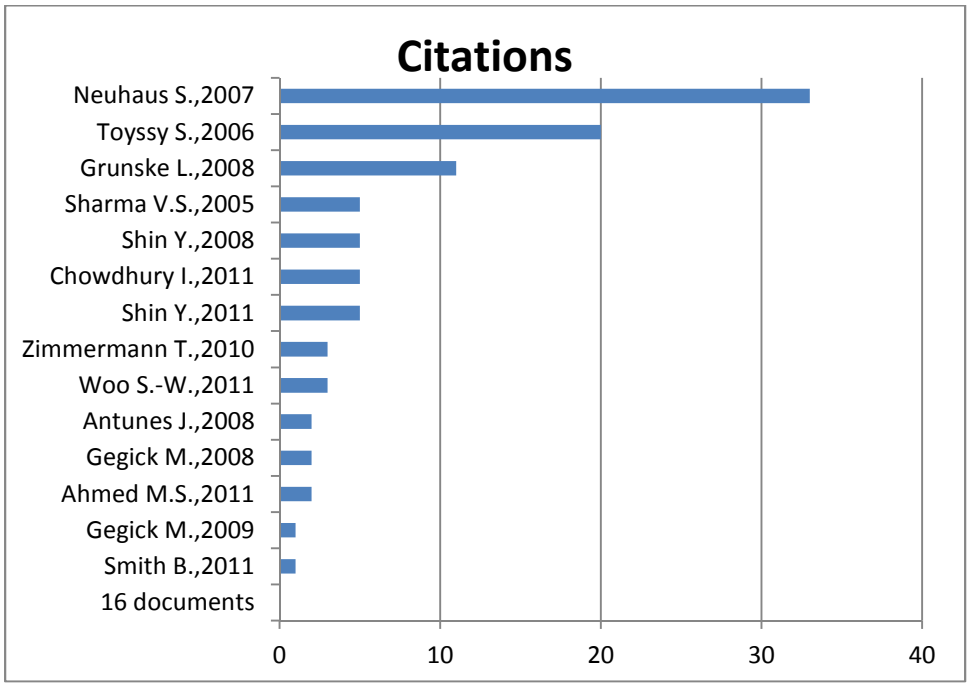


Figure 10: Frequency Count for Publication Citations

4.4 Limitations of the Studies

The limited set of programs for development and verification make drawing generalizations problematic. Moreover, large open-source programs such as Mozilla seem to be preferred. While the use of large open-source packages provides many advantages to the researcher, including a larger supply of similar software modules, a large number of releases for longitudinal studies, a rich development environment from which data can be extracted, and access to the source code, several studies explicitly cautioned that the performance is domain dependent. Therefore, over-reliance on a single package, such as the Mozilla browser, or even a single domain, such as web apps, threatens generalizability.

A related threat shared among the studies is the potential bias to study large, highly used products which have been in use for an extended period of time. The scale of the developments may affect developer behaviors, design, or test conditions, biasing the types of defects remaining in the product after release into production. Moreover, these programs may suffer a survivor bias if they differ in important ways from typical software packages. Another related threat is that the usage of these programs likely differs from smaller programs, biasing the types of vulnerabilities likely to be found and submitted for repair.

4.5 Gaps

The research examined in this mapping study focused almost exclusively on outcome measures, to the exclusion of process measures. A single study examined the developer intents by survey. This study did not, however, examine the actual behaviors. The result is that product state is measured at various times in the process without an underlying theory or model to guide how the product state should change from one life-cycle phase to the next. Because the models are purely

phenomenological, they suffer two substantial problems for utility and one for credibility and generalizability.

First, the measures are late in the process (i.e., lagging). A product can be evaluated only after it has been completed to some late life-cycle state. Second, different developments use different practices, processes, tools, and developers. The apparent exceptions to process measures, developer network analysis, and code churn test this conclusion.

In the developer network analysis, the analysis can be performed only after the product is complete; similarly, code churn can only be measured after the code has been produced. Rather than being an in-process measure, it is a lagging indicator. Although this might be considered a process measure, it is actually a measure of developer behavior that may correlate with some aspect of process. The causation is also unclear as the communication patterns may result from some underlying difficulty. Code churn, though framed to suggest a "process" measure, is actually a size measure of the code product. There may be aspects of process, such as design effort, that affect the size of code churn, but only the artifact outcome is used in the models.

The third difficulty is that an underlying theory or model is not available to be examined and tested. A process-informed model could be used to guide behavior toward improvement or to evaluate the effectiveness of behavior changes or identify risk earlier in the development.

The difficulty of no guiding model is particularly important because the individual studies use a narrow set of data for fitting or calibration so that generalizability is already a concern. A theory would help frame analysis of generalizability. Moreover, the diverse set of metrics used raise the concern that any individual study might produce spurious results by chance. The effects are not particularly strong or convincing.

Only three of the studies explicitly addressed design issues. One focused on structural complexity. A second used a tool to induce stress by automating simulated malicious traffic. A third addressed modeling the performance degradation when fault-tolerant techniques or security mechanisms (e.g., encryption) were employed. None of the papers explicitly addressed requirements. Moreover, the studies of artifacts examined only the product (e.g., code complexity) rather than the design process leading to the results. The applications of design standards, design techniques, representations, inspections, design analysis, or design effort are ignored.

The use of defect and vulnerability history were used primarily to predict the location of other vulnerabilities. Truth tables addressed the probability by modules, but not the overall likelihood of vulnerabilities in the product or the expected count. Because this approach is limited, several questions remain for future research, including the following:

- What is the range of vulnerability densities and how do they vary by domain?
- Can defect or vulnerability history be used to set prediction intervals on the remaining number of defects? For example, can very low vulnerability levels be reliably predicted from low overall defect levels?
- Can the predictions be improved by including other data not always found in open-source repositories, such as defects found in test, use of specific development practices, and test cases?
- How effective and efficient would use of static or dynamic code analysis during development be? Studies indicate static analysis has some predictive capability for vulnerabilities but have

not addressed the engineering question of cost/benefit or how to include it in the development process. Moreover, the reports typically do not include the use of the tools as part of the development process, but rather post development.

- How strong are the correlations between defects and vulnerabilities?
- How do the correlations between defects and vulnerabilities vary by programming language, program size, defect density, or problem domain?
- How dependent are defect-vulnerability correlations on other conditions, such as test-case coverage or development process?
- How do results generalize among problem domains or between open- and closed-source development?
- What are the costs associated with false positives; that is, what level of false positives is acceptable?

4.6 Conclusions from Mapping Study

The literature in this field is difficult to locate because it has been published in a large variety of conferences and journals with inconsistent indexing and keywords. Although there are 60 authors identified, few have published repeatedly, and only a small core group exists. The core group published fewer than half the documents found, while the documents outside the core receive the most citations, and cross citations among the publications are limited. This is suggestive of a relative immaturity and lack of cohesion of the field.

Models have typically used code metrics and regression-related techniques. There is some work with developer behavior and intended behavior, and novel techniques include text analytics. No work has been identified that examines the requirements or design practices. Large open-source projects have been the main source of data, although one study used Cisco software and two used Microsoft IIS. Models that have been developed and tested tend to have high precision (low false positives) but low recall (low true positives). That is, identified modules are likely to be defective, but only a small fraction of defective modules are identified. Research is not yet “connecting the dots” and getting to the root causes so vulnerabilities can be effectively prevented during development.

The studies identified used heterogeneous models and approaches that are not suitable for a follow up meta-analysis. Moreover, reporting is often in the form of decision tables rather than statistical summaries. Better methods are needed to predict vulnerability levels during and after development, but new approaches would require more research and better connections in the research community. The field needs continued research leading to improved prediction in a way that directly addresses the management of development and customer assurance needs. More studies with more homogeneous methods are required if we are to someday aggregate studies for improved statistical power and generalizability.

5 Detailed Vulnerability Analyses

A major accomplishment for the DDSA project in FY13 was the detailed analyses of selected vulnerabilities from the CERT Vulnerability Notes Database.[4] This work provided considerable insight related to our research questions described below.

- Vulnerability databases have data that can be used by researchers to determine which vulnerabilities are likely to have an origin early in the SDLC.
- Using other public sources in addition to vulnerability databases, we can objectively identify architecture and design deficiencies and estimate impacts on multiple parties in many cases.
- We can define multiple, sensible ways to intervene in the SDLC (or operations) to mitigate vulnerabilities and their impacts.

We used the steps listed below to perform the detailed vulnerability analyses.

1. Define a heuristic to identify vulnerabilities that likely resulted from defects injected during the requirements and design phases of the software development life cycle.
2. Implement this process on CERT vulnerability data to identify design-related vulnerabilities.
3. Complete an initial orthogonal defect classification (ODC)/root cause analysis on the vulnerabilities identified in Step 2.
4. Develop a template of the necessary information required for a detailed analysis of a vulnerability.
5. Conduct the analysis on selected vulnerabilities.

5.1 Heuristic to Identify Design-Related Vulnerabilities

The heuristic used to identify design-related vulnerability candidates was implemented as a search filter, and then a manual inspection was performed by a researcher very familiar with the CERT Vulnerability Notes Database. The first step was to search the database for published vulnerability reports, excluding common terms in the title that typically indicate implementation (i.e., coding-related) vulnerabilities:

VulNoteInitialDate is after 01/01/1970 and
field Name does not contain overflow and
field Name does not contain XSS and
field Name does not contain SQL and
field Name does not contain default and
field Name does not contain cross and
field Name does not contain injection and
field Name does not contain buffer and
field Name does not contain traversal

[4] The Vulnerability Notes Database, online at http://www.kb.cert.org/ vulnerabilities/, provides timely information about software vulnerabilities. Vulnerability notes include summaries, technical details, remediation information, and lists of affected vendors.

The resulting vulnerability reports were then manually inspected to de-select those that did not appear to be design vulnerabilities. De-selected reports typically lacked sufficient information to determine cause or had strong indications of implementation-related vulnerabilities. Finally, the remaining reports were inspected, and those that appeared to have design causes were selected. From this process, 30 reports were chosen as design-related vulnerabilities and placed on the research list, shown in Appendix D. The team also completed an initial basic root cause analysis on each of the vulnerabilities to confirm that they were, in fact, likely to be caused by requirements or design defects.

The team then developed a detailed template to guide the detailed vulnerability analysis that included the following steps:

1. Identify the vulnerability. (What happened?)
2. Identify the cast of characters. (Who?)
3. Outline the main story identifying main events. (Timeline)
4. Create a “character X event” table identifying the impact of each change in state on the character.
5. Identify ODC-related attributes (including what might have prevented or mitigated the vulnerability).
6. Analyze economic impacts. (What actually took place compared to what could have taken place.)

The team selected three vulnerabilities from the table of 30 on which to do detailed analysis. The detailed analyses are included in Appendices E-G. Peer reviews were performed of each completed detailed vulnerability analysis by at least one team member not involved in that analysis. Below are short descriptions of the vulnerabilities analyzed. Some of the content in these descriptions comes from the CERT Vulnerability Notes Database and reflects the analysis, impact, workaround, and solutions at the time the associated note was originally written. Thus, the information describing the state of mitigation or “recent” research may no longer be up-to-date.

5.2 SYSRET Summary

VU#649219: SYSRET 64-bit operating system privilege escalation vulnerability on Intel CPU hardware

A full analysis of this vulnerability is presented in Appendix E.

5.2.1 Description

Some 64-bit operating systems and virtualization software running on Intel CPU hardware are vulnerable to a local privilege escalation attack. The vulnerability may be exploited for local privilege escalation or a guest-to-host virtual machine escape. A ring3 attacker may be able to specifically craft a stack frame to be executed by ring0 (kernel) after a general protection exception (#GP). The fault will be handled before the stack switch, which means the exception handler will be run at ring0 with an attacker's chosen RSP, causing a privilege escalation.

5.2.2 Impact

This security vulnerability affects 64-bit operating systems or virtual machine hypervisors running on Intel x86-64 CPUs. The vulnerability means that an attacker might be able to execute code at the same privilege level as the OS or hypervisor.

The x86-64 architecture was originally developed by AMD, with the aim of producing a 64-bit CPU that was backward-compatible with the 32-bit IA32 architecture. This was implemented in Intel's Pentium processor, among others. The vulnerability exists because of a subtle difference between AMD's implementation and Intel's. [5]

When running a standard operating system, such as Linux or Windows, or a virtual machine hypervisor, such as Xen, a mechanism is needed to rapidly switch back and forth from an application, which runs with limited privileges, to the OS or hypervisor, which typically has no restrictions. The most commonly used mechanism on the x86-64 platform uses a pair of instructions, SYSCALL and SYSRET. The SYSCALL instruction does the following:

- copies the instruction pointer register (RIP) to the RCX register
- changes the code segment selector to the OS or hypervisor value

A SYSRET instruction does the reverse; that is, it restores the execution context of the application. (There is more saving and restoring to be done—of the stack pointer, for example—but that is the responsibility of the OS or hypervisor.)

The difficulty arises in part because the x86-64 architecture does not use 64-bit addresses; rather, it uses 48-bit addresses. This gives a 256 terabyte virtual address space, which is considerably more than is used today. The processor has 64-bit registers, but a value to be used as an address must be in a canonical form; attempting to use a value not in canonical form results in a general protection (#GP) fault.

The implementation of SYSRET in AMD processors effectively changes the privilege level back to the application level before it loads the application RIP. Thus, if a #GP fault occurs because the restored RIP is not in canonical form, the CPU is in application state, so the OS or hypervisor can handle the fault in the normal way. However, Intel's implementation effectively restores the RIP first; if the value is not in canonical form, the #GP fault will occur while the CPU is still in the privileged state. A clever attacker could use this to run code with the same privilege level as the OS.

Intel says that this is not a flaw in its CPU since it works according to its written spec. However, the whole point of the implementation was to be compatible with the architecture as defined originally by AMD. Quoting from Rafal Wojtczuk, "The root cause of the vulnerability is: on some 64 bit OS, untrusted ring3 code can force the kernel to execute SYSRET instruction that would return to a non-canonical address. On Intel CPUs, this results in an exception raised while still in ring0. This exception cannot be handled safely."[6]

---

[5] Much of this impact description is summarized from Rich Gibb's website at http://richg74.wordpress.com/2012/06/19/us-cert-intel-cpu-vulnerability/.

[6] http://media.blackhat.com/bh-us-12/Briefings/Wojtczuk/BH_US_12_Wojtczuk_A_Stitch_In_Time_WP.pdf

Clearly, many OS and hypervisor vendors with considerable market presence were affected. Multiple parties could have prevented the vulnerability as Intel's SDM is very clear on the behavior of SYSRET (and not every x86-64-based OS or hypervisor was affected). For example, they could have adopted a safer transition back to the application following a SYSCALL. While originally noted and reported by the Linux community back in 2006, the vulnerability was characterized and easily dismissed as a Linux-specific issue. Also from Wojtczuk, "This is likely the reason why developers of other OS have not noticed the issue, and they remained exploitable for six years." Intel could also have prevented the vulnerability by not introducing a dangerous re-interpretation of how to return from a rapid system call.

5.2.3 Solution

Reading some of the references above and considering the short time window where all seems to be fixed (from April to June 2012) might give the impression that the vendor only needed to find (for its OS or hypervisor) a different, safer way to handle SYSRET (e.g., return other than through SYSRET or check for a canonical address), but doing so is not straightforward. That perhaps the same patch/approach might not work for all affected OS can be seen in the different ways the vulnerability can be exploited for different OS. So, each vendor must conduct its own careful analysis of what computing assets are at risk or can be leveraged for an exploit and carefully redesign/code system calls/returns to ensure safe transition from application to system and back again. Also, as the intent of SYSCALL/SYSRET is that these calls be reserved for something only the OS can do but for which execution performance is critical (e.g., by minimizing saving off registers, except for those actually needed by the system function being called), the OS-specific patch(es) need to be designed and coded for execution speed as well as safe transition.

One of the vendors, Xen, has been particularly revealing relative to the considerable difficulties it encountered in working with select stakeholders to diagnose, design, code, and test patches for VU#649219, including providing a detailed timeline that describes an enormous amount of coordination and analysis "behind the scenes," giving rise, no doubt, to enormous frustration. (See Appendix E for further details.)

5.3 DNS Resolvers Summary

VU#800113: DNS resolvers don't sufficiently randomize DNS query ID () or source port.

A full analysis of this vulnerability is presented in Appendix F.

5.3.1 Description

The domain name system (DNS) is responsible for translating host names to IP addresses (and vice versa) and is critical for the normal operation of internet-connected systems. DNS cache poisoning, sometimes referred to as cache pollution, is a technique that allows an attacker to introduce forged DNS information into the cache of a caching name server. DNS cache poisoning is not a new concept; in fact, there are published articles that describe a number of inherent deficiencies in the DNS protocol and defects in common DNS implementations that facilitate DNS cache poisoning.

5.3.2 Impact

The following are examples of these deficiencies and defects:

- **Insufficient transaction ID space**

  The DNS protocol specification includes a transaction ID field of 16 bits. If the specification is correctly implemented and the transaction ID is randomly selected with a strong random number generator, an attacker will require, on average, 32,768 attempts to successfully predict the ID. Some flawed implementations may use a smaller number of bits for this transaction ID, meaning that fewer attempts will be needed. Furthermore, there are known errors with the randomness of transaction IDs that are generated by a number of implementations. Amit Klein researched several affected implementations in 2007. These vulnerabilities are described in the following vulnerability notes:

  - VU#484649 - Microsoft Windows DNS Server vulnerable to cache poisoning
  - VU#252735 - ISC BIND generates cryptographically weak DNS query IDs
  - VU#927905 - BIND Version 8 generates cryptographically weak DNS query identifiers

- **Multiple outstanding requests**

  Some implementations of DNS services contain a vulnerability in which multiple identical queries for the same resource record (RR) will generate multiple outstanding queries for that RR. This condition leads to the feasibility of a "birthday attack," which significantly raises an attacker's chance of success. This problem was previously described in VU#457875. A number of vendors and implementations have already added mitigations to address this issue.

- **Fixed source port for generating queries**

  Some current implementations allocate an arbitrary port at start-up (sometimes selected at random) and reuse this source port for all outgoing queries. In some implementations, the source port for outgoing queries is fixed at the traditional assigned DNS server port number, 53/udp.

  Recent additional research into these issues and methods of combining them to conduct improved cache-poisoning attacks has yielded extremely effective exploitation techniques. Primarily, caching DNS resolvers are at risk—both those that are open (a DNS resolver is open if it provides recursive name resolution for clients outside of its administrative domain) and those that are not. These caching resolvers are the most common target for attackers, but stub resolvers are also at risk.

5.3.3 Solution

Because attacks against these vulnerabilities all rely on an attacker's ability to predictably spoof traffic, the implementation of per-query source-port randomization in the server presents a practical mitigation against these attacks within the boundaries of the current protocol specification. Randomized source ports can be used to gain approximately 16 additional bits of randomness in the data that an attacker must guess. Although there are technically 65,535 ports, implementers cannot allocate all of them (e.g., port numbers <1024 may be reserved; other ports may already be allocated). However, randomizing the ports that are available adds a significant amount of attack resiliency. It is important to note that without changes to the DNS protocol, such as those that the DNS security extensions (DNSSEC) introduce, these mitigations cannot completely prevent cache

poisoning. However, if properly implemented, the mitigations reduce an attacker's chances of success by several orders of magnitude and make attacks impractical.

5.4 Advanced Micro Devices/Address Space Layout Randomization Summary

VU#458153: Graphic card drivers do not support ASLR

A full analysis of this vulnerability is presented in Appendix G.

5.4.1 Description

Advanced micro devices (AMD)/ATI video card driver software design requires known/static address space layout and does not support address space layout randomization (ASLR); graphics drivers are kernel on Windows so machines crash on boot. Non-randomized locations allow attackers to use return-oriented programming (ROP) methods to bypass other runtime mitigations like data execution prevention (DEP).

5.4.2 Impact

AMD video drivers were a problem because they often precluded end users (including IT organizations) from running ASLR "always on," which the CERT Division and others (e.g., Microsoft) were encouraging as a means of reducing exposure to ROP-type attack vulnerabilities. A key to the success of ROP-based attacks is having a sufficiently large code base with guessable content at known addresses for the attacker to work with. The AMD video driver code base had a large footprint (perhaps a megabyte or more—see http://support.amd.com/us/gpudownload/windows/legacy/Pages/legacy-radeonaiw-vista64.aspx) and hence could contribute to a successful ROP-type attack.

The incompatibility of AMD video card drivers and ASLR was becoming an increasingly important concern because of the following:

1. Video card drivers generally increase in size (new versions incorporate new features), making more code available to successfully engineer a ROP-based attack.
2. To provide an effective defense, ASLR must be enabled for all processes. When any process (e.g., the AMD driver) does not use ASLR, an attacker can use ROP techniques against that process to execute arbitrary code.
3. Attackers might increasingly be drawn to try to engineer such attacks as more time passes and the code base becomes increasingly studied and known (through leaks, OSS efforts, etc.).
4. When ASLR is not set "always on," the end user must identify which applications will need to be opted in, a conscious act requiring time, thought, skill, and effort and thus prone to error (or worse, indefinite postponement).

5.4.3 Solution

1. AMD ensured that source code for the driver did not depend on "known addresses."
2. AMD modified driver software to be compatible with ASLR.
3. AMD improved communication between its driver team and its tech support team to improve overall responsiveness to warnings of potential vulnerabilities.

.

5.5 Summary of Detailed Analyses

Table 4 summarizes the three vulnerabilities for which we completed a detailed analysis (see Appendices E-G for full analyses).

Table 4: Summary of Vulnerability Attributes

	VU#649219:SYSRET 64-bit OS privilege escalation vulnerability on Intel CPU hardware	**VU#800113: DNS resolvers don't sufficiently randomize**	**VU#458153: AMD/ATI Graphic card drivers do not support ASLR**
Stakeholder Dependencies	1. Virtualization Infrastructure (VI) users ->[7] VI hosts -> VI vendors. 2. Other users -> OSs (e.g., Microsoft Windows) on 64-bit CPU (less explored here)	Critical Infrastructures (and other internet users) -> DNS vendors.	Windows and internet users for whom security is important (everyone?) -> Microsoft EMET (ASLR, DEP) + AMD/ATI video card driver.
What's Happening Over Time	Internet and cloud server market continues to grow, and so does application dependence on that market. An undetected vulnerability opens the door to attacking increasing high-value targets.	Internet use expands, encompassing more critical infrastructures, making DNS resolvers a more tempting target.	Demand for 2D/3D video streaming and video card driver footprints is increasing, making ROP tempting. Security settings are getting complex. The broader ecosystem benefits when all players adopt improved security features.
Attack	Carefully crafted code can force kernel to execute SYSRET that would return to a non-canonical address, resulting in a #GP interrupt with ring0 privilege. (The exact triggers are VI vendor product specific.)	DNS cache poisoning through response spoofing (MiTM, DoS).	An automated attack could exploit a first vulnerability (e.g., buffer overflow) and use the resulting crash to apply ROP against known video driver addresses to read sensitive data or deposit malware, bypassing runtime EMET mitigations.
Potential Extent of Attack	Users of cloud services dependent on Intel 64-bit CPU-based internet servers. Vendors include Xen, Microsoft, and others. About 100 VI and OS products and versions are involved (but not AMD and not 32-bit CPUs).	Almost everything on the internet depends on DNS returning the right number for the right request. Vendors included Microsoft, Linux/ICS, Sun, Cisco, and others. 1-3% of monitored unpatched nameservers had a poisoning event detected.	About 300 million Windows and internet users have AMD video cards.
Underlying Flaws	1. AMD specification of SYSRET in system developer's manual (SDM) leaves ambiguous how CPU will behave during SYSRET. 2. Intel adopts a conformant but different interpretation, which is unsafe. 3. Complexity of "the programming stack" and looseness of instruction-set specification obscures attack opportunities. 4. Vulnerability analysis process (including stakeholder consultation and notification) is very complex.	1. Design of domain name system (DNS) protocol. 2. Inadequate randomization of transaction ID and output port allows spoofing of DNS resolution transaction responses leading to DNS cache poisoning.	AMD/ATI video card driver design requires known/static address space layout, precluding end users (and IT) from running address space layout randomization (ASLR) “always on.” Thus, 1) due to the sizeable predictable footprint, users are vulnerable to ROP attacks and 2) user sophistication and discipline are needed to manage security settings (that are not “always on”).

[7] We use an arrow to indicate dependence among stakeholders: A->B means that stakeholder A depends on what stakeholder B does to operate securely.

	VU#649219:SYSRET 64-bit OS privilege escalation vulnerability on Intel CPU hardware	VU#800113: DNS resolvers don't sufficiently randomize	VU#458153: AMD/ATI Graphic card drivers do not support ASLR
Mitigations	1. OS and VI vendors patch their products (examine every SYSRET call and include a safe prologue). 2. VI hosts apply additional firewalls, ACLs, permissions to limit reach of compromised hosts. 3. VI purchasers consider hybrid cloud solutions.	1. Apply patches to increase randomization. 2. Adopt DNSSEC (security extensions).	1. AMD updated source code, so driver did not depend on "known addresses." 2. AMD modified driver software to be compatible with ASLR. 3. Updates were released and distributed.
What Triggered Mitigation	Security researchers (e.g., Rafal Wojtczuk) identify and communicate the vulnerability and risk; media spotlights potential effects. Coordination by security researchers, the CERT Division, and possibly affected vendors.	Security researchers (e.g., Dan Kaminsky) identify and communicate the vulnerability and risk; media dramatically spotlights potential effects. Coordination of simultaneous patches by security researchers and the CERT Division.	Certain well-regarded blogging sites and security newsletters brought unwanted (from AMD's perspective) attention to the vulnerability, AMD's role in it, and potential consequences to users and the broader Windows ecosystem. AMD had something precious to lose: market share to NVIDIA should its concern for user security be increasingly questioned.
How It Could Have Been Prevented	1. Provide unambiguous, safe specification for SYSRET. (AMD CPU and SDM: improvements to requirements development and design processes.) 2. Design CPU to be compatible not just with AMD SDM but also with AMD CPU instruction behavior to ensure deep compatibility. (Intel CPU: improvements to requirements development and design processes.) 3. Design SDM to include adequate warning of how to safely prologue use of SYSRET. (Intel CPU and SDM: improvements to requirements development and design processes.) 4. Require and verify secure VI from vendors. (VI hosts: improvements to requirements development, design, risk management, and vendor management processes) 5. Monitor emerging vulnerabilities and confirm whether one's products have the vulnerability. (Intel, VI vendors & hosts: improvements to ongoing validation processes) 6. Specify policies to ensure the safe, correct, predictable use of SYSRET. (OS and VI vendors: improvements to requirements development and design processes)	1. Design DNS protocol for updates and address trust. 2. Consider low-entropy-based response spoofing and DNS cache poisoning in threat modeling. (Improvements to requirements development processes) 3. Select algorithms that provide sufficient entropy in transaction ID and port utilization. (Improvements to design processes.) 4. Monitor emerging vulnerabilities and confirm whether one's own products have the vulnerability. (Improvements to requirements development ongoing validation processes)	1. Identify and evaluate threats (threat modeling). Needs to be ongoing because risks may increase due to network effects (or, in the case of ROP, as code footprint gets larger). 2. Leverage security enhancements offered for the platform for which you develop (Windows VISTA EMET). (Improvements to requirements development and design processes) 3. Monitor emerging threats (e.g., ROP) and security enhancements (e.g., EMET) and evaluate impact on one's own products and design practices. (Improvements to ongoing validation and design processes)

6 System Dynamics Model and Simulation

The research team began the third phase of the project, developing an initial economic model, after conducting the detailed vulnerability analyses. We chose to construct a system dynamics model (SDM), which allows people to study systems with many interrelated factors using stocks and flows [Sterman 2000]. The goal of a simulation model is first to represent the normal behavior of a system and then to introduce new input to see how the responses change.

Our model, created using Vensim, represents the design vulnerability life cycle and includes variables representing key influencers gleaned from the literature search, detailed vulnerability analyses, and experience with TSP process. The variables influencing flow were modeled and incorporated into the SDM and the model was exercised using hypothetical data to assess their impacts. The model is presented in this section after a brief introduction to the system dynamics method.

6.1 System Dynamics Background

The system dynamics method enables analysts to model and analyze critical behavior as it evolves over time within complex socio-technical domains. A powerful tenet of this method is that the dynamic complexity of critical behavior can be captured by the underlying feedback structure of that behavior. The boundaries of a system dynamics model are drawn so that all the enterprise elements necessary to generate and understand problematic behavior are contained within them. The method has a long history and is described in *Business Dynamics: Systems Thinking and Modeling for a Complex World* [Sterman 2000].

System dynamics and the related area of systems thinking encourage the inclusion of soft factors in the model, such as policy, procedural, administrative, psychological, and cultural factors. The exclusion of soft factors in other modeling techniques essentially treats their influence as negligible, which is often an inappropriate assumption. By taking the more holistic approach to modeling that SDM offers over other modeling approaches, analysts can identify mitigations to problematic behavior that are otherwise easily overlooked.

In the case of design vulnerabilities, Table 4 in Section 5 shows that soft factors (e.g., refusal to agree there is a problem and the motivation to act brought by media attention) play an important role in the creation, persistence, and eventual resolution of some costly design vulnerabilities.

Figure 11 summarizes the notation used in system dynamics modeling. The primary elements are variables of interest, stocks (which represent collection points of resources), and flows (which represent the transition of resources between stocks). Signed arrows represent causal relationships, where the sign indicates how the variable at the arrow's source influences the variable at the arrow's target. A positive (+) influence indicates that the values of the variables move in the same direction, whereas a negative (-) influence indicates that they move in opposite directions. A connected group of variables, stocks, and flows can create a path that is referred to as a feedback loop.

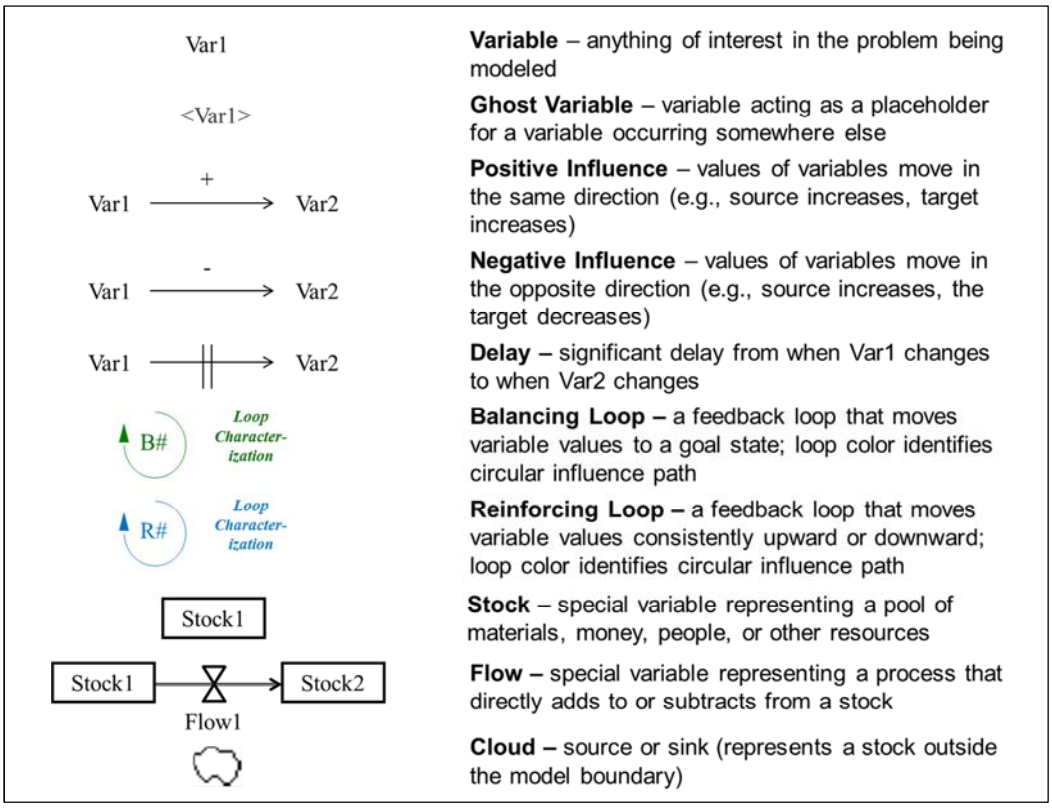


Figure 11: System Dynamics Notation

Because the initial scope of our simulation efforts is on a single time through the development and release cycle (i.e., a single increment), no noteworthy feedback loops are simulated, and thus the balancing and reinforcing loop notations identified in Figure 12 do not appear in our model. In the future we plan to expand the model to include feedback loops. For example, discovered vulnerabilities could be used to adjust or improve practices for secure design and coding. Currently, these notions are represented as variables in Figure 13: Known Vuls in Software appears in the upper right of the figure, and *attn to secure design practices* and *attn to secure coding practices* appear toward the left and center of the figure. The latter two variables are input-only variables that are external to the model but that can be adjusted to drive the simulation. These two variables do not currently receive input from Known Vuls in Software. Expanding the model to include these feedback loops will provide the opportunity to evaluate the impact on multiple development increments over time.

6.2 System Dynamics Model

Figure 12 provides an overview of the system dynamics model that we developed. The subsections that follow incrementally describe the fundamental characteristics of this model.

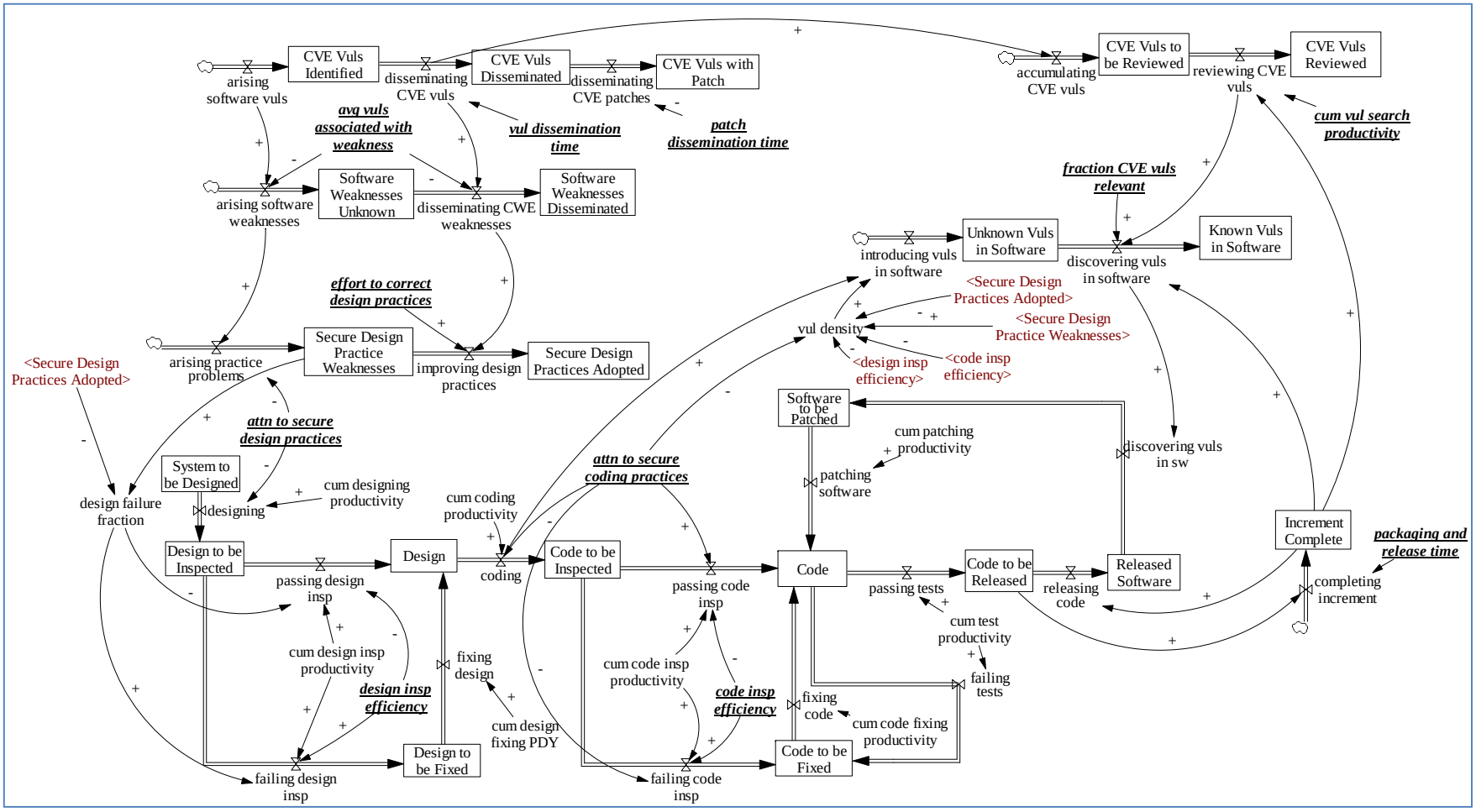


Figure 12: System Dynamics Model Overview

6.2.1 Software Development Flow

Figure 13 depicts a typical software development flow. Starting on the left, the designed system is inspected, and any artifacts that fail are fixed. The rate of designing the software system and the rate of inspection of software artifacts are regulated by associated productivity measures as shown. The discovery of problems in the design depends on the design inspection efficiency (i.e., more efficient inspections lead to greater detection of problems, all other things being equal). The variable *design insp efficiency* is underlined and italicized to indicate that it is an input parameter to the simulation.

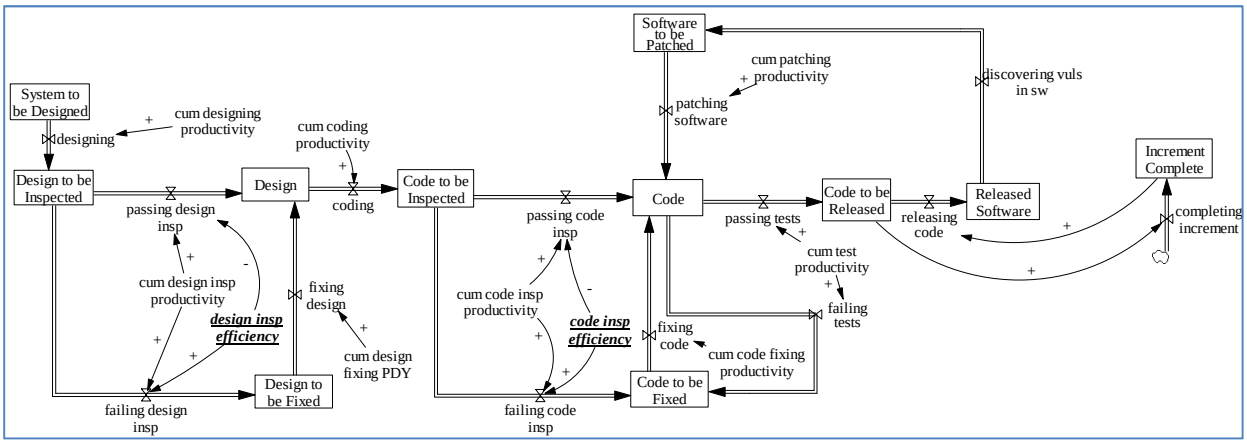


Figure 13: Software Development Flow

Coding is depicted toward the middle of the diagram followed by an inspection process similar to that for the design. From the stock named Code, more detailed acceptance testing results in either more fixes or passing of tests with an accumulation of Code to be Released. Once the increment is complete, the code can be released. Of course, any later discovery of vulnerabilities in the code needs to be patched and the code re-tested. Subsequent sections below deal with the implications of inattention to secure design and coding practices, which influences the discovery of vulnerabilities later in the lifecycle.

6.2.2 Attention to Secure Design Practices

Figure 14 highlights the effects of secure design practices on the software development life cycle. The new part of the model is at the top of the figure. The variable *attn. to secure design practices* by the developers influences both the rate of the system design and the rate at which practice weaknesses arise in the design. The variable design failure fraction, along the middle left of the figure, influences the ratio of design artifacts that fail design inspection. The design failure fraction is influenced by Secure Design Practice Weaknesses and Secure Design Practices Adopted. Note that the latter variable is represented as a stock in the upper right portion of the figure and as a ghost variable in the upper left portion to reduce the clutter of arrows crossing in the full model exposition.

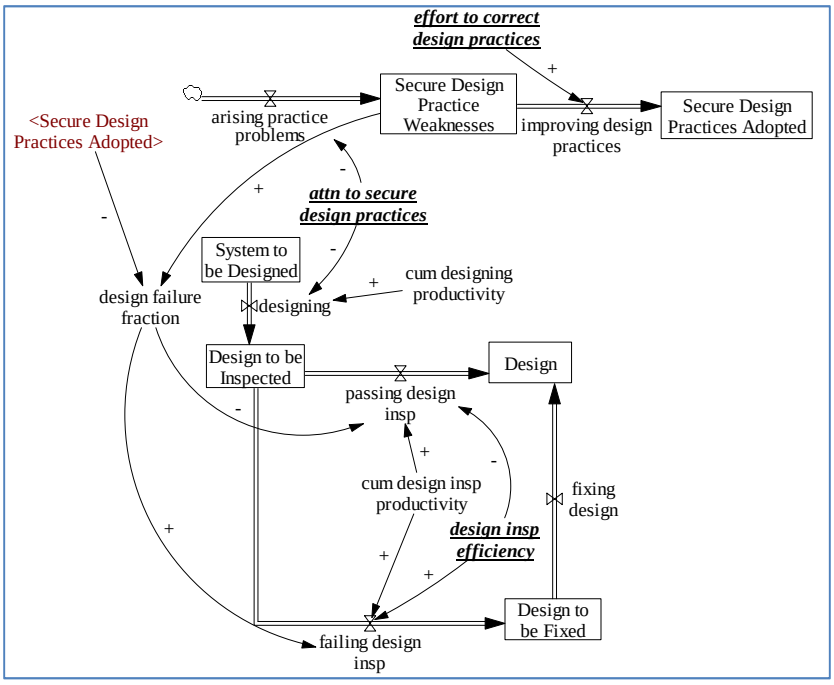


Figure 14: Attention to Secure Design Practices

6.2.3 Attention to Secure Coding Practices

Figure 15 shows the effects of secure coding practices on the software development life cycle. The input variable *attn to secure coding practices*, shown in the middle left, influences the rate of coding, the ratio of artifacts failing code inspection, and the vulnerability density in the software system. Actually, vul density is a key variable in the model that is influenced by the secure design practices and inspection efficiencies in the development and review processes. As shown, the vulnerability density influences the introduction of vulnerabilities in the software, which are initially assumed to be unknown. Once the software increment is complete, the fact that vulnerabilities are a problem in the product becomes known. Identifying the places in the software code where these vulnerabilities reside allows patching of those vulnerabilities and re-release of the patched system.

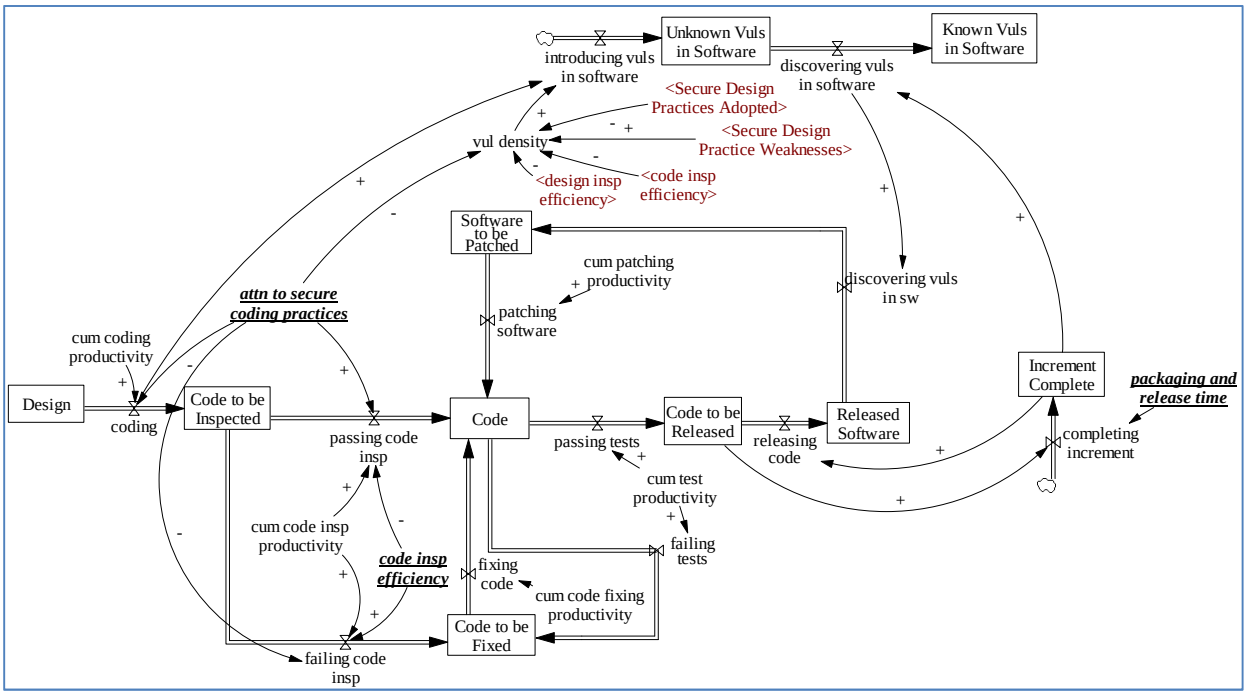


Figure 15: Attention to Secure Coding Practices

6.2.4 CVE and CWE Dissemination and Review

Figure 16 shows the effects of common vulnerabilities and exposures (CVE) dissemination and review. Data on CVE identification rates are used to instantiate the arising software vulnerabilities variable in the top left portion of the figure. Once the CVEs are disseminated, software developers can review them to discover vulnerabilities that might exist within their own systems, as shown on the left side of the figure. New software vulnerabilities can sometimes lead to new classes of unknown software weaknesses. The dissemination of these software weaknesses as part of the common weaknesses enumeration (CWE) resource can lead to improving the security design practices, as shown on the bottom left portion of the figure, if an organization actually puts effort into improving the design practices based on the information disseminated.

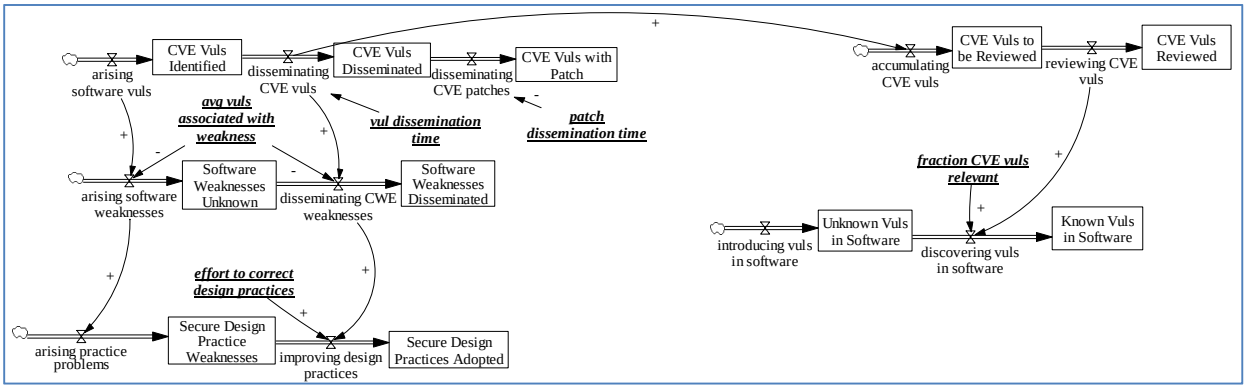


Figure 16: CVE and CWE Dissemination and Review

The overview of the model presented at the beginning of this section integrates the model segments presented in the subsections above. The next section elaborates some of the key simulation results

from this preliminary model. While some available CVE data were used to run the simulation more robust and comprehensive data are needed, especially in the following areas:

- CVE vulnerability discovery, dissemination, and patch production
- creation of CWE weaknesses through the analysis of CVEs, and the dissemination of those weaknesses
- effectiveness and timeliness of patches generated in response to CVEs disseminated
- effectiveness and timeliness of process improvement approaches (improving design and inspection practices) generated in response to CWEs disseminated

6.3 Initial Simulation

The model was exercised on hypothetical data to assess the impact that increasing attention to design might have on the outcomes that stakeholders care about. Stakeholders might include, for example, developers, managers of developers, acquirers, and those in operations. This was an initial run since we intend to carefully calibrate the SDM with data from the TSP database in the future as it becomes available. The TSP data would also provide a default calibration of the SDM when deployed on an actual project. In such a situation, the default calibration would be replaced by data from the project or similar projects as it becomes available. Nevertheless, the model simulation results, presented in this section, tell a seemingly natural story.

Developers could choose to pay attention to security in the software design process with varying degrees of rigor, as the variable *attn to secure design practices* allows in the above SDM. In the simulation graphs that follow, we execute the model with low, medium, and high levels of attention to security and show the results for each simulation run. The simulation model is structured to reflect the typical life-cycle sequence of source code release followed by vulnerability discovery, patching, and re-release of the software. Figure 17 shows the fraction of the increment that has been packaged for release.[8] As the figure shows, while low attention to secure design practices may be more efficient initially, problems occur that need to be fixed, so a low level of rigor results in added work and a later release time than either medium or high attention to design security. However, the graph also shows that, at some point, increasing attention to design has diminishing returns (i.e., modestly extending the development schedule), suggesting that there is a "sweet spot" even for an incremental release, which is important if being first to market is an objective. However, the difference is minimal in the simulation and relates only to the initial incremental release.

[8] Future model refinements will involve multiple increments released through time.

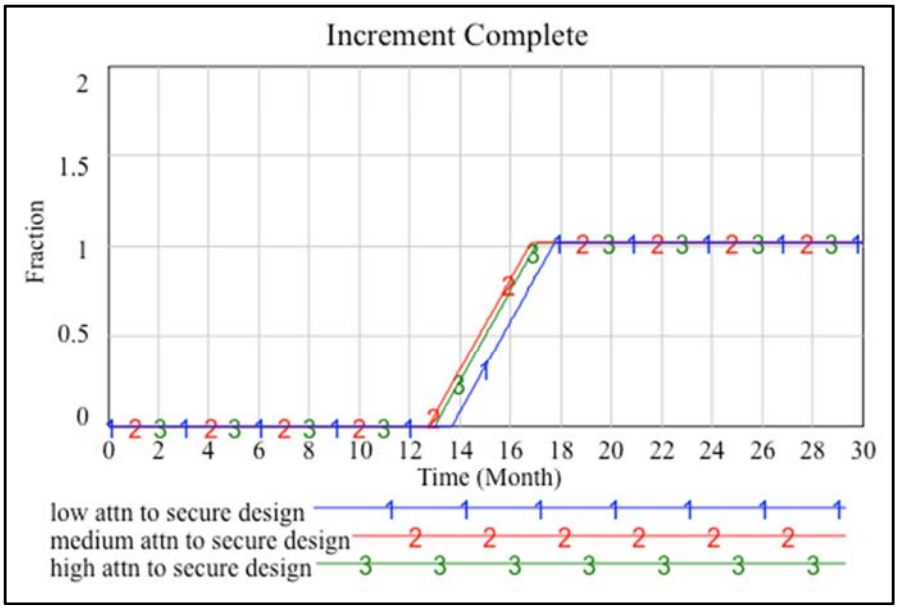


Figure 17: Increment Complete

Of course, that initial release comes before problems are discovered in the field. As shown in Figure 18, the initial release between months 16 and 18 for the three runs is followed by patching and re-release of the software. Because of the significant number of additional problems that occur for lower levels of attention to secure design practices, the re-release is significantly delayed for the medium and low levels of attention. Low attention to secure design practices delayed the re-release (i.e., the bug fix release) until about month 27. Medium attention resulted in delays to month 24. In contrast, high attention to security design practices led to a re-release at about month 19, a full 8 months before the low attention run and 5 months before the medium attention run.

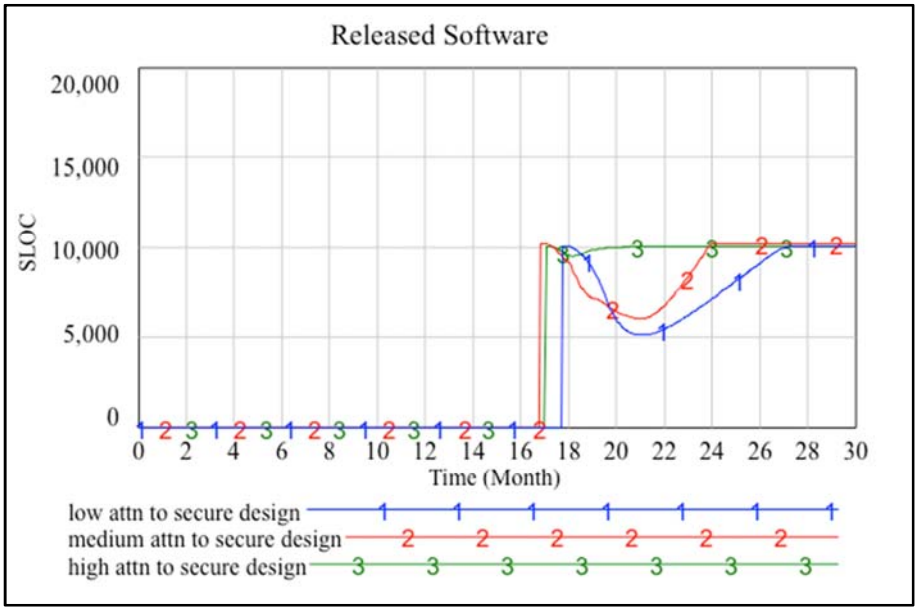


Figure 18: Released Software

Figure 19 and Figure 20 show the reasons for the delay in the re-releases. Unknown vulnerabilities are more prevalent in the software developed with less attention to secure design practices, result-

ing in more work in the re-releases to find and patch those problems. This preliminary, proof-of-concept model shows that high attention to secure design pays off in the end and does not delay initial release by much, if at all. By contrast, low attention to secure design is costly throughout the product life cycle, resulting in a higher total cost of ownership.

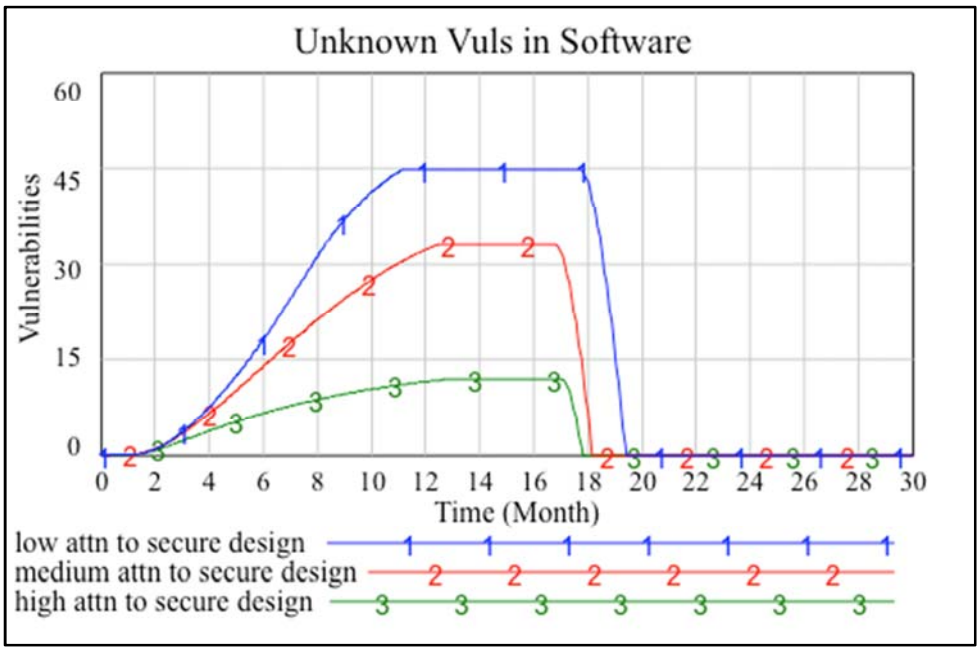


Figure 19: Unknown Vulnerabilities

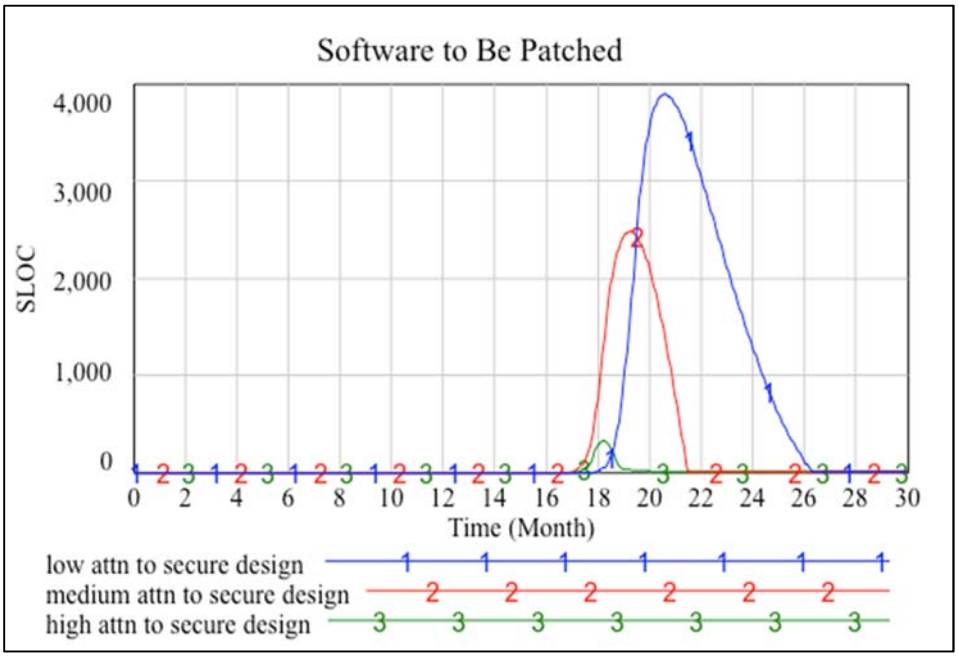


Figure 20: Software to Be Patched

6.4 Conclusions from the Initial Model Development

The SDM we developed of a design vulnerability life cycle included variables representing key influencers and was exercised with hypothetical data. The initial simulation suggests that improving attention on secure design may lead to the following:

- modest improvements in “increment complete” duration (what developers care about)
- more of the released software functionality remaining available to operations (what operations and the acquirer care about)
- reduced fix/patch backlog and thus more staff available for new product development (what development management cares about)
- fewer latent design vulnerabilities (what all parties should care about)

7 DDSA Results, Stage 1

In FY13, the DDSA project established that a number of significant, pernicious, and infamous vulnerabilities likely have their origin in requirements and design activities. A qualitative research approach was pursued with tasks falling under three major phases: literature review, detailed vulnerability analyses, and SDM development.

Literature review – In FY13, the project team performed a literature review addressing vulnerability taxonomies (e.g., CVE, CWE), secure requirements and design practices (e.g., Microsoft Security Development Lifecycle, Open Web Application Security Project), and estimating impacts (e.g., COQUALMO). We also conducted a systematic mapping study on predicting the level of design vulnerabilities. In general, the literature on design vulnerabilities is very sparse, especially with regard to development and impact metrics, offering almost no foundation for a meta-analysis or solution that the project could leverage.

Detailed vulnerability analyses – Given the sparseness of the research literature, we performed a series of detailed vulnerability analysis explorations into selected design vulnerabilities to better understand their life cycle (origin, discovery, reporting, and remediation) in context, characterizing the vulnerabilities' effects on major stakeholders including users, the vendors whose software contained the vulnerability, and the providers and maintainers of the platform in which the software operated. Ecosystems sampled included the following:

- virtualization and cloud software (Xen on Intel 64-bit CPUs)
- video streaming and gaming (AMD/ATI drivers on Windows)
- the internet's own infrastructure (DNS resolvers)

This demonstrated that design vulnerabilities exist and affect many major computing ecosystems. A recurring theme in the detailed vulnerability analyses is that the software vendors favor development paths that tangibly and immediately serve user or market demands over alternatives that more fully position the broader ecosystem for resilience to less tangible but growing threats. In more graphical prose, the vendor is the captain of the Titanic with all major stakeholders on board, steering the ship through the dense fog further and further into dangerous waters. Can the individual initiative of the passengers (research analysts and other vendors and organizations acting quietly behind the scenes) save the ship and its other passengers in time?

SDM development – In FY13, we developed an initial system dynamics model (SDM) in Vensim. The SDM spans the life cycle of a design vulnerability from its origin in requirements and design through operations to enable analysis and estimation of longer term impacts of multiple vulnerabilities under different "what if" development scenarios. Key variables influencing flow (design failure rate, vulnerability density, inspection efficiency, new vulnerabilities discovery rate, and productivity) were modeled and incorporated into the SDM. We then exercised the model in an initial simulation to assess the impacts that secure design practices might have. We learned that under hypothetical circumstances, a broader initial focus during design can benefit all parties. While paying attention to secure design may cost a little more initially, it yields substantial benefits to both the developer and operational communities.

8 Proposed Next Steps, Stage 2

In the second stage of this project, which depends on available SEI research and customer funding, we propose to further develop and pilot the vulnerability life-cycle SDM using the following steps:

1. Extend the vulnerability life cycle systems dynamics model to address the following:
   a. additional feedback loops (see discussion in Section 6.1)
   b. multiple release increments (What are the cumulative, longer-term effects of design vulnerabilities?)
   c. requirements elicitation (a source of emerging design vulnerabilities when the surrounding ecosystem changes)
   d. operations incident capacity (At what point does patching overwhelm operations?)
2. Calibrate the model by estimating ranges for variables representing these key influencers of vulnerability density:
   a. defect injection rates and defect detection rates (including how such rates are affected by particular mitigations)
   b. development and operations staffing costs, established in part from mining the TSP database

   These variables will help assess where mitigations can most cost effectively be applied (and provide an argument for developing and deploying such mitigations).
3. Select promising secure requirements and design practices and estimate their impacts.

   Sources for practices include the FY13 literature review, public vulnerability taxonomies (e.g., Building Security In Maturity Model (BSIMM)), and detailed vulnerability analyses. Possible sources for data include the TSP database and estimates from the CERT Division; however, in many cases, practice-specific impacts may have to be estimated by a user organization's process owners on a case-by-case basis, especially for novel practices.
4. Refine the SDM through reviews and piloting. Collaboration with an external organization would provide mutual benefits, allowing us to test our model while providing the organization with a fully functional model calibrated with its own data and estimates.
5. Develop criteria and guidelines for intended SDM use.

The following research questions frame our planned work for Stage 2.

Table 5: Research Questions for Stage 2

Research Question	Approach to answering
1. Does the vulnerability life-cycle model address the key factors affecting design vulnerability density? *(Identified some key factors in FY13 through literature review and detailed vulnerability analyses)*	Similar factors are used by high-discipline development teams to estimate defect density. We plan to validate that the key variables have been covered by convening a Technical Challenge Workshop of leaders from high-discipline software organizations.
2. Can data for calibrating the SDM be easily obtained? Is it domain or program specific?	High-discipline development teams (e.g., TSP teams) within organizations that routinely collect and maintain such process data should be able to establish an initial calibration of the SDM, replacing it with project-specific calibrations during the first and second release increments. We will evaluate feasibility and ease of performing such calibration through pilots within high-discipline organizations. Data for innovative practices with no usage history may have to be estimated based on a broader statistical understanding of a particular organization's development practices or by analogy with practices that have such a usage history.
3. Are the estimates of practice costs and impacts produced for the SDM sufficiently accurate and precise to support quality-cost-schedule tradeoffs? *(High-discipline development teams routinely make such tradeoffs using defect density, but the economics of design vulnerabilities are likely underappreciated*.)	Mine the TSP database, which now includes design practice and defect data, to independently evaluate reasonableness of cost and impact estimates.
4. Can the resulting models be used (1) during project planning to ensure adequate attention to design and inspection practices and (2) during acquisition to determine what incentives will help ensure contractor attention to broader program needs for security and total cost of ownership?	Evaluate usefulness for quality-cost-schedule tradeoffs through reviews and pilots.

9 Conclusions

Our initial research questions were effectively answered through our FY13 efforts. We confirmed that the current ship-then-fix approach is sub-optimal and in the long term untenable. Our analyses of vulnerabilities included examples in which vulnerabilities could never be fully eradicated from the user community once the product was distributed. The system dynamics model we developed showed that even at the level of a single development increment, the economics often favor earlier attention to security-related requirements and design, as well as ongoing validation. In other words, it is often not necessary to consider longer time scales to experience benefits that exceed the costs, for all major stakeholders.

We began our research by surveying the research literature with a mapping study to determine what was known about when vulnerabilities become visible and how that information might be actionable, and we drew these conclusions:

- The programs covered tended to be large, highly used open-source programs (e.g., Mozilla), limiting generalizability.
- Only three of the studies explicitly addressed design issues, and none addressed requirements.
- The focus was almost exclusively on outcome measures, or at best, lagging indicators, to the exclusion of in-process measures (e.g., from design reviews), providing little information that is actionable early in the lifecycle.

In terms of the utility of the models and results obtained, we drew these conclusions:

- Models that have been developed and tested tend to have high precision (low false positives) but low recall (low true positives). That is, identified modules are likely to be defective, but only a small fraction of defective modules are identified. Research is not yet "connecting the dots" and getting to the root causes so vulnerabilities can be effectively prevented during development.
- Better methods are needed to predict vulnerability levels during and after development, but new approaches would require more research and better connections in the research community. More studies with more homogeneous methods are required if we are to someday aggregate studies for improved statistical power and generalizability.

Next, in our analyses of vulnerabilities, we discovered that vulnerability databases supplemented by information generally found on the internet often can provide sufficient information to retrospectively link known vulnerabilities to suspected deficiencies early in the software development life cycle, from which indicators to monitor and processes to improve can be inferred. We can identify multiple sensible ways to intervene in the software development lifecycle or operations to mitigate vulnerabilities and their impacts. In fact, both the vulnerability analyses (in Section 5) and the system dynamics model (in Section 6) suggest specific ways to intervene early in the SDLC, often to the benefit of all stakeholders, including the developer.

A recurring theme in the detailed vulnerability analyses is that software vendors often favor development paths that serve immediate market demands over alternatives that more fully position the broader ecosystem for resilience to less tangible but growing threats. The costs attendant to addressing a vulnerability in operations, and concern over what actions competitors and customers might take, often result in the vendor continuing to refuse to acknowledge suspected or known vulnerabili-

ties. If security analysts and the media succeed in attracting sufficient public attention to the potential consequences, the vendor may finally be pressed to take action. The costs associated with a vulnerability and its later resolution are often borne by multiple parties and over a long period of time.

In one area, the research team only partially succeeded: collecting the data needed to develop a decision model to evaluate quality-cost-schedule tradeoffs. The system dynamics model (Section 6) largely fits the bill but was calibrated based on one coach's experience with TSP teams. In Stage 2, we plan to make use of the TSP database, when more complete and detailed statistics relevant to the variables used in the model and simulation are made available.

In summary, the research team demonstrated that an economic model could be constructed and calibrated to assist organizations conducting tradeoff analyses to compare alternative interventions. The system dynamics model described in Section 6 models early SDLC parameters (e.g., design inspection efficiency) that can be calibrated to reflect the effects of different interventions (e.g., through team training or process improvements) on the number of vulnerabilities entering operations. More generally, such a model could be used by any member of a software development ecosystem to help guide quality-cost-schedule-related decisions or in negotiations with other members. Use of the model could provide the information needed to incentivize and drive behavior toward the development and use of software with higher levels of assured performance. In future work, we hope to create a more encompassing economic model that can be calibrated to a particular organization's capabilities and vulnerability exposure and investigate how useful it can be.

Appendix A: Additional Mapping Study Results

This appendix contains detailed results from the mapping study discussed in Section 4.

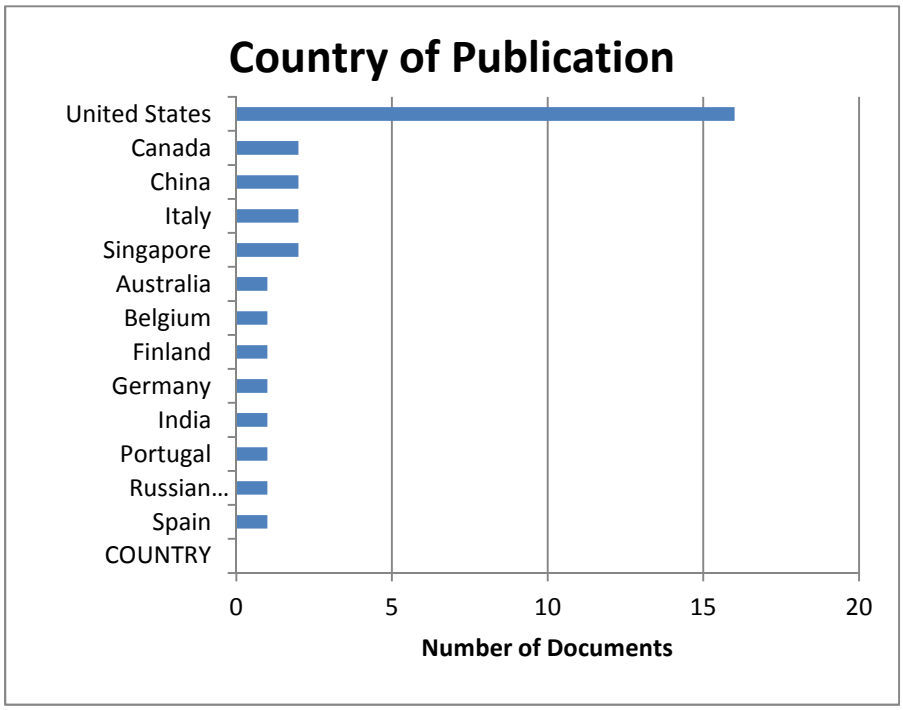


Figure 21: Country of Publication

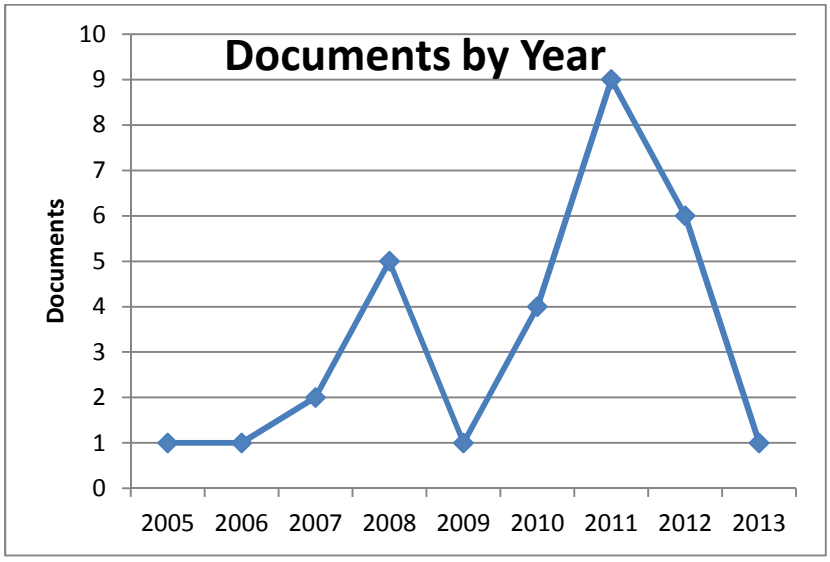


Figure 22: Number of Publications by Year of Publication

Table 6: Journals in Which Only One Article Appeared

Journal
Empirical Software Engineering
IEEE Transactions on Software Engineering
International Journal of Human Computer Studies
Jisuanji Yanjiu yu Fazhan/Computer Research and Development
Journal in Computer Virology
Journal of Network and Systems Management
Journal of Systems and Software
Journal of Systems Architecture
Journal of Universal Computer Science
Ruan Jian Xue Bao/Journal of Software
Telecommunications and Radio Engineering (English translation of *Elektrosvyaz and Radiotekhnika*)

Table 7: Conferences and Number of Papers Published

Conference Proceedings	**Number of papers**
2012 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012 - Proceedings	1
6th International Workshop on Security Measurements and Metrics, MetriSec 2010	1
Computers and Security	1
ICST 2010 - 3rd International Conference on Software Testing, Verification and Validation	1
ISARCS'12 - Proceedings of the 3rd International ACM SIGSOFT Symposium on Architecting Critical Systems	1
MetriSec'12 - Proceedings of the 4th International Workshop on Security Measurements and Metrics	1
Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011	1
Proceedings of the Annual Southeast Conference	1
Proceedings of the Fifth International Workshop on Software and Performance, WOSP'05	1
Second International Conference on Internet Monitoring and Protection, ICIMP 2007	1
Proceedings of the ACM Conference on Computer and Communications Security	3
Proceedings - International Symposium on Software Reliability Engineering	4

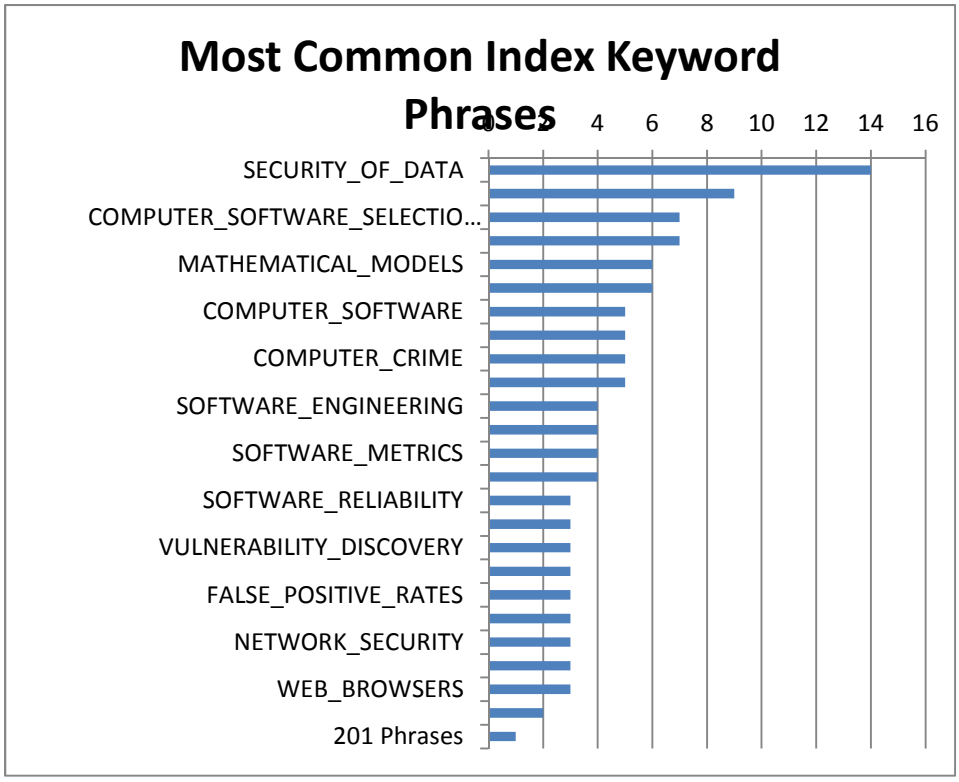


Figure 23: Frequency Count of Indexing Keyword Phrases

Table 8: Keyword Phrases Occurring Once

Attack propagation	Model	Theory of planned behavior
Classification and regression tree	Model-based performance prediction	Theory of reasoned action
Complexity metrics	Model-driven security evaluation	Vulnerability discovery model (VDM)
Composability	Multi-cycle	Vulnerability predicts
Confidentiality	N-gram analysis	Vulnerability-prone
Coupling	Open-source project	Web server
Coverage	Parametric constraints	Attack
Data mining	Predict	Attack immunity
Database	Prioritization	Automated text classification
Defect prediction	Quality of protection	Churn
Dependencies	Rc-gram analysis	Cohesion
Efficiency	Remote code execution	Complexity metrics
Empirical study	Risk evaluation	Component-based systems engineering
Estimation	Risk prediction	Cosine similarity
Execution metrics	Secure development of applications	Critical systems
Fault prediction	Security evaluation	CSF
Fault-tolerant techniques	Security mechanisms	Cyber security
Hotspots	Security metrics	Cyber-security
Inherited vulnerability	Software analysis	Empirical
Input sanitization	Software complexity	History vulnerability
Input validation and sanitization	Software vulnerabilities	Information security

Malicious software detection	Software vulnerability prediction	Performance
Metrics	SQL	Quantitative modeling
Microscopic parameters	SQL injection	

Table 9: Keyword Phrases Occurring Twice

Attack-prone	Protection	SysML
Complexity	Quantitative evaluation	Threat
Fault prediction	Randomized projections	Vulnerabilities
Information retrieval	Reliability	Vulnerability measure
Information security	Risk	Web application vulnerabilities
Organizational structure	Secrecy	Web security vulnerabilities
Payoff matrix	Security	Wikkawiki
Prediction	Software	Wordpress
Privacy	Static code attributes	

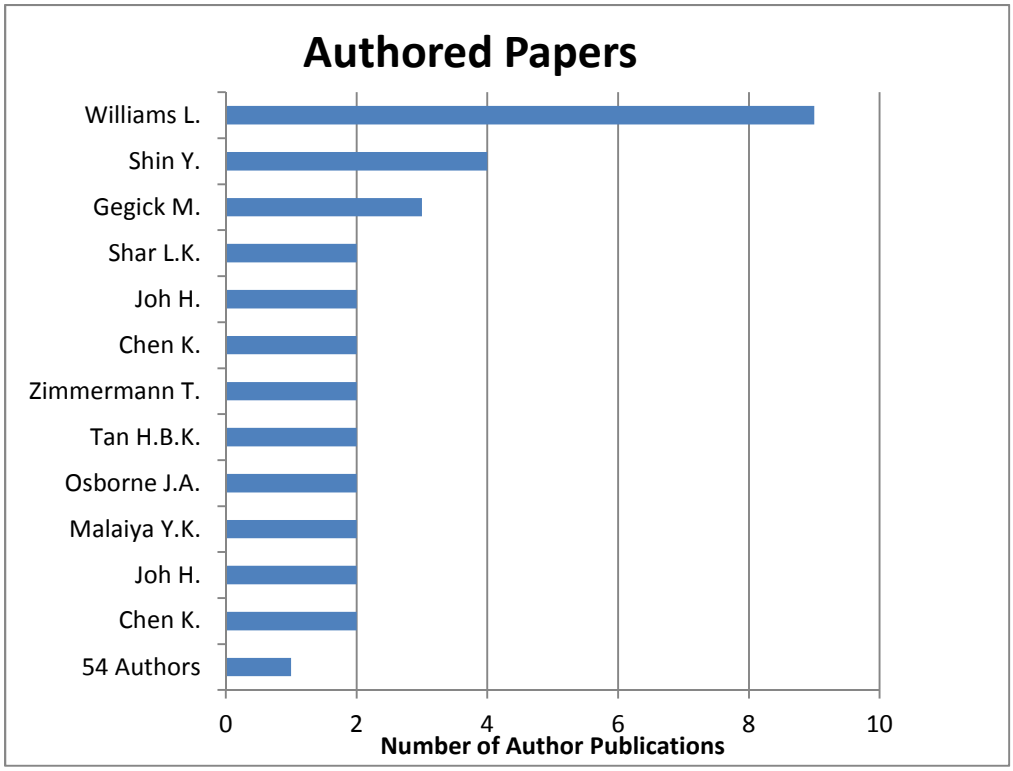


Figure 24: Frequency Count of Author Appearance in Publication Author List

Table 10: Authors Appearing in the Author List of a Single Publication

Al-Shaer E.	Merseguer J	Verissimo P.	Holm H.
Caragea D.	Mikhaleva U. A.	Vouk M.	Hovsepyan A.
Ekstedt M.	Nagappan N.	Walden J.	Lidskii E. A.
Feng D. G.	Neves N F.	Zeller A.	Neuhaus S.
Franke U.	Nie C. J.	Zhang X.F.	Nguyen V.H.
Hamed H.H.	Ou X.	Zhao X.	Nie C.
Han Z.	Rotella P.	Zulkernine M.	Rodriguez R.J.
Helenius M.	Scandariato R.	Ahmed M.S.	Sharma V.S.
Holler C.	Sommestad T.	Al-Shaer E.S.	Smith B.
Joosen W.	Su P. R.	Antunes J.	Toyssy S.
Joyce D.	Taibah M.	Atkison T.	Woo S.W.
Kankanhalli A.	Tran L.M.S.	Chowdhury I.	Woon I.M.Y.
Khan L.	Trivedi, K.S.	Grunske L.	Zhang S.
Meneely A.	Trubiani C.		

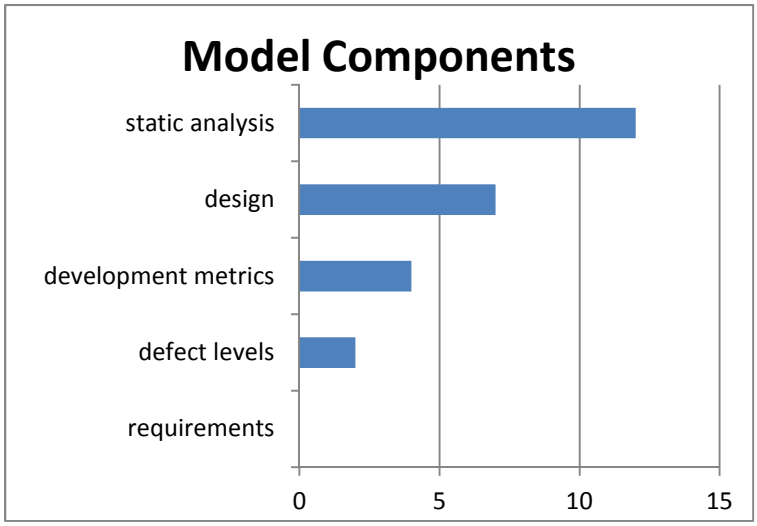


Figure 25: Frequency Count of Predictive Metric Used in Model

Appendix B: Relevant Documents for Mapping Study

Year	Document Title	Authors	Journal Title	Volume	Issue
2005	"Architecture-based analysis of performance, reliability and security of software systems"	Sharma V.S., Trivedi K.S.	*Proceedings of the Fifth International Workshop on Software and Performance, WOSP'05*		
2006	"About malicious software in smartphones"	Toyssy S., Helenius M.	*Journal in Computer Virology*	2	2
2007	"Predicting vulnerable software components"	Neuhaus S., Zimmermann T., Holler C., Zeller A.	*Proceedings of the ACM Conference on Computer and Communications Security*		
2007	"Toward the use of automated static analysis alerts for early identification of vulnerability- and attack-prone components"	Gegick M., Williams L.	*Second International Conference on Internet Monitoring and Protection, ICIMP 2007*		
2008	"Quantitative risk-based security prediction for component-based systems with explicitly modeled attack profiles"	Grunske L., Joyce D.	*Journal of Systems and Software*	81	8
2008	"Is complexity really the enemy of software security?"	Shin Y., Williams L.	*Proceedings of the ACM Conference on Computer and Communications Security*		
2008	"Prioritizing software security fortification through code-level metrics"	Gegick M., Williams L., Osborne J., Vouk M.	*Proceedings of the ACM Conference on Computer and Communications Security*		
2008	"Detection and prediction of resource-exhaustion vulnerabilities"	Antunes J., Neves N.F., Verissimo P.	*Proceedings - International Symposium on Software Reliability Engineering, ISSRE*		
2008	"Seasonality in vulnerability discovery in major software systems"	Joh H., Malaiya Y.K.	*Proceedings - International Symposium on Software Reliability Engineering, ISSRE*		
2009	"Toward non-security failures as a predictor of security faults and failures"	Gegick M., Rotella P., Williams L.	*Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*	5429	
2010	"Searching for a needle in a haystack: predicting security vulnerabilities for Windows Vista"	Zimmermann T., Nagappan N., Williams L.	*ICST 2010 - 3rd International Conference on Software Testing, Verification and Validation*		
2010	"Multi-cycle vulnerability discovery model for prediction"	Chen K., Feng D.-G., Su P.-R., Nie C.-J., Zhang X.-F.	*Ruan Jian Xue Bao/Journal of Software*	21	9
2010	"Aiding prediction algorithms in detecting high-dimensional malicious applications using a randomized projection technique"	Atkison T.	*Proceedings of the Annual Southeast Conference*		
2010	"Predicting vulnerable software components with dependency graphs"	Nguyen V.H., Tran L.M.S.	*6th International Workshop on Security Measurements and Metrics, MetriSec 2010*		
2011	"Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities"	Shin Y., Meneely A., Williams L., Osborne J.A.	*IEEE Transactions on Software Engineering*	37	6

2011	"Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities"	Chowdhury I., Zulkernine M.	*Journal of Systems Architecture*	57	3
2011	"Modeling vulnerability discovery process in Apache and IIS HTTP servers"	Woo S.-W., Joh H., Alhazmi O.H., Malaiya Y.K.	*Computers and Security*	30	1
2011	"Objective risk evaluation for automated security management"	Ahmed M.S., Al-Shaer E., Taibah M., Khan L.	*Journal of Network and Systems Management*	19	3
2011	"Using SQL hotspots in a prioritization heuristic for detecting all types of web application vulnerabilities"	Smith B., Williams L.	*Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011*		
2011	"An empirical study on using the national vulnerability database to predict software vulnerabilities"	Zhang S., Caragea D., Ou X.	*Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*	6860	1
2011	"A software vulnerability number prediction model based on micro-parameters"	Nie C., Zhao X., Chen K., Han Z.	*Jisuanji Yanjiu yu Fazhan/Computer Research and Development*	48	7
2011	"An initial study on the use of execution complexity metrics as indicators of software vulnerabilities"	Shin Y., Williams L.	*Proceedings - International Conference on Software Engineering*		
2012	"Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities"	Shar L.K., Tan H.B.K.	*Proceedings - International Conference on Software Engineering*		
2012	"Principles of prediction of information protection in the communication network"	Lidskii E.A., Mikhaleva U.A.	*Telecommunications and Radio Engineering (English translation of Elektrosvyaz and Radiotekhnika)*	71	18
2012	"Software vulnerability prediction using text analysis techniques"	Hovsepyan A., Scandariato R., Joosen W., Walden J.	*MetriSec'12 - Proceedings of the 4th International Workshop on Security Measurements and Metrics*		
2012	"Predicting common web application vulnerabilities from input validation and sanitization code patterns"	Shar L.K., Tan H.B.K.	*2012 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012 - Proceedings*		
2012	"Fault-tolerant techniques and security mechanisms for model-based performance prediction of critical systems"	Rodriguez R.J., Trubiani C., Merseguer J.	*ISARCS'12 - Proceedings of the 3rd International ACM SIGSOFT Symposium on Architecting Critical Systems*		
2013	"Can traditional fault prediction models be used for vulnerability prediction?"	Shin Y., Williams L.	*Empirical Software Engineering*	18	1

Appendix C: Template for Detailed Vulnerability Analysis

The steps for creating an economic analysis of a vulnerability mitigation story are described in this appendix.

Step 1: Identify the vulnerability. Answer "What?"

Purpose: Bring focus to the investigation; provide scope, inputs, and direction to the steps that follow.

The vulnerability should be described in sufficient detail to assist in identifying who was involved and provide some sense of the impact and probability of attacks exploiting the vulnerability. Also, key drivers of impact and probability should be mentioned (e.g., if the probability of successful exploit is increasing in time, why?).

Step 2: Identify "cast of characters." Answer "Who?"

Purpose: Multiple parties gain/lose from a vulnerability, so an early analysis step should be to identify who the participants are (or could have been). In addition, in support of identifying and quantifying what was or might have been the gain/loss, it is important to identify assets or positions that might have been at risk (e.g., mind share, market share, reputation, integrity).

Typical "cast members" include the attacker, software provider, and end user.

As appropriate, identify subclasses of these for whom some distinction may be economically important. Also, provide some description of their role or goal (you can return to this step later and add detail). Finally, provide a count (i.e., size of the class at the time the vulnerability was discovered) for parties that suffered loss or potential loss.

Step 3: Outline the main story identifying main events (timeline).

Purpose: The main "story" includes certain events or actions (which may include errors of omission as well as commission) by various parties that caused the vulnerability and related events to unfold. These events often provide insight into the knowledge and motivation of various parties (i.e., what they know and what they value). Thus, these events may help us later in identifying possible points in time (relative time) where the introduction of data feeds or decision points could have improved the economic outlook for the characters we are concerned about.

Roughly indicate when events took place, and characterize each event in sufficient detail that its impact on the cast of characters can be determined (Step 4). Knowing what took place and when facilitates some of the economic analysis (e.g., how long some asset was vulnerable to attack).

Step 4: Create "Character x Event Table" identifying the impact of each change in state (produced by the event) on the character.

Purpose: Systematically examine the impacts of each event (Step 3) on each character (Step 2) and estimate the gains/losses that can be inferred (on what they value because it enables them to pursue their goal or role).

Of course, you could also construct the transpose of such a table (i.e., Event x Character). In general, you should choose to have fewer columns than rows to allow more horizontal space for the prose and thus reduce the risk of word truncation/wraparound. You could choose whichever representation works best for you, but in order for the results of these detailed vulnerability analyses to be more comparable and to facilitate reuse across detailed vulnerability (and because time is usually placed along the x-axis) Character x Event analyses is recommended.

To assist in an economic analysis, include the count estimates from Step 2 and dates from Step 3.

Add to the Event Table two columns that characterize a) the state of things prior to the vulnerability being discovered (or whatever the first event was) and b) the state of things following exploitation and/or mitigation, which we call "aftermath." These enable identifying more explicitly the effects each event had on a character's goal, asset, or position, thereby assisting in estimating the loss/gain.

Add to the Events Table a final row ("Total Estimated Losses") to also assist in capturing the results of an economic analysis.

The last column (Aftermath) and last row (Total Estimated Losses) provide us two views into the unfolding story. In "Aftermath," we can provide a calculation of the net impact of all the losses, while in "Total Estimated Losses," we can look at the effects each event had on all the characters, with an eye to how we might make the losses more salient to the other actors in the story and identify what data/decisions might have helped (see also Step 5).

Step 5: Identify hypothetical decision points and data flows in the story.

The purpose is to help identify what could have happened as part of the economic analysis, to make more salient the long-term consequences of decisions (what is happening to actors in the other rows), to be more conscious that those decisions exist (as opposed to automatically underplaying what could be at stake), and to help identify what data could have led to a different outcome.

For example, at some point, AMD could have decided it was time to bite the bullet and release an ASLR-compatible video driver. It didn't need to take six years and a lot of embarrassment and (I presume) many millions in likely lost sales.

Note the opportunity lost though: other incompatible drivers or applications (probably) deserved attention, but until the situation with AMD resolved itself, their incompatibility was less visible.

Step 6: Review and identify externalities (impacts not considered in the previous steps) as well as assumptions/missing information that might cast a different light on the economic analysis.

For example, from the CERT description of the vulnerability mentioned in Step 5: "Software design requires known/static address space layout, doesn't support ASLR (address space layout randomization), graphics drivers are kernel on Windows so machine crashes on boot. Non-randomized locations allow attackers to use return-oriented programming (ROP) methods to bypass other runtime mitigations like DEP."

Does this imply any particular economic consequences that we have overlooked?

Appendix D: Selected Design-Related Vulnerabilities

	VU#	Vulnerability Title or Name	Vulnerability Description	Affected Vendors and Software	Date Discovered
1	458153	Graphic card drivers do not support ASLR	Software design requires known/static address space layout, doesn't support ASLR, graphics drivers are kernel on Windows, so machine crashes on boot. Non-randomized locations allow attackers to use return-oriented programming (ROP) methods to bypass other runtime mitigations like DEP.	ATI/AMD	9/20/2012
2	800113	Non-random DNS query ID or source port	DNS resolvers don't sufficiently randomize DNS query ID () or source port. Both make cache response spoofing/poisoning possible with available network and CPU resources.	Alcatel-lucent, Apple, Avaya, Blue Coat Systems, Bluecat Networks Inc, Disco, Debian, DNSmasq, F5 Networks, Norce10 Networks, Freebsd, Fujitsu, Funkwek Enterprise Communications, Gentoo Linux, HP, IBM, Infoblox, Internet Systems Consortium, Juniper, Mandriva, Microsoft, NEC Corporation, Nixu, Nominum, Nortel Networks, Novell, OpenBSD, Openwall, QNX Software System Inc, Redhat, Secure Computing Network Security Evision, Slackware Linux, Sun/Oracle, Suse Linux, Ubuntu, Wind River Systems Inc, Yamaha Corporation	3/19/2008
3	498440	TCP sequence number prediction	TCP sequence number prediction, allows man-in-the-middle (MITM) and other spoofing attacks.	FreeBSD, Fujitsu, HP, OpenBSD, SGI, Sun/Oracle	3/5/2001
4	261869	Clientless SSL VPN products break web browser domain-based security models	Clientless SSL VPN products from multiple vendors operate in a way that breaks fundamental browser security mechanisms. An attacker could use these devices to bypass authentication or conduct other web-based attacks. http://<example.com>/page1.html can access DOM objects on http://<example.com>/page2.html, but cannot access objects hosted at http://<example.net>/page.html. Many clientless SSL VPN products retrieve content from different sites, then present that content as coming from the SSL VPN, effectively circumventing browser same origin restrictions.	Cisco, AEP Networks, Checkpoint Systems, Cirtix, Juniper, Microsoft, Nortel Networks, OpenVPN Technologies, Safenet, Sonicwall, Stonesoft, Sun/Oracle	9/14/2009

	VU#	Vulnerability Title or Name	Vulnerability Description	Affected Vendors and Software	Date Discovered
5	178990	Erlang/OTP SSH library uses a weak random number generator	The Erlang/OTP SSH library implements a number of cryptographic operations that depend on cryptographically strong random numbers. Unfortunately the RNG used by the library is not cryptographically strong, and is further weakened by the use of predictable seed material. The RNG (Wichman-Hill) is not mixed with an entropy source.	Ericsson	4/21/2011
6	925211	Debian and Ubuntu OpenSSL packages contain a predictable random number generator	A weakness exists in the random number generator used by the OpenSSL package included with the Debian GNU/Linux operating system and derivative systems that causes the generated numbers to be predictable. As a result of this weakness, certain encryption keys are much more common than they should be. This vulnerability affects cryptographic applications that use keys generated by the flawed versions of the OpenSSL package. Affected keys include SSH keys, OpenVPN keys, DNSSEC keys, and key material for use in X.509 certificates and session keys used in SSL/TLS connections. Any of these keys generated using the affected systems on or after 2006-09-17 may be vulnerable. Keys generated with GNUPG or GNUTLS on the affected systems are not vulnerable because these applications use their own random number generators and not the one from the flawed version of OpenSSL.	Debian, Ubuntu	5/12/2008
7	570177	Foxit Reader vulnerable to arbitrary command execution	Foxit Reader is software designed to view Portable Document Format (PDF) files. The Adobe PDF reference supports a "launch action" that "... launches an application or opens or prints a document." Foxit Reader uses the ShellExecute function to handle PDFs that use a launch action. In some cases, Foxit Reader will not prompt the user before an application is launched with a launch action. It is also reported that the launch action can be used to launch an executable that is included in the PDF document, which results in arbitrary code execution.	Foxit Software Company	3/30/2010
8	448051	eEye Retina audit script could execute untrusted programs as root	The eEye Retina Network Security Scanner software executes various audits against target systems to conduct security vulnerability assessment testing. eEye provides audit scripts to help perform security reviews of various operating systems and applications. One audit script for Solaris, HP-UX, and IRIX systems (audit ID 2499) checks the program version by searching the /usr/local portion of the file system and executing a file with options to display version information. The script executes a program based on file name. If an attacker can place an executable file with an appropriate name in /usr/local, that file will be executed by the audit script.	eEye	9/28/2011
9	433821	DISA UNIX SRR scripts execute untrusted programs as root	DISA provides SRR scripts to help perform security reviews of various operating systems and applications. The UNIX SRR scripts check versions of various programs by searching the root file system and executing programs with options to display version information. The scripts generally use find(1) with the -exec expression primary. The scripts execute programs based on file name. If an attacker can place a file with an appropriate name on the file system, that file will be executed by the SRR script. The SRR scripts are designed to be run with root privileges.	DISA Field Security Office, Network Defense Watch Office	9/21/2009

	VU#	Vulnerability Title or Name	Vulnerability Description	Affected Vendors and Software	Date Discovered
10	357851	UPnP requests accepted over router WAN interfaces	Universal Plug and Play (UPnP) is a networking protocol mostly used for personal computing devices to discover and communicate with each other and the internet. Some UPnP enabled router devices incorrectly accept UPnP requests over the WAN interface. "AddPortMapping" and "DeletePortMapping" actions are accepted on these devices. These requests can be used to connect to internal hosts behind a NAT firewall and also proxy connections through the device and back out to the internet. Additional details can be found in Daniel Garcia's whitepaper, "Universal plug and play (UPnP) mapping attacks". A list of devices reported to be vulnerable can be found on the UPnP hacks website.	Canyon-tech, Edimax Computer Company, Linksys, Sitecom, Sweex, Technicolor, Zyxel	9/6/2011
11	649219	SYSRET 64-bit operating system privilege escalation vulnerability on Intel CPU hardware	Some 64-bit operating systems and virtualization software running on Intel CPU hardware are vulnerable to a local privilege escalation attack. The vulnerability may be exploited for local privilege escalation or a guest-to-host virtual machine escape. A ring3 attacker may be able to specifically craft a stack frame to be executed by ring0 (kernel) after a general protection exception (#GP). The fault will be handled before the stack switch, which means the exception handler will be run at ring0 with an attacker's chosen RSP causing a privilege escalation.	Citrix, FreeBSD, Intel, Joyent, Microsoft, NetBSD, Oracle, Redhat, Suse, Xen	5/1/2012
12	723308	TCP may keep its offered receive window closed indefinitely (RFC 1122)	Part of the transmission control protocol (TCP) specification (RFC 1122) allows a receiver to advertise a zero byte window, instructing the sender to maintain the connection but not send additional TCP payload data. The sender should then probe the receiver to check if the receiver is ready to accept data. Narrow interpretation of this part of the specification can create a denial-of-service vulnerability. By advertising a zero receive window and acknowledging probes, a malicious receiver can cause a sender to consume resources (TCP state, buffers, and application memory), preventing the targeted service or system from handling legitimate connections.	Checkpoint, Cisco, Extreme Networks, Force10, HP Linux Kernel Archives, Microsoft, Redhat, Sun/Oracle, SCO Group	10/1/2008
13	132419	Apple QuickTime "file: URL" arbitrary code execution	A URL handling issue exists in QuickTime's handling of file: URLs. This may allow arbitrary applications and files to be launched when a user plays maliciously crafted QuickTime content in QuickTime Player.	Apple	6/10/2008
14	127185	Apple Safari automatically executes downloaded files based on Internet Explorer zone settings	Apple Safari automatically executes downloaded files based on Internet Explorer zone settings, which can allow a remote attacker to execute arbitrary code on a vulnerable system.	Apple	6/9/2008
15	889195	RuggedCom Rugged Operating System (ROS) contains hard-coded user account with predictable password	RuggedCom Rugged Operating System (ROS), used in RuggedCom network infrastructure devices, contains a hard-coded user account named "factory" that cannot be disabled. The password for this account is based on the device's MAC address and can be reverse engineered easily (CWE-261: Weak Cryptography for Passwords).	Ruggedcom	2/3/2012

	VU#	Vulnerability Title or Name	Vulnerability Description	Affected Vendors and Software	Date Discovered
16	913483	Quantum Scalar i500, Dell ML6000 and IBM TS3310 tape libraries web interface and pre-configured password vulnerabilities	The Quantum Scalar i500, Dell ML6000 and IBM TS3310 tape libraries contain pre-configured passwords for certain accounts that are considered to be weak and could be exploited allowing an attacker user access.	Quantum, Dell, IBM	11/16/2011
17	632633	Wyse Simple Imager (WSI) includes vulnerable versions of TFTPD32	Wyse Simple Imager (WSI) includes older versions of TFTPD32 that contain publicly known vulnerabilities. An attacker could exploit these vulnerabilities to potentially execute arbitrary code on the system running WSI and TFTPD32.	TFTPD32, Wyse	7/23/2009
18	759307	Adobe Acrobat and Reader U3D memory corruption vulnerability	Adobe Reader supports two primary formats for 3D content in PDF documents: U3D and PRC. U3D support is accomplished via the Right Hemisphere 3DIF Import filter, which is provided by the 3difr.x3d file. This U3D parser contains a vulnerability that can result in arbitrary code execution.	Adobe	12/7/2011
19	723755	WiFi Protected Set-up (WPS) PIN brute force vulnerability	WPS is a computing standard created by the WiFi Alliance to ease the setup and securing of a wireless home network. WPS contains an authentication method called "external registrar" that only requires the router's PIN. By design this method is susceptible to brute force attacks against the PIN.	Belkin, Buffalo, Cisco, D-link, Linksys, Netgear, Technicolor, tp-link, Zyxel	11/10/2011
20	878044	SNMPv3 improper HMAC validation allows authentication bypass	SNMP can be configured to utilize version 3, which is the current standard version of SNMP. SNMPv3 incorporates security features such as authentication and privacy control, among other features. Authentication for SNMPv3 is done using keyed-Hash Message Authentication Code (HMAC), a message authentication code calculated using a cryptographic hash function in combination with a secret key.	Ecoscentric, Extreme Networks, Global Technology Associates, Internet Initiative Japan, Juniper Networks, Netsnmp, Network Appliance, Nokia, Redhat, SNMP Research, Sun/Oracle	5/24/2008
21	903934	Hash table implementations vulnerable to algorithmic complexity attacks	Some programming language implementations do not sufficiently randomize their hash functions or provide means to limit key collision attacks, which can be leveraged by an unauthenticated attacker to cause a denial-of-service (DoS) condition.	Apache, Microsoft, Ruby, PHP	10/31/2011
22	864643	SSL 3.0 and TLS 1.0 allow chosen plaintext attack in CBC modes	A vulnerability in the way the SSL 3.0 and TLS 1.0 protocols select the initialization vector (IV) when operating in cipher-block chaining (CBC) modes allows an attacker to perform a chosen-plaintext attack on encrypted traffic.	Google, Microsoft, Mozilla, Opera	9/26/2011
23	713878	Microsoft Internet Explorer does not properly validate source of redirected frame	Microsoft Internet Explorer (IE) does not adequately validate the security context of a frame that has been redirected by a web server. An attacker could exploit this vulnerability to evaluate script in different security domains. By causing script to be evaluated in the local machine zone, the attacker could execute arbitrary code with the privileges of the user running IE.	Microsoft	6/9/2004

	VU#	Vulnerability Title or Name	Vulnerability Description	Affected Vendors and Software	Date Discovered
24	928795	Netgear FVS318N router default remote management vulnerability	Netgear ProSafe Wireless-N 8-port Gigabit VPN Firewall FVS318N router allows remote (WAN) internet users access to the administrator web interface of the device by default.	Netgear	1/13/2012
25	520827	PHP-CGI query string parameter vulnerability	PHP-CGI-based setups contain a vulnerability when parsing query string parameters from PHP files.	PHP	2/20/2012
26	837092	InstallShield / Macrovision / Acresso FLEXnet Connect insecurely retrieves and executes scripts	Acresso FLEXnet Connect executes scripts that are insecurely retrieved from a remote web server, which can allow a remote, unauthenticated attacker to execute arbitrary code on a vulnerable system.	Acresso Software, Corel, IBM, Installshield, Macrovision, Roxio	2/5/2008
27	636312	Oracle Java JRE 1.7 Expression.execute() and Sun-Toolkit.getField() fail to restrict access to privileged code	The Java JRE plug-in provides its own security manager. Typically, a web applet runs with a security manager provided by the browser or Java Web Start plugin. Oracle's document states, "If there is a security manager already installed, this method first calls the security manager's checkPermission method with a RuntimePermission("setSecurityManager") permission to ensure it's safe to replace the existing security manager. This may result in throwing a SecurityException." Oracle Java 1.7 provides an execute() method for Expression objects, which can use reflection to bypass restrictions to the sun.awt.SunToolkit getField() function, which operates inside of a doPrivileged block. The getField() function also uses the reflection method setAccessible() to make the field accessible, even if it were protected or private. By leveraging the public, privileged getField() function, an untrusted Java applet can escalate its privileges by calling the the setSecurityManager() function to allow full privileges, without requiring code signing. Both the Oracle JRE 1.7 and the OpenJDK JRE 1.7 are affected.	Oracle	8/27/2012
28	707254	UTC Fire & Security Master Clock contains hardcoded default administrator login credentials	UTC Fire & Security GE-MC100-NTP/GPS-ZB Master Clock have default administrator login credentials that cannot be modified by an administrator.	GE, UTC Fire and Security	1/4/2012

	VU#	Vulnerability Title or Name	Vulnerability Description	Affected Vendors and Software	Date Discovered
29	362332	Wind River Systems VxWorks debug service enabled by default	Some products based on VxWorks have the WDB target agent debug service enabled by default. This service provides read/write access to the device's memory and allows functions to be called.	3com, Actelis Networks, Alcatel-Lucent, Allied Telesis, Alvarion, AMX, Aperto Networks, Apple, Arris, Avaya, Broadcom, Ceragon Networks, Cisco, D-link, Dell, Digicom, Draytek Corp, EMC, Enablence, Entrasys Networks, Epson, Ericson, Fluke Networks, Foundary Networks, Gilant Network Systems, Guangahou Gaoke Communications, HP, Huawei, Intel, Iwatsu Voice Networks, Keda Communications, Knovative, Lenovo, Lutron Electronics, Paipu Communications Technology, Mitel Networks, Motorola, Netgear, Nokia, Nortel, Polycom, Proxim, Rad Vision, Ricoh, Rockwell Automation, SFR, Shoretel Communications, Siemens, SMC Networks, Trendnet, Tut Systems, Wind River Systems, Xerox	6/10/2010
30	977312	Johnson Controls CK721-A and P2000 remote command execution vulnerability	Johnson Controls CK721-A and P2000 products contain a remote command execution vulnerability that may allow an unauthenticated remote attacker to perform various tasks against the devices.	Johnson Controls	5/30/2012

Appendix E: Detailed Vulnerability Analysis: SYSRET

SYSRET 64-bit Operating System Privilege Escalation Vulnerability on Intel CPU Hardware (Vulnerability Note VU#649219)

Description

Some 64-bit operating systems and virtualization software running on Intel CPU hardware are vulnerable to a local privilege escalation attack. The vulnerability may be exploited for local privilege escalation or a guest-to-host virtual machine escape. A ring3 attacker may be able to specifically craft a stack frame to be executed by ring0 (kernel) after a general protection exception (#GP). The fault will be handled before the stack switch, which means the exception handler will be run at ring0 with an attacker's chosen RSP causing a privilege escalation.[9]

Many OS and hypervisor vendors using Intel x86-64-based[10] processors as a platform were affected:

The CERT KB description of this vulnerability (whose first paragraph appears above) is interesting as it includes snippets of the security advisories/bulletins from multiple affected OS and Virtual OS providers including Xen, NetBSD, FreeBSD, and Microsoft Windows 7. This gives a sense of the impact of this vulnerability: multiple OS vendors (including Linux at an earlier time) and virtualization software providers (e.g., Xen, but not VMWare) were affected.

Additional Characterizations of the Vulnerability

- Xen blog from George Dunlap[11]
- Paper from the researcher, Rafal Wojtczuk, who co-discovered the vulnerability[12]

The description by Rich Gibbs is particularly succinct and clear: [13]

> *…security vulnerability [affects] 64-bit operating systems or virtual machine hypervisors running on Intel x86-64 CPUs…The vulnerability means that an attacker might be able to execute code at the same privilege level as the OS or hypervisor.*
>
> *The x86-64 architecture was originally developed by AMD, with the aim of producing a 64-bit CPU that was backward-compatible with the 32-bit IA32 architecture, as implemented in, for example, Intel's Pentium processor. The vulnerability exists because of a subtle difference between AMD's implementation and Intel's…I will attempt a brief summary here.*
>
> *Whether one is running a standard operating system, such as Linux or Windows, or a virtual machine hypervisor, such as Xen, a mechanism is needed to [rapidly] switch [back and forth] from an application, which runs with limited privileges, to the OS or hypervisor, which typically has no restrictions…The most commonly-used mechanism on the x86-64 platform uses a pair of instructions, SYSCALL and SYSRET. The SYSCALL instruction does the following:*

---

9 http://www.kb.cert.org/ vulnerabilities/id/649219

10 In this detailed vulnerability analysis, x86-64 will be used to refer to both the 64-bit instruction set standardized by AMD and downward compatible with the x86 32-bit and 16-bit legacy instruction sets as well as the AMD and Intel 64-bit processors manufactured to support that instruction set. Also, see: https://en.wikipedia.org/wiki/X86-64.

11 http://blog.xen.org/index.php/2012/06/13/the-intel-sysret-privilege-escalation/

12 http://media.blackhat.com/bh-us-12/Briefings/Wojtczuk/BH_US_12_Wojtczuk_A_Stitch_In_Time_WP.pdf

13 http://richg74.wordpress.com/2012/06/19/us-cert-intel-cpu-vulnerability/

- *copy the instruction pointer register (RIP) to the RCX register*[14]
- *change the code segment selector to the OS or hypervisor value*[15]

A SYSRET instruction does the reverse; that is, it restores the execution context of the application. (There is more saving and restoring to be done — of the stack pointer, for example — but that is the responsibility of the OS or hypervisor.)

The difficulty arises [in part] because the x86-64 architecture does not use 64-bit addresses; rather, it uses 48-bit addresses. This gives a 256 terabyte virtual address space, which is considerably more than is used today. The processor has 64-bit registers, but a value to be used as an address must be in a canonical form (see the Xen blog post for details); attempting to use a value not in canonical form results in a general protection (#GP) fault.

The implementation of SYSRET in AMD processors effectively changes the privilege level back to the application level before it loads the application RIP. Thus, if a #GP fault occurs because the restored RIP is not in canonical form, the CPU is in application state, so the OS or hypervisor can handle the fault in the normal way. However, Intel's implementation effectively restores the RIP first; if the value is not in canonical form, the #GP fault will occur while the CPU is still in the privileged state. A clever attacker could use this to run code with the same privilege level as the OS.

Intel says that this is not a flaw in their CPU, since it works according to their written spec. However, since the whole point of their implementation was to be compatible with the architecture as defined originally by AMD[16] *this seems a bit disingenuous.*

Quoting from Rafal Wojtczuk,

The root cause of the vulnerability is: on some 64bit OS, untrusted ring3 code can force the kernel to execute sysret instruction that would return to a non-canonical address. On Intel CPUs, this results in an exception raised while still in ring0[17]*. This exception cannot be handled safely.*

How to Exploit

VUPEN formally developed exploits for Windows Intel 64-bit and XEN, respectively:

- http://www.vupen.com/blog/20120806.Advanced_Exploitation_of_Windows_Kernel_x64_Sysret_EoP_MS12-042_CVE-2012-0217.php
- http://www.vupen.com/blog/20120904.Advanced_Exploitation_of_Xen_Sysret_VM_Escape_CVE-2012-0217.php

---

14 Which now points to where the next instruction is expected to be in "user land."

15 Typically, changing it from ring3 to ring0 (i.e., elevating the privilege to same as the kernel in preparation for executing the system function/service).

16 "Historically, AMD has developed and produced processors patterned after Intel's original designs, but with x86-64, roles were reversed: Intel found itself in the position of adopting the architecture which AMD had created as an extension to Intel's own x86 processor line." (Again, for more detail, see Wikipedia entry on x86-64.)

17 And with at least one CPU register whose content (going into the system call) is under end user control, the stack (RSP) register, which can be "set (in advance of the system call) to any value he/she wants to in…memory, and get the hardware (in delivering a #GP) [to overwrite it with the exception record (saved registers and state)]. This can in turn be leveraged into a full exploit.

Impacts and Mitigation

Clearly, many OS and hypervisor vendors with considerable market presence were affected. Multiple parties could have prevented the vulnerability as Intel's SDM is very clear on the behavior of SYSRET (and not every x86-64-based OS or hypervisor was affected) by, for example, adopting a safer transition back to the application following a SYSCALL. While originally noted and reported by the Linux community back in 2006, the vulnerability was characterized and easily dismissed as a Linux-specific issue. "This is likely the reason why developers of other OS have not noticed the issue, and they **remained exploitable for six years**" (From Rafal Wojtczuk link; emphasis added.) Intel could also have prevented the vulnerability by not introducing a dangerous reinterpretation of how to return from a rapid system call.

So how difficult was it to mitigate VU#649219?

Reading some of the references above and considering the short time window where all seem to be fixed (from April to June 2012) might give the impression that the vendor only needed to find (for its OS or hypervisor) a different, safer way to handle SYSRET (e.g., return other than through SYSRET or check for a canonical address), but doing so is not straightforward. That perhaps the same patch/approach might not work for all affected OS can be seen in the different ways the vulnerability can be exploited for different OS. So each vendor must conduct its own careful analysis of what computing assets are at risk or can be leveraged for an exploit and carefully redesign/code system calls/returns to ensure safe transition from application to system and back again. Also, as the intent of SYSCALL/SYSRET is that these calls be reserved for something only the OS can do but for which execution performance is critical (e.g., by minimizing saving off registers, except for those actually needed by the system function being called), the OS-specific patch(es) need to be designed/coded for execution speed as well as safe transition.

One of the vendors, Xen, has been particularly revealing relative to the considerable difficulties it encountered in working with select stakeholders to diagnose, design, code, and test patches for VU#649219, including providing a detailed timeline.[18] We excerpt a non-contiguous part of the timeline below, sufficient to indicate the enormous stress that vendors can find themselves in (and the need for very careful attention to a lot of detail) when trying to mitigate such a potentially dangerous vulnerability, the repeated communications and re-clarifications required with stakeholders (and revisiting which stakeholders it needs to have "predisclosure" with—and what about the clients of those stakeholders?), as well as the technical difficulties encountered in getting patches correct the first time (which might include trying to limit the need to fork the code base to distinguish AMD from Intel CPUs):

[After many entries in the timeline summarizing discussion among stakeholders leading to setting May 10 as the embargo date for publishing information about the vulnerability and its resolution]

[18] http://lists.xen.org/archives/html/xen-devel/2012-06/msg01072.html. Henceforth, we refer to this specific timeline as the Xen timeline.

2012-05-10 08:48 Z (3h12 before planned disclosure)
We send an ad hoc message to the predisclosure list requesting a disclosure delay for XSA-7 / XSA-8. We do not include a revised disclosure date as we do not have one confirmed.

2012-05-15 A member of the predisclosure list discovers what will become XSA-9. The context was testing the fix for XSA-7.

2012-05-16 We conclude that this is due to AMD erratum #121 and involve AMD.

2012-05-16 After discussions we conclude that there is no reasonable software workaround for Xen and that instead Xen should refuse to boot on affected systems.

2012-05-18 We decide that releasing this as a separate advisory with the same embargo date will be less confusing than trying to fold it into XSA-7.

2012-05-18 While investigating this we come to the conclusion that our previous patch to fix XSA-7 was incomplete.

2012-05-22 After intense internal discussions, we decide that the original fix for XSA-7 is too fragile and conclude that it should be replaced by something more obvious.

2012-05-24 We send XSA-7 v8 (now with a cross-reference to XSA-9 and with updated simpler patch), XSA-8 v6, and XSA-9 v1 with its own separate patch, to the predisclosure list.

Parties Involved

Attacker

Exploits the vulnerability by writing custom code for an Intel x86-64-based OS that causes the OS or hypervisor when using SYSRET to either crash the system or enable the attacker to attain privileged execution. As more and more enterprise systems and consumer app data are moved to virtualization infrastructure (VI) hosting, servers and clusters and the data they house are likely to become more and more appealing as targets for attack. An example attack might leverage the vulnerability in this way: Get onto the same cluster that houses the data for a high-value target, escape to host with privilege, and poke around for the targeted data.

Intel and AMD

Intel and AMD are the world's dominant providers and co-sustainers of the most widely used processor family not just for desktop/laptop computing—but of greater relevance to our analysis of this vulnerability— also for servers: the x86-64. According to ITCandor 2013 http://www.itcandor.com/server-2012/, "The server market ($58.8 billion in 2012) is now dominated by x86-64 processing from Intel and AMD—accounting for 73.6% of revenue and 98.2% of unit shipments in 2012." Business drivers for Intel and AMD include meeting the need for downward and upward compatibility of OS and application code in the x86-64 family while taking advantage of the increased capability to put more cores and memory on a chip. AMD provided 64-bit support for x86 before Intel and thus was able to establish the de facto behavior for 64-bit processor architecture including for handling rapid system calls (Intel's implementation of SYSRET is at the heart of this story). AMD released a statement (http://www.kb.cert.org/vulnerabilities

/id/JALR-8V8LFS) indicating their CPUs were not vulnerable and had verified this with "architecture team, design team…tests."

Assuming for a moment that AMD is in no way culpable for this vulnerability, we might observe that even "innocent" players in the larger ecosystem may suffer potential reputation/market loss from a vulnerability and need to invest time to carefully craft communication that reassures their current and potential customers and users. But there's a line of argument that AMD was also somewhat culpable as the original AMD specification for SYSRET was incomplete, allowing subtly varying compliant implementations; though another line of argument is that AMD did warn about the dangers that come with non-canonical addressing.

The view of the authors of this detailed vulnerability analysis is that each party (Intel, many VI vendors, and perhaps AMD) acted on a narrowed view (limited understanding) of what was at stake, or consciously made a poor tradeoff decision, and thus were culpable. Each party's limited understanding could arguably be due to a lack of "due diligence:" acting on an overly narrow view of the ecosystem (e.g., who is affected and how), ignorance (e.g., SYSRET implementation could lead to a guest-to-host escape), or finite attention capacity (you can only consider so many things within a fixed timeframe that you have been provided). These same constraints exist for risk management and thus one could conclude that each party needed to practice better risk management, with better approaches to risk identification, analysis, and mitigation.

Affected Virtualization Infrastructure (VI) and Hypervisor Software Vendors

These include Citrix, FreeBSD, Microsoft, NetBSD, Oracle, Red Hat, Xen, and others. (Note that some of these are also VI hosts/providers, e.g., Citrix.) As explained earlier, these vendors were both victims and potential perpetrators of the vulnerability that they could have prevented. For example, OpenBSD would have been on this list had it not pursued a code "cleanup."[19] Vendor mind share, market share, and reputation were all at stake for these vendors (remember that VUPEN actually published two successful exploits after first giving time to these parties to mitigate the vulnerability).

A key example is Xen, an open-source project sustaining virtualization software for the x86_64 server platform and providing hypervisors built on Linux and Solaris kernels as well as custom kernels. Hypervisors provide services that allow multiple computer operating systems (and associated applications) to execute on the same computer hardware concurrently. Xen provided both heavily constrained workarounds and binary patches to resolve the vulnerability. The amount of effort and attention given to this vulnerability by Xen and others was nontrivial as implied by the Xen timeline. Each affected vendor has its own "security researchers," "vulnerability notifiers," and principal end-user stakeholders that become involved as it develops, tests, and refines a vulnerability mitigation approach. When a vulnerability is discovered, it is not only these roles that get involved but also some of the key developers who would otherwise be involved in designing, implementing, or verifying new product features (or mentoring others to do the same). So there also can be significant opportunity loss.

The ability to correctly respond without needing to issue a subsequent clarification or correction to the first solution (i.e., to get the solution right the first time) is a factor of the vendor's ability to

[19] http://www.openbsd.org/cgibin/cvsweb/src/sys/arch/amd64/amd64/locore.S.diff?r1=1.47;r2=1.48

correctly diagnose the vulnerability, design a solution, implement the solution, review and test the solution, clearly communicate the solution, clearly and correctly identify which machines are affected, and provide instructions on how to determine whether a system is affected. Thus, addressing vulnerabilities can be a major undertaking for a corporation, stealing its limited attention from new products and services in a large, competitive, and lucrative market.

Security Researchers

Rafal Wojtczuk (Bromium) and Jan Beulich (SUSE) are credited by VUPEN with the (re)discovery. VUPEN, after working behind the scenes with the affected vendors, published example exploits and similarly, Wojtczuk presented the vulnerability and possible ways to exploit it at Black Hat.[20] There are also many security analysts internal to the vendors (see previous section on affected VI and software vendors) who carried out their own analyses, validating the vulnerability, determining which of many products were affected, designing patches or workarounds, testing those out, and crafting communications.

Vulnerability Notifiers

These are the vulnerability analysts, often coordinating with other parties (especially the CERT Division), who "get the word out" on vulnerabilities, alerting IT organizations and end users of what the threat is and what might be done about it. This role may include responding as best as one can to questions that arise, which in turn involves some degree of obtaining the attention of security analysts and coordinating the response. Many security bloggers mentioned the vulnerability (e.g., Rich Gibbs, who was mentioned earlier).

Virtualization Infrastructure (VI) Host/Providers

For example, companies providing internet/cloud-based storage and services, such as Amazon S3 and EC2, mentioned the vulnerabilities. As implied in the previous section on affected VI and software vendors, the virtualization server market has recently become very large and diverse. A vulnerability in the virtualization products offered by one VI vendor could compromise (or be perceived as potentially compromising) a VI host/provider's server, placing client data and relationships (and eventually company reputation) at risk. A secondary factor amplifying the loss or risk to a VI host/provider is the growth in the overall cloud solution market and the number of alternative products and sources for such solutions: A serious misstep could yield significant momentum to a VI host's competitors. Therefore, this vulnerability could cause VI hosts/providers to do the following:

1. Put pressure on established VI vendors to ensure their products do not suffer from such vulnerabilities in the future, or if they do, to rapidly find a fix. As any fix will likely be scrutinized by multiple security researchers, the fix also needs to be correct, or trust in the VI vendor will quickly degrade.
2. Mitigate the risks of such vulnerabilities by utilizing multiple VI vendors (or perhaps even other VI hosts/providers), building firewalls between clusters, maintaining redundancy and multiple physical locations, developing hard-to-predict or dynamic process and data allocation and migration strategies toward gaining resilience and flexibility should future similar problems arise. Because the VI host/provider is often an intermediary between the VI vendor

[20] http://www.blackhat.com/usa/bh-us-12-briefings.html#Wojtczuk

and VI purchaser/user, the VI host/provider will carefully weigh how it should approach (e.g., stage) disclosure when it becomes aware of a vulnerability in a VI vendor product. On the one hand, the host/provider wants to appear transparent to its clients (i.e., be open about vulnerabilities and risks and thus demonstrate their concern for secure handling of client data)—and its readiness to resolve problems as they arise. But due to limitations on attention and energy, and concern over the damaging effects and risks of vulnerabilities being brought out into the spotlight for potentially harmful speculation, it may prefer that such information not be fully disclosed for discussion. This is mentioned because it is difficult to ascertain from public sources what efforts and risks VI hosts/providers bore due to their dependence on VI vendors, in contrast to Xen, a vendor who was comparatively transparent in its efforts to resolve the vulnerability. For example, while the Amazon EC2 service relies on Xen, and acknowledged as much, Amazon claimed no impact from the vulnerability without providing any justification or explanation for the claim.[21]

VI Purchasers/Users

Purchasers/users of cloud services may transfer critical business data to a VI host/provider (however, in hybrid cloud configurations, such data might be kept in a private cloud). VI purchasers/users may be affected by this vulnerability if their VI host/provider uses one of the affected products from a VI Vendor. Often, purchasers/users don't know which VI vendors' products they are dependent on. A VI host might suffer disclosure of its data if they were stored on the same server handling (or within reach of) a compromised app running as a guest on an affected VI product.

Media

Reporters, journalists, and bloggers have multiple motivations: to raise end user awareness of risk so action can be taken, gain readership, be the first to bring a story, and so forth. But whatever the motive, the effects of media attention are to bring a spotlight to bear on particular parties (Intel,VI vendors, and VI hosts/providers in this case), often driving those parties to align their behavior (or at the very least, their observable behavior) with social norms and expectations, such as doing what they can to protect the security of others. In the case of this vulnerability, certain well-regarded blogging sites and security newsletters brought unwanted attention to the vulnerability and its potential consequences (from Intel's, VI vendors,' and VI hosts/providers' perspectives) and also "innocent" parties such as OpenBSD and perhaps AMD. And VI vendors had something precious to lose: time, reputation, and market share to their competitors. The vulnerability was serious enough that it caused FreeBSD, Microsoft, Red Hat, and other VI vendors to put out their own advisories. The vulnerability also gained some prominence on the VUPEN website and a presentation at Black Hat.

The CERT Division

The CERT Division handled coordination between researchers and affected OS/hypervisor vendors. Given the level of coordination involved, the coordination from CERT was both critical and represented the tip of the iceberg relative to the amount of investigation, analysis, and double checking resolving such a vulnerability entailed. CERT has broad credibility that amplifies its influence in its roles in security research, vulnerability notification, and the news media.

[21] http://aws.amazon.com/security/security-bulletins/xen-security-advisories/)

Additional Parties

These were not included to keep the analysis simpler but are listed in this section. This vulnerability was also a privilege escalation vulnerability for Windows OS.[22] So PC end users and companies that were not using VI providers were also affected by this if they were running Windows on a 64-bit CPU. Parties selling commercial versions of Xen (e.g., Citrix, Oracle); those including Xen or other affected VI vendor products in their OS distributions (primarily Linux/Unix in various flavors); and those patching their OS to create paravirtualized versions of their OS to run on Xen or other affected VI vendor product.

Timeline

1. 2000-2006: 64-bit computing for the x86 (i.e., x86-64) becomes a reality for consumer market, first by AMD (Opteron, Athlon), then by Intel ("newer" Pentium 4, Core 2 Merom).
2. 2006: First discovery—The vulnerability was fixed for Linux with CVE-2006-0744. At the time, the Linux community involved with resolving this vulnerability considered it as a possible basis for a DOS-type attack (vs. escalating privilege to execute malicious code), and the description sounded Linux specific: "Linux kernel before 2.6.16.5 does not properly handle uncanonical return addresses on Intel EM64T CPUs…"
3. 2006-2012 (six years of "slipping through the cracks"): Vulnerability is present on many other OS/hypervisors running on Intel x86-64-based CPUs but no exploits are known.
4. April 2012: Rediscovery—Wojtczuk (and perhaps others) discovers the vulnerability. CERT learns of this and begins notifying various possibly affected vendors of the vulnerability. Vendors assess their vulnerability, find, test, mitigate, and so forth. Actually, a lot is involved in investigating possible vulnerabilities as described in the Xen timeline mentioned previously.
5. June 12, 2012: CERT advisory—After vendors have communicated back to CERT that they have achieved some level of preparedness and mitigation, CERT releases vulnerability note VU#649219. In the days that follow, further vendor-specific mitigation is announced and testing is performed.
6. Aftermath (See following table for details.)

[22] http://technet.microsoft.com/en-us/security/bulletin/MS12-042

SYSRET Event Table

	2000-2006 *(General state leading to vulnerability discovery)*	***First discovery (2006)***	***6 Years of "slipping through the cracks" (until early 2012)***	***Rediscovery (April 2012)***	***CERT VU# posted (June 12, 2012)***	***Aftermath*** *(General state following vulnerability mitigation)*
Attacker	Looking for vulnerabilities to exploit in new 64-bit x86-based architectures for profit, notoriety, or to serve a cause.	Attackers might not recognize the potential seriousness of the Linux-discovered vulnerability and its applicability to other x86-64 OS.	See at left.	Possibly until the CERT announcement, no attacker knew of or exploited this vulnerability; on the other hand, some may have quietly exploited it for some time.	With the CERT announcement, some attackers would be motivated to take a closer look.	Except for undetected attacks prior to CERT's announcement, the time window for new attacks is generally short, particularly for high-value targets.
Intel and AMD **(Sources for x86-based 64-bit CPUs competing for OEM attention)**	AMD grabbed mind share if not market share by being the first to provide downward-compatible 64-bit CPU support for x86. De facto standard emerges around its instruction set. But AMD's specification for SYSRET is imprecise, allowing too much variation in implementation. Pressure from x86 developers drives Intel to mostly adopt AMD's solution, but Intel's different implementation of SYSRET is unnoticed or tolerated (Intel might actually have preferred its implementation for subtle performance reasons [SYSRET is intended for rapid system calls].)	The detailed sequencing in Intel's implementation of SYSRET is inconsistent with AMD's and potentially dangerous. With the vulnerability discovery in the Linux community—if Intel learned of this—Intel had opportunity to investigate, acknowledge, and resolve (or at least moderate the severity of the problem). Later, Intel will argue that it is the vendor's responsibility to read the SDM (developer manual), which is explicit and correct relative to how Intel's SYSRET behaves.	Perhaps, over time, Intel took its interpretation of SYSRET to be every bit as "de facto" as AMD's. Even if it considered resolving the vulnerability, it might have seen the steps to do so as fraught with risk and require an admission of error (e.g., updating the SDM) for something that was arguably a problem owned and resolvable by multiple parties ("We've done our part, it's now someone else's problem").	Following rediscovery, it is unclear whether Intel was involved in the analysis of possible impacts and resolution. Interestingly, Xen reached out to AMD (Xen timeline); unclear if it reached out to Intel. Intel's reputation and integrity takes some bruising from those investigating the vulnerability. The media somewhat amplified the attribution of the vulnerability to Intel by how they characterized the vulnerability (e.g., in article titles). AMD also felt some need to explain.	The CERT announcement, though carefully worded, helps place Intel in the spotlight. Its defense gets more thoroughly tested. It would have been easier for everyone if six years before Intel had provided a fix by updating the SDM to provide a more cautious prologue (coding) for safe SYSRET use. It is also possible that Intel and AMD underestimated the potential consequences of this difference in SYSRET implementation.	Competition with AMD, heated by AMD's leap into 64, was the main psychological driver (remember Intel's incentives to OEMs to use Intel). OS and VI vendors' needs for clarity on SYSRET implementation were not fully addressed by anyone. Taking any (direct) steps to resolve were perhaps fraught with risks. But also, there was a lack of clarity as to who owned this problem; Intel had already tried to be transparent (explicitness in SDM) and thus could say it saw this as someone else's responsibility. So Intel could "stay the course."

	2000-2006 *(General state leading to vulnerability discovery)*	***First discovery (2006)***	***6 Years of "slipping through the cracks" (until early 2012)***	***Rediscovery (April 2012)***	***CERT VU# posted (June 12, 2012)***	***Aftermath*** *(General state following vulnerability mitigation)*
Affected OS and Hypervisor (VI) Vendors **(About 50-100 separate products and versions?)**	64-bit CPUs are a way to offer better performance and a competitive edge. How can I best leverage what Intel and AMD offer or can offer for x86-64? There is more focus on learning the new architecture than in identifying and evaluating associated risks (multiple modes, canonical addressing, etc.). AMD is first with x86-64 instruction set—VI vendors put pressure on Intel to conform to the de facto AMD standard, thereby reducing their product complexity and costs, and ultimately, growing the 64-bit ecosystem faster, enabling support for growing server and VI markets.	Outside Linux, no one seems aware of vulnerability. With increasing x86-64 ubiquity, "minor" differences in AMD and Intel implementations of x86-64 are increasingly perceived as benign and fade into status quo. Around 2005, there is a resurgence…in the use of virtualization technology among Unix and Linux server vendors, including moves into the cloud, fueling the rise of new vendors and a new market for x86-64 adoption.	See at left. In 2011, OpenBSD mitigates the inconsistency in SYSRET during code cleanup, apparently without recognizing its broader significance.	Many of these vendors have multiple affected OS (or hypervisor) versions running on different CPU versions. So there can be a lot of tedious work to determine which systems are affected and how to fix them; test the fix; and then to determine which stakeholders to involve in all this investigative activity. Key technical staff are pulled off other priority projects to ensure rapid correct resolution of the vulnerability. Vulnerability notifiers prepare vendor-specific advisories. Perhaps significant opportunity loss for new products, mentoring staff, and detecting or resolving other problems.	The mitigation involves a lot of communication and coordination with stakeholders, clients, and users, including help with issues arising during patching, configuring, etc. for each affected version of the VI vendor's products. A possible reflection of the complexity of the change and its idiosyncrasies from vendor to vendor is the difference in approach needed to successfully exploit the vulnerability (as implied in Wojtczuk's article). Each vendor had to carry off its own investigation and coordinate and communicate about patch installation and deployment.	Patching, sleuthing 100 versions of systems, and identifying a relatively high-confident deployable mitigation. There may have been significant loss of new product opportunities for several weeks up to a few months for the affected vendors during which competitors forge ahead. Had this vulnerability not been addressed in 2012, the next generation gaming devices (Sony PS4, Xbox One), which would be x86-64-based due to the increased market power possessed by PC game publishers who want to be able to easily publish across all platforms, might have created an additional attack vector (e.g., for credit card account fraud).

	2000-2006 *(General state leading to vulnerability discovery)*	***First discovery (2006)***	***6 Years of "slipping through the cracks" (until early 2012)***	***Rediscovery (April 2012)***	***CERT VU# posted (June 12, 2012)***	***Aftermath*** *(General state following vulnerability mitigation)*
VI Hosts and Providers (50-100 approx.)	Utility computing emerged in the late 90s with IBM, HP, and Microsoft taking the lead in supporting on-demand computing and backup storage for the enterprise, but data centers consisting of servers running x86-64 emerge as Intel and AMD triumph with commercial production of processors, setting the stage for x86-64-based VI.[23]	Era of "cloud computing" begins[24] with launch of Amazon Web Service (AWS) in 2006. Its widely used AWS service, EC2, is based on the Xen x86-64 based hypervisor. SalesForce follows suit with a cloud-based platform, Force.com, in 2007. Unclear what due diligence VI hosts were exercising before utilizing a VI vendor's products.	With the rise of more powerful hardware and reduced pricing, VI becomes a more feasible and trusted option: "The enterprise market saw a huge transformation…in IT…driven totally by consumers… [with] applications hosted on far away data centers becoming a rage… launching the era of cloud computing and services based upon an "as a service" delivery model."[25]	VI hosts and providers mostly avoid the spotlight created by the vulnerability, but behind the scenes, they were 1) putting pressure on VI vendors to verify their products 2) reassuring nervous clients 3) increasing enterprise market confidence in the security of the cloud solutions they offer through security standards certification (e.g., ISO 27001).	To compete with other VI hosts and providers, and to assuage nervousness, some VI hosts/providers create more diversified service portfolios by offering hybrid cloud solutions and services. VI hosts and providers suffered some opportunity loss addressing the vulnerability and limiting reputation damage.	The overall server/cloud market suffered some nervousness for a few weeks, with some concern about reliance on the cloud, perhaps translating to more business for private and hybrid cloud models, favoring VI hosts who were better positioned to offer such services. VI hosts probably have good operational processes to keep vendor software up to date.

[23] Utility computing has usually envisioned some form of virtualization so that the amount of storage or computing power available is considerably larger than that of a single time-sharing computer. Multiple servers are used on the "back end" to make this possible. These might be a dedicated computer cluster specifically built for the purpose of being rented out.

[24] http://sourcedigit.com/497-timeline-history-of-cloud-computing/

[25] http://sourcedigit.com/497-timeline-history-of-cloud-computing/

	2000-2006 *(General state leading to vulnerability discovery)*	***First discovery (2006)***	***6 Years of "slipping through the cracks" (until early 2012)***	***Rediscovery (April 2012)***	***CERT VU# posted (June 12, 2012)***	***Aftermath*** *(General state following vulnerability mitigation)*
VI Purchasers and Users (10-30 M. enterprises? 1.3 billion consumers [Facebook market share])	Many large and medium enterprises (e.g., banks, engage utility computing-based services). Growth of data centers continues as small-to-medium-sized enterprises increasingly conduct business over the web.	See at left. Consumer presence on the web continues to grow and results in additional growth for data centers, and the server market starts to really take off.	See at left and VI hosts and providers cell. With the rise of cloud computing for enterprises, more and more business processing and data moves to the cloud. Smartphone use "skyrockets" creating enormous demand[26] for additional data center processing of requests and services. The same could be said for geo-locating services.[27]	See at left. Consumers mostly in the dark over the vulnerability (except for savvy users) but large enterprises have security experts who quietly became aware of the vulnerability and pressured their VI host and provider source(s) for status and risk information. Perhaps some enterprises put off decisions to expand their presence in the cloud, while others contemplated moving critical data off the cloud and back to the enterprise.	Vendors moved quickly, and the time window for exploit was relatively short for a major exploit. (The most high-value targets are also the ones likely to have staff to enable them to take appropriate action, e.g., to apply patches.) Potentially a large number of mobile devices—and more broadly, most computing devices—provide data online that goes to an external server that, due to the vulnerability, could leak data, but such servers usually get regular updates.	Unpatched users are vulnerable, not just to this vulnerability but many other serious ones. User economic losses could have been very significant, but, excepting the damage done by unnoticed exploits, apparently were not.
Security Researcher (Those working for a vendor are included under "Affected Vendors")	X86-based vulnerabilities are a long-term focus of researchers given the prevalence of Windows and Linux. The emerging x86-64 architecture provides a new generation of architecture vulnerabilities to study.	Apparently the susceptibility of other OS to the vulnerability was not understood—nor were the exploits it enables (beyond DDoS). But this early and modest correction paid off for the Linux community six years later.	Perhaps researchers assumed that any significant divergences in implementing x86-64 would have been addressed long ago, and not worth the time to pursue. It apparently takes 6 years to recognize the difference is significant.	For the 1-2 who discovered the vulnerability, a path to (added) glory; for others (vendors), a very stressful journey to understand the vulnerability and its mitigation. The vulnerability can be exploited in various ways.	Wojtczuk goes to Black Hat and addresses the range of possible exploits; VUPEN gets to "show its chops." Other researchers test SYSRET behavior and confirm which systems were affected. Other differences in x86-64 draw scrutiny.	A very small number of security researchers gain credit and at least one (Wojtczuk) gains some fame. But security researchers working for vendors are perhaps stressed out and perhaps temporarily more error-prone.
CERT	Coordinating response to attacks; broadening mission to include prevention. Want to get the word out if any vulnerabilities threaten business or consumer data.	See at left.	See at left.	Security researcher contacts CERT who reaches out to possibly affected vendors. OS vendors seem prone to take this seriously.	CERT posts its advisory after it is satisfied some critical affected vendors have done what they can to mitigate.	All in a day's work (several days!) for CERT. Maybe one-month of effort ensuring notification and testing the vulnerability and mitigations?

[26] As more users are using their phones to check prices of products, book a table at a restaurant, and so forth, the service owner needs to load balance their services and the most inexpensive way to do this is by using cloud services.

[27] If 50% of your customer base is in Hong Kong, it would make sense to acquire a VI hosting provider in China to lower bandwidth costs.

	2000-2006 *(General state leading to vulnerability discovery)*	***First discovery (2006)***	***6 Years of "slipping through the cracks" (until early 2012)***	***Rediscovery (April 2012)***	***CERT VU# posted (June 12, 2012)***	***Aftermath*** *(General state following vulnerability mitigation)*
Media	Fascinated by AMD vs. Intel; promise of 64-bit computing.	Interest in how Linux defines a relatively new business model for software development, but the significance of the vulnerability goes unexplored.	Intel vs. AMD on 64-bit CPUs is yester-year's story.	Media report this vulnerability and shine a spotlight on Intel (e.g., "Intel CPU flaw is vulnerable to hacker attacks").[28]	See at left.	1) Educated end users and made them less vulnerable, 2) intensified spotlight and damage to Intel and VI vendors, 3) made VI purchasers/users concerned about reliance on vulnerable VI vendor products.
Total estimated losses	No vulnerability-related losses (of course).	Some effort to detect, correct, test within Linux community. Maybe 0.1 FTE?	OpenBSD "accidentally fixes" as part of "code cleanup." Maybe 0.1 FTE? We could find no mentions of actual exploits in the wild.	Hosts/providers might have spent additional money to set up additional mitigation options (i.e., hardware firewalls or more granular ACLs or permissions to prevent compromised virtual hosts from spreading). Vendor mitigation: $20K for 1-4 person-weeks to code and test per OS/hypervisor version * 100 versions = $2 million. Associated security researcher burnout and market loss to competitors is more difficult to estimate. Perhaps the vulnerability contributed to some continued hesitancy in including Xen in Linux distributions. The x86-64 lock on the cloud market is very solid, but perhaps the vulnerability introduced some hesitancy to future use of Intel and AMD offerings. Assumes no exploitation.	See at left.	Perhaps one way to provide a very gross estimate of opportunity loss would be to assume 100K staff reading and acting in some way on the advisory, averaging one person-day on the vulnerability = 100K person-days = 400 person-years = (at 150k per person) $60 million **opportunity loss.**

[28] See http://www.theinquirer.net/inquirer/news/2185052/intel-cpu-flaw-vulnerable-hacker-attacks and http://threatpost.com/intel-processor-sysret-vulnerability-affecting-some-64-bit-systems-061812/.

Orthogonal Defect Classification-Related Attributes

Origin/phase when injected: Two possible interpretations of where the vulnerability was injected depending on how far back one considers errors:

1. Errors in the x86-64 CPU architecture specification:
    a. AMD: the specification of SYSRET was ambiguous relative to when privilege de-escalation took place. The specification apparently also includes a general warning of the unsafeness of non-canonical addresses. However, this constraint and caution are insufficient to predict how the CPU will behave in some circumstances. The AMD specification should have been more precise in these areas, and thus one could say that the error was introduced during development of the system developer's manual (SDM). This ambiguity apparently also affected AMD because it later had to test its processors to ensure they handled SYSRET correctly.
    b. Intel: its specification of SYSRET is more precise and apparently satisfies the letter of the AMD specification, but Intel designed and implemented x86-64 for its CPUs in a way that differed from the safer (but undocumented) AMD interpretation. One can take at least two perspectives on what the error was here: i) the error was introduced during requirements development for the x86-64 CPU processor family. One source for requirements was the AMD x86-64 instruction set. Intel should have investigated the behavior of SYSRET more thoroughly (both to identify the potential danger in differing interpretations of SYSRET and the danger in its own interpretation) before implementing according to its own interpretation. Here the initial error lay not in Intel's SDM but with Intel's CPU architecture. ii) There were sound reasons for adopting a variant interpretation of SYSRET in the hardware (e.g., reuse existing CPU design, better hardware performance) that led Intel to implement SYSRET the way it did, in which case the error might lie instead with its SDM not including adequate warning of the lack of safeness in its implementation of SYSRET and guidance on how to safely use SYSRET. Such an error is actually a requirements allocation error, the error regarding safe implementation of x86-64 set would here be allocated to the SDM rather than to Intel's CPU architecture.
2. Errors in VI vendors' products: VI vendors (and associated open-source communities) failed to recognize during the development of their products some of the subtleties of the AMD and Intel platforms they were writing to (we found no mention of anything they did to mitigate any downsides arising from any SYSRET-related subtleties). This might, at first glance, seem to be an implementation or training issue—after all, OS and VI product programmers are the utilizers of the x86-64 instruction set, and thus a correct understanding of it would seem to be a focus of their training and skills development. But given the incompleteness of both AMD's (ambiguous) and Intel's (dangerous without being pointed out as dangerous) instruction set specifications, the fault arguably lies not so much with training but with developing a correct understanding of the x86-64 instruction set and its limitations—something which is not in the provided specification, and thus is more of a requirement or design issue. The consistency with which this oversight occurred across various vendors even more strongly suggests that this is a requirements or design issue. Even if it is the case—and there is no particular reason to think so—that the misunderstanding started with one open-source

code developer, the obligation falls on others reusing the code to ensure that it will work correctly in its new environment. Notwithstanding the issue of Intel's implementation of SYSRET, writing code using ambiguously specified instructions (i.e., AMD's version of SYSRET) cannot be considered a safe practice unless additional checks on the processor state leading to the call to SYSRET were introduced as a design policy and perhaps also automated to help ensure safe use of SYSRET. Again, handling SYSRET correctly in such situations is a design concern as well as a training concern (so that implementers understand the policy, the motivation for it, and any automation support).

3. Note that at some future date, perhaps AMD might also come to rely on the specification for SYSRET found in its SDM, allowing its CPU designers additional freedom in how they interpreted its behavior. The result could be a dangerous implementation of SYSRET, repeating Intel's error.
4. Perhaps there is also a trust issue here: some time has passed and no exploits have been discovered. Therefore it is probably safe to rely on this specification of SYSRET.

Trigger: The vulnerability can be triggered in different ways depending on the particular VI vendor product, but it starts through carefully crafting code to force a #GP interrupt during SYSRET prior to it de-escalating privilege. Wojtczuk outlines several ways to initiate the exploit. (The exact triggers can be very VI-vendor-product specific.)

Age: The vulnerability had been around mostly undetected for six years. Over those six years, more users and businesses became dependent on internet servers and the cloud for much of their application data and processing. Thus, there came into being more high-value targets and different ways to exploit the vulnerability, but no exploit was observed in the wild.

Source: Arguably AMD and Intel were the source (see "Origin/SLC phase when injected" above), but affected VI vendors should have exercised more care in understanding the nuances of the platform for which they were developing code.

Direct mitigation/preventative actions: Different patches and workarounds are required for different VI vendors, and to some degree, different versions of the product. Specific mitigations are mentioned by Dunlap and others:

a. VI vendors review every SYSRET call and appropriately replace with a different prologue (e.g., check RCX register for canonical address prior to issuing SYSRET; if non-canonical, exit safely).
b. AMD and Intel update their SDM documentation and ensure consistency with the way SYSRET is actually implemented in their x86-64-based CPUs.

DoD: might have become victim of exploits somewhat in proportion to its dependence on external cloud and internet servers.

Economic Considerations: What Actually Took Place vs. What Could Have Taken Place Instead

	What actually took place	What could have taken place had the vulnerability been prevented
AMD and Intel	Ambiguity in SYSRET specification and differences in SYSRET implementation persist for six years.	Back in 2004-5: AMD should have been more precise in its specification of SYSRET behavior (its implementation of SYSRET in its CPU processors is apparently safe). Intel should have recognized that its implementation of SYSRET might be unsafe and either 1) adopted a safer interpretation (e.g., what AMD actually implemented) or 2) included appropriate caveats when SYSRET might be called from a processor in a possibly (attacker-crafted) unsafe state by suggesting the system developer include an appropriate prologue in advance of the call to SYSRET to ensure a safe return to the application. In 2006 when the Linux vulnerability was discovered, Intel could have updated the SDM to make it easy and obvious for others how to invoke SYSRET safely in system calls.
VI Vendors	Didn't understand the risk that lay with the SYSRET ambiguity or difference: most VI vendors developing for Intel CPUs introduce the vulnerability. (And had AMD implemented their specification differently, this might have also resulted in unsafe situations).	The Xen blog suggests that improved vulnerability disclosure, investigation, and remediation process(es) needed, and that certainly seems true, but one should question their requirements and design processes as well (as noted in item 2 under "Origin/SLC Phase when Injected"). In 2006, VI vendors (then in existence) could have recognized the reported Linux vulnerability was a problem for them too. They could have pressured Intel (and perhaps AMD) to update SDMs and make it easier for the broader ecosystem to use their CPUs safely. More broadly, VI vendors should have recognized that SYSRET might not be safely implemented (in AMD CPUs too given the imprecision of the AMD specification) and ensured SYSRET is only called after appropriate due diligence checking of the processor state.
VI Hosts/ Providers	Apparently, trusting in VI vendor capabilities to safely handle processor exception/faulting conditions, but vulnerable for six years to potential exploits.	Could do due diligence with any VI vendors they depend on to ensure a VI vendor's products are free from such vulnerabilities. Could investigate possible risks to the internet server and cloud services they offer if such vulnerabilities escaped detection and were discovered and exploited in the wild.
Ecosystem as a whole	A potentially very significant vulnerability goes undetected (maybe unexploited too) for six years. Many players had to become involved in a short-duration time window to minimize damage to VI purchaser/user trust and confidence in internet server and cloud solutions. During that time, there was some opportunity loss and potential market loss to competitors. Some of the more sophisticated VI purchasers and users became more wary of cloud solutions and perhaps investigated hybrid clouds and use of strong encryption and other mitigations against future exploits.	Maintain sufficient level of trust in the VI ecosystem to ensure its sustained growth and enable efficient business growth and handling unexpected swings in demand.

Other possible cost calculations:

1. Relative to the VI ecosystem as a whole, we assumed 100K reading and taking some action (1 day effort on average) because of the CERT advisory. The former number may be low, but the latter number may be high.
2. Potential damage to the application developer, website developer, and IT organization confidence and trust in internet server and cloud solutions.
3. Xen was one of the VI vendors most affected given its market presence and exposure to the unsafe Intel implementation of SYSRET. There was an enormous amount of (normally behind-the-scenes communications) and furious efforts to craft, review, and test patches. What on the surface might have seemed simple was much, much more convoluted as can be observed from the Xen posted timeline of its disclosure process and what went wrong.

Closing Thoughts

When it comes to drawing lessons from our observations, it is not just the observed behavior that matters, it is also the justification one offers for one's behavior. Integrity can be thought of as how well our behavior aligns with who we say we are, and reputation as how well our integrity has held up over time. Integrity and reputation influence the degree to which a communicated purpose or justification is accepted. If all align well, then over time, we include that party among those we trust. On the other hand, the perception that a social norm has been violated can incur much loss of social capital. Transactions with the party become more taxing as we need to account for risk that what the party purports to be the case for its product or service is not so. An entity not demonstrating integrity quickly moves outside the trust circle, and it can take a long while before it returns. This is the outcome that those thought to have created a vulnerability and to be brought into the media spotlight are trying to avoid. All their actions will be scrutinized—past, present, and future—against their words. Past some point of suspicion, even the words won't immediately improve matters much. In a growing market, as is the case with servers, one of the last things you want is to not be trusted. This is why media—in its many forms—has such an important role in the rapid resolution of a vulnerability. Likewise, industries are increasingly dependent on the trust that their customers, end users, partners, providers, subcontractors, and so forth. place in commerce over the internet. This can create a sense of shared identification among all companies within the same ecosystem to work together to rapidly overcome weaknesses in the internet and software infrastructure. In such contexts, integrity and reputation become very important. Interdependence can develop as several parties work together to address vulnerabilities that threaten all their collective livelihoods or market positions. Identification and interdependence are what binds multiple parties into pursuing a shared mission.

These forces are essential in understanding the life cycle of vulnerabilities and more strategic ways to address them.

Appendix F: Detailed Vulnerability Analysis: DNS Resolvers

VU#800113: DNS resolvers don't sufficiently randomize DNS query ID () or source port. Both make cache response spoofing/poisoning possible with available network and CPU resources.

Description

The Domain Name System (DNS) is responsible for translating host names to IP addresses (and vice versa) and is critical for the normal operation of internet-connected systems. DNS cache poisoning (sometimes referred to as cache pollution) is an attack technique that allows an attacker to introduce forged DNS information into the cache of a caching name server. DNS cache poisoning is not a new concept; in fact, there are published articles that describe a number of inherent deficiencies in the DNS protocol and defects in common DNS implementations that facilitate DNS cache poisoning. The following are examples of these deficiencies and defects:

- **Insufficient transaction ID space**

  The DNS protocol specification includes a transaction ID field of 16 bits. If the specification is correctly implemented and the transaction ID is randomly selected with a strong random number generator, an attacker will require, on average, 32,768 attempts to successfully predict the ID. Some flawed implementations may use a smaller number of bits for this transaction ID, meaning that fewer attempts will be needed. Furthermore, there are known errors with the randomness of transaction IDs that are generated by a number of implementations. Amit Klein researched several affected implementations in 2007. These vulnerabilities are described in the following vulnerability notes:

  VU#484649 - Microsoft Windows DNS Server vulnerable to cache poisoning

  VU#252735 - ISC BIND generates cryptographically weak DNS query IDs

  VU#927905 - BIND Version 8 generates cryptographically weak DNS query identifiers

- **Multiple outstanding requests**

  Some implementations of DNS services contain a vulnerability in which multiple identical queries for the same resource record (RR) will generate multiple outstanding queries for that RR. This condition leads to the feasibility of a "birthday attack," which significantly raises an attacker's chance of success. This problem was previously described in VU#457875. A number of vendors and implementations have already added mitigations to address this issue.

- **Fixed source port for generating queries**

  Some current implementations allocate an arbitrary port at startup (sometimes selected at random) and reuse this source port for all outgoing queries. In some implementations, the source port for outgoing queries is fixed at the traditional assigned DNS server port number, 53/udp.

  Recent additional research into these issues and methods of combining them to conduct improved cache poisoning attacks have yielded extremely effective exploitation techniques. Caching DNS resolvers are primarily at risk—both those that are open (a DNS resolver is open if it provides recursive name resolution for clients outside of its administrative domain),

and those that are not. These caching resolvers are the most common target for attackers; however, stub resolvers are also at risk.

Because attacks against these vulnerabilities all rely on an attacker's ability to predictably spoof traffic, the implementation of per-query source port randomization in the server presents a practical mitigation against these attacks within the boundaries of the current protocol specification. Randomized source ports can be used to gain approximately 16 additional bits of randomness in the data that an attacker must guess. Although there are technically 65,535 ports, implementers cannot allocate all of them (port numbers <1024 may be reserved, other ports may already be allocated, etc.). However, randomizing the ports that are available adds a significant amount of attack resiliency. It is important to note that without changes to the DNS protocol, such as those that the DNS Security Extensions (DNSSEC) introduce, these mitigations cannot completely prevent cache poisoning. However, if properly implemented, the mitigations reduce an attacker's chances of success by several orders of magnitude and make attacks impractical.

Parties Involved

Attacker (hypothetical)

An attacker with the ability to conduct a successful cache poisoning attack can cause a name server's clients to contact the incorrect, and possibly malicious, hosts for particular services. Consequently, web traffic, email, and other important network data can be redirected to systems under the attacker's control.

DNS vendors (the software providers)

Provides DNS software. While other DNS providers were aware of this vulnerability, some chose to not address it.

CritSec

The DNS is critical to the effective operation of critical infrastructures as it provides the ability to interconnect devices dynamically. This vulnerability could allow an attacker to perform a denial of service or man-in-the-middle attack on a critical device in order to prevent or modify its behavior.

Other user

The DNS is what allows the internet to function. Without the DNS, users would need to communicate IP addresses to each other through another channel for communications. This vulnerability could be leveraged to enable man-in-the-middle attacks against users in order to compromise their systems.

Security researchers: Both the original vulnerability reporter and other security researchers begin to put the pieces of the puzzle together in hopes of understanding the core of the vulnerabilities and how to mitigate it.

Media

News organizations, media outlets, online blogs, and so forth reporting on and interviewing relevant parties.

The CERT Division

Handling coordination between researchers and affected DNS vendors.

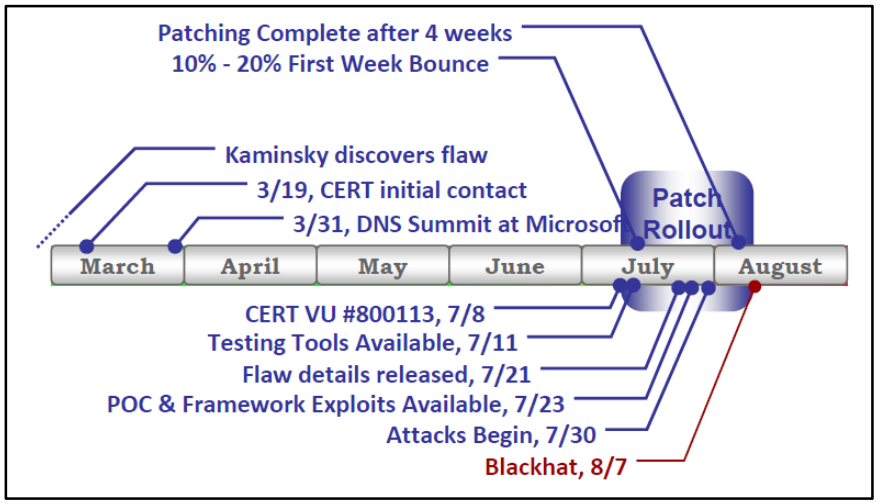
Timeline

1. Feb 2008 - Researcher Dan Kaminsky discovers a fundamental flaw in the design of the Domain Name System (DNS).
2. Mar 19, 2008 - Kaminsky sends out a notification to a small number of parties via CERT.
3. Mar 31, 2008 - At DNS Summit 2008, Kaminsky discusses
   - detailed disclosure
   - proposed solution
   - proposed patch date 2008.08.07
   - detailed release date at Blackhat
4. Jul 08, 2008 - Public announcement of the DNS flaw and proposed patch date. Kaminsky states he will fully describe the vulnerability in 30 days, allowing time for companies that operate web sites or internet service providers to put the patches in place. The CERT Division publishes its vulnerability note.[29]
5. Jul 09, 2008 - Some details are leaked about how many DNS providers are affected. But no official information has been released.
6. Jul 14, 2008 - Additional information is leaked.
7. Jul 21, 2008 – The DNS security flaw is completely leaked prior to disclosure date. “Patch DNS as fast as possible” is advised.
8. Jul 23, 2008 - DNS attack code gets published (Metasploit module).
9. Jul 24, 2008 - US-CERT updates its vulnerability note with reports of DNS exploit code in the wild; emphasizes urgent patching.
10. Jul 28, 2008 - Possible first attacks leveraging the DNS flaw have been reported.
11. Jul 30, 2008 - Reports that HD Moore (creator of Metasploit) is hacked using the DNS flaw.
12. Aug 07, 2008 - Kaminsky DNS Bug Disclosure - Dan Kaminsky releases the full vulnerability information at Blackhat 2008.

A pictorial version of the timeline is shown in the following figure.[30]

[29] http://www.kb.cert.org/ vulnerabilities/id/800113

[30] This figure was originally published at http://www.cert.org/netsa/publications/faber-OARC2008.pdf.


Patching Complete after 4 weeks
10% - 20% First Week Bounce
Kaminsky discovers flaw
3/19, CERT initial contact
3/31, DNS Summit at Microsoft
Patch Rollout
March
April
May
June
July
August
CERT VU #800113, 7/8
Testing Tools Available, 7/11
Flaw details released, 7/21
POC & Framework Exploits Available, 7/23
Attacks Begin, 7/30
Blackhat, 8/7

	Prior to 2008 *(General state of each character leading to vulnerability discovery)*	***Vulnerability discovered*** ***(****Early 2008—up to just before CERT advisory)*	***DNS vulnerability announced*** *(March 2008 - up to just before CERT advisory)*	***CERT VU# posted*** *(July 8, 2008)*	***DNS vulnerability leaked and exploit module released*** *(July 23-25 2008)*	***BlackHat presentation with full vulnerability details*** *(Aug 7, 2008)*	***Aftermath*** *(general state of each character following vulnerability mitigation)*
Attacker	DNS was typically out of scope unless an attacker could compromise the administrative account and modify the DNS records.	See at left.	Attacker knows there are flaws in the DNS but does not know all the details yet. DNS is widely used across the internet and would be a powerful attack vector.	Attempts to exploit the vulnerability might initially increase as CERT announcement might "validate" a closer look.	Attacker is now able to attack a DNS server remotely without needing to compromise an administrative account.	Full vulnerability details are released and demonstrated at Blackhat. Attackers have all the information needed to create their own exploits (including tools and demo files from presentation).	DNS cache poisoning is still happening; however, most DNS servers have been patched to address these vulnerabilities.
DNS Vendors	DNS servers built and placed on the internet/networks without understanding possible risks. There was some knowledge of flaws in the DNS among specific vendors, but no one realized the vulnerabilities were more widespread and could be combined.	Vendors privately are given details of the vulnerability. Vendors begin to realize the severity of the flaws in their software. They begin developing patches.	Vendors are called out for being vulnerable and are trying to release patches ASAP. It puts a fire under the burner for getting patches available.	See at left.	Vendors that have released patches tell end users to patch ASAP as sites are beginning to be exploited. Vendors who are still working on patches are scrambling to get a patch released.	See at left.	Vendors are more aware of the impact that shortcuts or flaws in their designs have on their customers.
CritSec ***(15-30 million)***	Push toward moving information and data to be online and shared among private and sometimes public users (facilitates access to and management of the critical infrastructure and associated services).	As more critical devices and information is placed online, we are adding additional methods attackers can use to acquire the information.	Potentially all DNS servers could be compromised. Patches needed to be deployed ASAP before vulnerability details are released.	See at left.	Vulnerability details are released allowing all critical DNS servers to be compromised. Patches needed to be deployed ASAP.	See at left.	More users are adding additional security options such as DNSSEC and trusted zones for communicating between DNS servers. If a DNS only receives updates from one or two trusted sources, it verifies the update through a second method.
Other users (300 million)	DNS was a "black hole" that most end users didn't understand.	See at left.	Users begin to understand how the DNS works and the risks associated with the vulnerability information.	In part through the media, users are advised to read advisories and patch their DNS servers.	Potentially all DNS servers could be compromised. Users are at risk visiting typical websites.	See at left.	Potentially all DNS servers could be compromised. Users are at risk visiting typical websites.

	Prior to 2008 ***(General state of each character leading to vulnerability discovery)***	*Vulnerability discovered (**Early 2008—up to just before CERT advisory)***	*DNS vulnerability announced* ***(March 2008 - up to just before CERT advisory)***	*CERT VU# posted* ***(July 8, 2008)***	*DNS vulnerability leaked and exploit module released* ***(July 23-25 2008)***	***BlackHat presentation with full vulnerability details (Aug 7, 2008)***	***Aftermath (general state of each character following vulnerability mitigation)***
Security researcher	Other DNS vulnerabilities were found in the past: • VU#484649 - Microsoft Windows DNS Server vulnerable to cache poisoning • VU#252735 - ISC BIND generates cryptographically weak DNS query IDs • VU#927905 - BIND version 8 generates cryptographically weak DNS query identifiers	See at left.	Researcher gives basic information to the public but tries to prevent vulnerability details from being released.	Other researchers realize parts of this vulnerability were talked about before, but until that point no one had a way to exploit them.	Everyone is at risk including vulnerability researchers, at least one of whom gets his own website compromised.	With additional tools and demos released, researchers are able to assist with creating mitigation strategies for customers and sponsors.	As researchers for-hire, they can perform pen-tests for customers that include DNS attacks.
Media	Little to no coverage of the DNS.	See at left.	Tons of coverage about the DNS. Some news networks begin to make statements that the internet is at risk (partially true). Media coverage helps bring the risks of vulnerability to non-technical users.	Confirms (and deepens understanding of) the risks that they had learned about from the researcher. Allows more coverage and central information-gathering sources.	Media publishes that details have been released and urges users to patch ASAP. Helps vendors broadcast information about where patches are available.	Media covers presentation.	News coverage helped spread the word about the vulnerability and patches to users who would be unaware of the vulnerability otherwise (i.e., they do not follow US-CERT tech notes or CERT/CC vulnerability notes).
CERT	CERT has handled large scale coordination efforts before.	Researchers brings CERT into coordination effort when they realize how large affected parties are. CERT begins to work with affected parties.	CERT continues to work with researcher and affected parties drafting their vulnerability report.	Report gets published.	CERT continues to work with researcher and affected parties drafting their vulnerability report.	CERT continues to work with researcher and affected parties drafting their vulnerability report. Report continues to get updated as new information about patches are released.	News coverage helped bring more users to the US-CERT and CERT/CC websites who had not known about it before.

Orthogonal Defect Classification-Related Attributes

Origin/phase when injected: There are several possible origins, listed below.

1. Early in the vendor's design phase when the decision to use predictable ordered numbers or any available random number generator was made instead of carefully selecting a generator having sufficient entropy.
2. Early in the design phase of the particular open source software (OSS) community developing such software for web servers that gets re-packaged and bundled with services by a software vendor. In this case, one might credit the vulnerability's origin to either that OSS community or to the vendor's decision to base its business around that OSS without doing the necessary due diligence on the licensed software to ensure it was relatively impervious to threats such as low-entropy-based attacks.
3. More broadly, the injection could be credited to insufficient attacks/misuse/abuse case consideration during requirements development and design. Early in the establishment of the internet, such threats might seem (and be) easier to mitigate. As internet use expanded and the market for DNS resolvers grew, it became an increasingly tempting target for attack and difficult to mitigate (expansion favored the attacker over the ability of vendors to completely mitigate).
4. Also, as the vulnerability was discovered and analyzed in some DNS vendors' products, other DNS vendors still failed to confirm if their own products were affected. In other words, once injected, the vulnerability might still have been caught and mitigated before its exploitation but many DNS vendors failed to appreciate or act on their own products' vulnerabilities.
5. Finally, one could point to the design of the DNS system itself as lending itself to various stratagems of attack, or even go further back than that to the overall design of the open internet as inherently causing opportunities to create such vulnerabilities (as Dan Kaminsky did).

Trigger: Vulnerability is triggered by an attacker being able to predict transaction IDs at the same time as predicting source ports on the DNS server. Once attackers are able to acquire both pieces of information, they can simultaneously send a "request" and "answer" packet to the DNS server allowing them to inject malicious data into the DNS cache table.

Age: Other DNS providers began to notice there was a risk of using predictable IDs and ports around 2008.

Source: Generally speaking, 1) open-source software developers (Apache Software Foundation) or 2) a commercial DNS vendor's software developers.

Direct mitigation/preventative actions: Preventing update access of DNS tables by untrusted third parties, limiting data leakage, using a more secure method of DNS data transfer such as DNSSEC (was still being developed and ratified in 2008).

DoD: DoD runs DNS servers both private and public, which could have been affected. DoD employees are vulnerable when visiting external websites as their credentials could be compromised on the attacker's website.

Economic Considerations: What Actually Took Place V. What Could Have Taken Place Instead

	What actually took place	What could have taken place had the vulnerability been prevented
DNS vendors	Since most DNS applications are open source or freely available it is difficult to estimate loss of sales, but reputation costs would account for the majority of losses.	A DNS vendor exercising care to prevent this vulnerability would have stood out from their competition if they had addressed this vulnerability sooner.
CritSec	Exploits may have led to obtaining credentials of key staff responsible for critical infrastructures.	Critical infrastructures—and the staff that sustain and secure them—would have been at less risk from attack (e.g., by diverting those seeking to legitimately access/monitor the infrastructure [and their email] to an attacker's website). Credentials of the staff would be more secure.
Security researcher, Media, CERT		
Ecosystem as a whole		

Appendix G: Detailed Vulnerability Analysis: AMD/ASLR

VU#458153: Graphic card drivers do not support ASLR.

Description

"[AMD/ATI video card driver] Software design requires known/static address space layout, doesn't support ASLR (address space layout randomization), graphics drivers are kernel on Windows so machine crashes on boot. Non-randomized locations allow attackers to use return-oriented programming (ROP[31]) methods to bypass other runtime mitigations like DEP."

Further detail

AMD video drivers were a problem because they precluded end users (including IT organizations[32]) from running ASLR "always on," which CERT and others (e.g., Microsoft) were encouraging as a means of reducing exposure to ROP-type attack vulnerabilities. A key to the success of ROP-based attacks is having a sufficiently large code base with guessable content at known addresses for the attacker to work with. The AMD video driver code base had a large footprint (perhaps a megabyte or more,[33] and hence could contribute to a successful ROP-type attack.

This is why AMD video card drivers incompatible with ASLR were becoming an increasingly important concern:

1. Video card drivers generally increase in size (new versions incorporate new features), making more code available to successfully engineer an ROP-based attack.
2. Attackers might be drawn increasingly to try to engineer such attacks, as more time passes and the code base becomes increasingly studied and known (e.g., through leaks and OSS efforts).
3. When ASLR is not set "always on," it becomes necessary for the end user to identify which applications will need to be opted in—a conscious act requiring time, thought, skill, and effort and thus prone to error—or worse—indefinite postponement.

Parties involved

Attacker (hypothetical)

Executes an attack pattern in which the system is made to crash (e.g., via a buffer overflow) in a state allowing a return-oriented (or possibly JIT compiler code injection) type of attack. Goal: obtain full-privilege control allowing installation of malware or execution of data breach (for economic, political, or notoriety reasons). Video drivers are tempting targets for attack because they run in kernel mode with full privileges, so anyone who can leverage a vulnerability in a video

[31] For an unusual use of ROP, as well as a summary description of how it works, see http://www.faqs.org/patents/app/20120030758

[32] We could—but won't—distinguish IT organizations from the end users they support, treating both as end users. A more careful economic analysis would of course separate these two. Such a separation is essential in the case of the Wind River vulnerabilities, for example, in which 100 vendors marketed devices using VxWorks as their embedded RTOS. For this particular example, such separation may be unnecessary.

[33] http://support.amd.com/us/gpudownload/windows/legacy/Pages/legacy-radeonaiw-vista64.aspx

driver gains full privileges—not just user privileges. In this example, the attacker's existence is hypothetical but plausible, and considered as such, but his existence was assumed by some of the other cast members.

AMD (the software provider)

Provides an ATI graphics driver that does not support ASLR. Later, through CERT involvement, AMD provided an update (AMD Catalyst 12.6 = 6th month of 2012) that does support ASLR.

CritSec

Those end users of the ATI video card who need, for reasons of security risk, to run Microsoft's Enhanced Mitigation Experience Toolkit (EMET)[34] turned on with mandatory ASLR—and who have the capability to do so (relatively easy actually, there's a download and link and instructions on the Microsoft website). This includes IT departments trying to ensure protection of proprietary data on employee machines. Count: About 15-30 million; assuming 5-10% of all 300 million end users have critical security requirements for ASLR always on.

Other end users

Those with the ATI video card but for whatever reason do not run ASLR (e.g., security not of much concern, those who take other precautions, those that have infrequent internet access, those with no awareness of ASLR). Actually, not all these should be treated economically the same (e.g., those without awareness might have much to lose), but we'll simplify the economic analysis. Count: About 300 million; assuming about 1.5 billion Windows machines and 20% of them having AMD graphics cards (based on estimates found online for both number of Windows machines and AMD/ATI market share). We could reduce this number by the 15-30 million noted in #3 above, but this seems close enough.

Security researchers

The individuals who found and investigated the vulnerability or who participated in finding a solution can be a bit broadly defined. One or more of these bring the vulnerability to CERT's attention. They may also participate in validating a proposed solution.

Media

Multiple motivations: Raise end-user awareness of risk so action can be taken, gain readership, be the first to bring a story, and so forth. But whatever the motive, the effect of media attention is to bring a spotlight to bear on particular parties (AMD, the security researchers), often driving those parties to align their behavior with social norms (such as caring about the security of others) that they might otherwise be unaware of, ignore, or avoid. In the case of this vulnerability, certain well-regarded blogging sites and security newsletters brought unwanted (from AMD's perspective) attention to the vulnerability, AMD's role in it, and potential consequences. And AMD had something precious to lose: market share to Nvidia.

The CERT Division

CERT got involved and nudged AMD into resolving the vulnerability. CERT acts a bit like both security researchers and the media but is motivated to use media more to nudge the correct behavior of the software provider than to gain spotlight on itself or to grow an audience. Relative to the story of this particular vulnerability, Michael Orlando shared that CERT at this time, and others in

[34] Name changed slightly with the release of EMET V2 as described in https://blogs.technet.com/b/srd/archive/2010/07/28/announcing-the-upcoming-release-of-emet-v2.aspx.

the security community, were trying to increase vendor and end-user awareness of the value of ASLR, DEP, and so forth (under the Windows package of security enhancements known as umbrella EMET) in mitigating exploitation of vulnerabilities.

Timeline

1. Ongoing (but focusing prior to 2006): Security research community explores return-oriented programming-type attacks/exploits and approaches to defend/mitigate them over a couple of decades as computer systems grow in complexity and new avenues of attack become available.[35]
2. May 2006: Microsoft announces Windows Vista Beta 2 to feature ASLR.[36]
3. September 2009: Microsoft announces EMET 2.0 with mandatory ASLR.[37]

   The EMET 2.0 announcement summarizes Microsoft's intent for EMET 2.0: "…provides users with the ability to deploy security mitigation technologies to arbitrary applications. This helps prevent vulnerabilities in those applications (especially line of business and third party apps) from successfully being exploited. By deploying these mitigation technologies on legacy products, the tool can also help customers manage risk while they are in the process of transitioning over to modern, more secure products. In addition, it makes it easy for customers to test mitigations against any software and provide feedback on their experience to the vendor." Among those security mitigation technologies is mandatory ASLR, and this is explained (along with other security mitigation technologies) in the above announcement that includes a link to the users' guide, which explains[38] mandatory ASLR and how to take advantage of the increased protections it provides on an application-by-application basis.
4. Early 2012: Security researcher(s) notes ASLR "always on" causes Windows systems with AMD video drivers to crash. CERT also learns of this.
5. June 6, 2012: CERT posts VU#458153 that notes that AMD video drivers for Windows systems create significant vulnerabilities.[39]

   CERT's announcement served multiple inter-related purposes: 1) continuing to educate end users of the value that ASLR has for them (and more broadly, remind end users of the need to take steps to minimize exposure to vulnerabilities) 2) continue to educate vendors on the value that ASLR (and similar mitigations) has for end users of that vendor's products, and thus the need for

---

[35] We are not sure how much was actually achieved by attackers using this type of exploit but knowing where certain code is in memory provides significant automated exploit opportunities, and as the footprint of such code becomes larger, the more code there exists for exploitation via a ROP-type attack.

[36] Announcement/motivation for it appears in this Michael Howard blog: http://blogs.msdn.com/b/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx. The overall message of the blog is that Microsoft is introducing with the new version of Windows some options and features at compile time, link time, and runtime so that the applications an end user uses can have more protection from successful exploit of vulnerabilities they might contain.

[37] http://blogs.technet.com/b/srd/archive/2010/09/02/enhanced-mitigation-experience-toolkit-emet-v2-0-0.aspx

[38] The Users' Guide says this on page 16: "There is an unsafe option for the ASLR setting called "always on." This setting will force address space randomization for binaries that do not specifically support it. This setting is not visible by default due to the risk of introducing system instability. "In our tests, we encountered issues in a common use scenario where having ASLR set to "always on" would cause a system to blue screen during boot. This occurred because the address space for certain third-party video drivers was being randomized. These drivers had not been built to support this randomization and subsequently crashed, causing the whole system to crash as well.

[39] The same goes for AMD video drivers for PAX (Linux) systems (excluded from this analysis).

them to create ASLR-compatible products where possible, and 3) a strong nudge to AMD, in tight competition with Nvidia and Intel, to fix the problem.[40]

6. June 29, 2012: AMD releases Catalyst 12.6 for Windows systems that is compatible with ASLR "always on" only 23 days later. This announcement was a big issue for AMD and obtaining a solution was urgent. Validation of the solution would follow relatively quickly.
7. July 23, 2012: Update to CERT's announcement noting availability of a solution by AMD
8. **Aftermath** (See the table on the following page for details.)

[40] CERT researcher Michael Orlando states, "We have seen vendors fix something in <24 hours, and other vendors take 18 months. Three to five weeks is a safe number as it falls within our typical 45 day disclosure window."

	Prior to 2006 (General state of each character leading to vulnerability discovery)	**ASLR announced** (May 2006)	**Vulnerability discovered** (Early 2012—up to just before CERT advisory)	**CERT VU# posted** (June 6, 2012)	**AMD releases ASLR compatible driver** (June 29, 2012)	**CERT notifies solution exists** (July 23, 2012)	**Aftermath** (General state of each character following vulnerability mitigation)
Attacker	Looking for ways to use increasing knowledge of Windows systems (and applications) to design attacks for profit, notoriety, or serve a cause	PC market grows, and more code can be leveraged for ROP-based automated attacks. But announcement of ASLR suggests "time is running out" on ability to exploit.	Attackers might have been quietly exploiting such vulnerabilities. (We couldn't verify one way or the other.)	Attempts to exploit the vulnerability might initially increase as CERT announcement might "validate" a closer look.	Attackers, sensing reduced opportunity, might look elsewhere for exploits.	See at left.	Some ROP attacks end (if there were any). But perhaps some exploits continue on unpatched internet-accessible systems.
AMD	As PC market grows, AMD designs new features, increasing its video driver code base. AMD also leverages OSS community to improve cycle time and as competitive differentiator, but such actions make more of its code known.	Maybe initial evaluation concluded "little risk" from ROP-type attacks, but PC market and risk are growing. Also, Vista has a rough launch,[41] which might embolden those building on the Windows platform not to rush to comply with ASLR, etc.	See at left. Also: Engaged in competitive feature race. Maybe ASLR incompatibility not seen as sufficiently high risk, relative to beating (or keeping up with) the competition on features.[42]	Spotlight on AMD. Fear that sales will take a hit. Damage control. Urgent attention given to making drivers ASLR compatible.	Solution release. Now need to get security researchers and CERT to validate the solution and official channels to distribute the word through channels.	AMD has solved the immediate problem and resumes dealing with its more persistent, longer term existential threats: Intel and Nvidia.	Losses in sales from June 6 through July (~1 month). At 20% of 170 million/yr at 50% impact => 1.5 million lost sales of $300 => $0.45 billion; also some impact on future sales—harder to estimate.

[41] http://en.wikipedia.org/wiki/Criticism_of_Windows_Vista

[42] This could be the motivation for keeping a narrow focus during the SDLC: neglecting consideration of ROP threat (during Requirements Development/Threat Modeling) and availability of VISTA counter measures (during Design). This information corresponds to the "SDLC Phase when Injected" attribute in ODC.

	Prior to 2006 (General state of each character leading to vulnerability discovery)	**ASLR announced** (May 2006)	**Vulnerability discovered** (Early 2012—up to just before CERT advisory)	**CERT VU# posted** (June 6, 2012)	**AMD releases ASLR compatible driver** (June 29, 2012)	**CERT notifies solution exists** (July 23, 2012)	**Aftermath** (General state of each character following vulnerability mitigation)
CritSec (15-30 million)	Increasingly vulnerable to ROP-type attacks as attackers learn what kernel code/data is stored where. Decreasing confidence in Windows platform as attacks and exploits increase. Put pressure on Microsoft to resolve.	As a whole, Vista (and ASLR, DEP, etc.) seems promising but also a radical change that comes at the cost of usability and may leave key providers building on the Windows platform behind. Perhaps stay with XP and wait for Windows 7, pressuring Microsoft to get it right.	Growing interest but too many vendors not cooperating, resulting in risk from many kernel-level apps not being ASLR-ed. Windows ecosystem more aligned but with key players such as AMD not being compatible.	Perhaps add to spotlighting of AMD not going along with ASLR. Probably fewer purchases and AMD losing "favored supplier" status. (And perhaps not just for video drivers.)	AMD no longer an obstacle to operating Windows systems with ASLR "always on." Driver updates need to be applied.	See at left.	CritSec no longer as likely to be subjected to ROP-type attacks. Perhaps AMD again becoming an acceptable supplier, thereby offering a sometimes cheaper alternative ($ saved).
Other end users (300 million)	Also vulnerable but more oblivious to the nature of the risk or solution.	Mostly oblivious; perhaps seen as unnecessary. Perhaps worried about Vista being a BIG step backwards. Become as cynical about security as with Microsoft.	PCs work tolerably well and that's what matters. Oblivious to silent malware until personally affected.	For a subset of end users, the CERT advisory makes it less socially acceptable to purchase AMD. Others don't care.	Further impact on AMD purchases is reduced but may persist for some time period and even be slightly contagious.	See at left.	Updates perhaps obtained automatically. Not as much an attractive target for attackers, but some segment might still be vulnerable.
Security researcher	"Kicking the tires" to find new vulnerabilities or ways to exploit them (seeking recognition and appreciation).	Recognition of the ROP general problem achieved. Continue to look for novel exploits or defenses.	For the researcher who found the vulnerability, a gain of appreciation for where to place spotlight.	(See at left.) ROP researchers obtain validation for their research and can publish it.	See at left.	See at left.	Have gained attention and respect—and for doing what they enjoy, caused the world to be a safer place.

	Prior to 2006 (General state of each character leading to vulnerability discovery)	**ASLR announced** (May 2006)	**Vulnerability discovered** (Early 2012—up to just before CERT advisory)	**CERT VU# posted** (June 6, 2012)	**AMD releases ASLR compatible driver** (June 29, 2012)	**CERT notifies solution exists** (July 23, 2012)	**Aftermath** (General state of each character following vulnerability mitigation)
Media	Looking for the good story to educate readership and increase eyeballs—and revenue.	Might have been slow to fully appreciate. Maybe overlooked.	Probably not aware.	If brought to readers' attention, could be beneficial for readers and also for revenue.	With a solution at hand, can continue publicizing this.	See at left.	Events exploited for profit but also 1) educated end users and made them less vulnerable 2) intensified spotlight and damage to AMD.
CERT	Coordinating response to attacks; broadening mission to include prevention.	Go after vendors to help make ASLR etc. effective.	See at left.	Put AMD in the spotlight while educating end users of risk.	Before publicizing existence of the solution, CERT needs to validate that AMD release 12.6 is a solution. Perhaps under pressure to do so.	CERT expends effort to get the word out and monitors for any hiccups (new vulnerabilities revealed).	CERT has brought awareness to a problem and helped a larger community benefit from reduced likelihood of a class of exploitations.
Total estimated losses	Scale of ROP-based attacks is unknown43, but presumably monotonically increasing given growth in PC sales (targets), size of code at known locations (attack opportunities).	Economic impact of ASLR announcement is unclear. Perhaps slowly regaining trust for Microsoft products. Some vendors getting angry and tired of incompatibilities.	Security researcher only direct beneficiary at this point. Actual ROP-based attacks are unknown on Windows, but the risk is real.	AMD almost immediately begins to lose planned purchases among CritSec and perhaps others. Other laggards see need to comply.	Effect on AMD sales gradually diminishing. CritSec can achieve a level of security from automated ROP-type attacks. AMD again an alternative supplier of video card drivers.	See at left.	Missing: Analyst time worrying about ROP-type attacks but unable to quantify their likelihood?

[43] It is unclear how we might estimate "background ROP exploitation," and it perhaps continues, resulting in a further loss for those with unpatched Windows systems and ASLR not "always on." We have not heard of any successful ROP exploits on Windows systems (other than for demonstration by security researchers).

Orthogonal Defect Classification-Related Attributes

Origin/ phase when injected: Combination of getting the requirements complete (threat modeling may have neglected considering ROP-type attacks) and design (leverage newly provided security features in Vista vs. the consequences to end users and to the larger Windows ecosystem of NOT doing so). (Already mentioned in earlier steps.)

Trigger: An automated pre-designed attack could exploit a first vulnerability (e.g., a buffer overflow) and use the resulting crash to apply ROP against known (video driver) addresses to read sensitive data or deposit malware. This kind of attack is hard to engineer. (Already mentioned in earlier steps.)

Age: Emergent (i.e., initially there was no ASLR to push back on ROP-type attacks). ROP-type attacks were more of a concept than a reality initially, particularly with the initially smaller code bases that existed in earlier Windows ecoystems.

Source: In-house (vs. third party/OSS)

Direct Mitigation/Preventative actions:[44]

1. AMD ensured source code for driver did not depend on "known addresses."
2. AMD modified driver software to be compatible with ASLR.
3. AMD improved communication between AMD's driver team and AMD's tech support team.

DoD: might have 1) been/become victim of ROP attacks 2) borne additional expenses equipping PCs with video cards if, due to the vulnerability, they were to drop AMD as a supplier.

[44] Inferences based on http://blogs.amd.com/play/2012/06/28/our-driver-team-answers-the-call-once-again/

Economic Considerations: What Actually Took Place vs. What Could Have Taken Place Instead

	What actually took place	What could have taken place had the vulnerability been prevented
AMD	AMD might have **lost sales** in mid/late 2012 (back-of-the-envelope style estimation was $0.45 billion) and perhaps beyond, and in **reputation** costs.	A counterpoint: by deferring attention to security, could AMD actually have benefitted in market share for the short term? But introducing threat modeling and observing the increasing risk of ROP were not expensive or time consuming. When it came time to create an ASLR-compatible driver, it took AMD maybe 10 days for 5 staff = 0.2 FTE effort. Couldn't it have done this years before?
CritSec	The overall Windows ecosystem grew more risky as long as major providers (AMD) created ASLR-incompatible drivers. The eventual mission disruptions and costs could have been enormous (and maybe were enormous in some attacks, e.g., at least one compiler was actually created that generated ROP-type instructions to create arbitrary executable code out of existing kernel code at known addresses).	CritSec no longer as likely to be subjected to ROP-type attacks. Perhaps AMD again becoming an acceptable supplier, thereby offering a sometimes cheaper alternative ($ saved). Perhaps other benefits reaped from a safer ecosystem as further described in the rows below.
Security researcher, media, CERT	Attention of all three parties was diverted to deal with this non-complying major actor (AMD), and this vulnerability took a share of whatever "attention space" these actors could claim. As a consequence, time and attention from potentially millions of end users was diverted to learn about or deal with this vulnerability.	All three parties could instead deal with other vulnerabilities that were also the basis of attacks or could become the basis of attacks. Losses due to these other vulnerabilities could have been stemmed earlier.
Ecosystem as a whole	Note there was also an opportunity lost: Other incompatible drivers or applications (probably) deserved attention, but until the situation with AMD resolved itself, their incompatibility was less visible. And perhaps had AMD complied to begin with, others might have more readily followed suit.	Overall ecosystem more aligned, which has some non-linear benefits: reinforce security norms for all and stronger expectations get set for heightened attention to security.

Other possible cost calculations:

Assuming 20% of CritSec upload the new graphics driver, maybe 3 million end users spent 5 minutes each to ensure they were running the new device driver =~ 100 full time equivalent (FTE) effort =~ $20 million in costs.

And for the small number of end users actually opting to replace AMD video card with other video HW = 2.5% of CritSec ~ 300K new customizations at $400 per and 15 minutes time =~ $130 million.

200K reading CERT advisory and reflecting on what to do about it = 20 mins/reader =~ 30 FTE =~ $6 million

What a more general process preventing the vulnerability and many "similar vulnerabilities" might have entailed:

- eliciting security needs and thresholds
- threat modeling assuming attacker orchestrates attacks such as ROP
- leveraging security enhancements offered by whatever platforms you develop products for (e.g., ASLR and more broadly EMET in the case of Windows systems)

Developers should avail themselves of the security features and options offered by whatever platforms or software sources they are using; this should be the usual organizational practice enforced through reviews during design and implementation.

Implementing these new practices, deploying them to where they are needed, offering training, motivation, and support, and assuring they are performed.

Closing Thoughts

Relative to leveraging security enhancements, Michael Howard's blog implies that Microsoft gave careful consideration to what could be done differently during OS/application development, compile, link, run, and deployment time to ensure any applications running on the Windows OS platform (including many older versions) would offer attackers a significantly reduced attack surface (reduced ease of and consequences from exploitation). That is, Microsoft gave careful consideration to security design based on assumptions of how vendors, end users, and attackers would behave so that end users would be more secure. This seems to involve considerations broader than what the phrase "OS design" might normally convey, perhaps "platform design" and even "ecosystem design" better captures the focus of Microsoft's considerations. But many vendors such as AMD were not initially aligned with Microsoft or each other on the overall approach (or its need), so Microsoft's security design strategy was "broken" (as Microsoft itself recognized as implied by the quote in footnote 38) potentially undermining the security and safety of the overall PC ecosystem.

Regarding this example from a perspective of the ecosystem and human behavior, the changing of one's attitudes and values toward adopting what might be called a "security mindset" (and more generally, any change in mindset) often takes a long while to achieve. In the case of this vulnerability, what was required was enough mindsets to change across the larger ecosystem (i.e., including AMD) that Microsoft's efforts with EMET could provide improved security for end users. Thus, in many ways, with its significant pivot toward security, what needed to be considered goes well beyond what has traditionally been treated as "design" (the organization of software resources at runtime) to include how other actors (stakeholders) in the ecosystem will behave during and after the intended changes.

References

[Aslam 1996]
Aslam, T. & Spafford, E. H. “Use of a Taxonomy of Security Faults.” *Proceedings of the 19th National Information Systems Security Conference* (1996).

[Budgen 2008]
Budgen, David. "Using Mapping Studies in Software Engineering.” *Proceedings of PPIG 8* (2008).

[Catal 2009]
Catal, Cagatay & Diri, Banu. “A Systematic Review of Software Fault Prediction Studies.” *Expert Systems with Applications 36*, 4 (May 2009): 7346-7354.

[Chillarege 1992]
Chillarege, Ram; Bhandari, Inderpal S.; Chaar, Jarir K.; Halliday, Michael J.; Moebus, Diane S.; Ray, Bonnie K.; & Wong, an-Yuen. “Orthogonal Defect Classification - A Concept for In-Process Measurements.” *IEEE Transactions on Software Engineering 18*, 11 (Nov 1992).

[GAO 2013]
United States Government Accountability Office (GAO). *Cybersecurity: National Strategy, Roles, and Responsibilities Need to Be Better Defined and More Effectively Implemented* (GAO-13-187) Feb 2013.

[Garber 2012]
Garber, Lee. “Have Java’s Security Issues Gotten out of Hand?” *IEEE Computer 45*, 12 (2012): 18-21.

[Grady 1992]
Grady, Robert B. *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, Inc., (1992).

[Hall 2012]
Hall, T.; Beecham, S.; Bowes, D.; Gray, D.; & Counsell, S. “A Systematic Literature Review on Fault Prediction Performance in Software Engineering.” *IEEE Transactions on Software Engineering 38*, 6 (2012): 1276–1304.

[IBM 1999]
IBM Research Group. *Details on Orthogonal Defect Classification for Design and Code.* IBM Center for Software Engineering,1999.
http://www.research.ibm.com/softeng/ODC/DETODC.HTM

[IEEE 2009]
IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993) IEEE Standard Classification for Software Anomalies, 2009.

[Kitchenham 2007]
Kitchenham, B.A. *Guidelines for Performing Systematic Literature Reviews in Software Engineering Version 2.3*. Software Engineering Group, Keele University and Department of Computer Science University of Durham, 2007.

[Landwehr 1994]
Landwehr, C. E.; Bull, A. R.; McDermott J. P.; & Choi, W. S. "A Taxonomy of Computer Program Security Flaws." *ACM Computing Surveys 26* (1994).

[Lindsey 1989]
Lindsey, D. "Using Citation Counts as a Measure of Quality in Science: Measuring What's Measurable Rather Than What's Valid." *Scientometrics 15* (1989): 189–203.

[Macroberts 2010]
Macroberts, M. H. & Macroberts, B. R. "Problems of Citation Analysis: A Study of Uncited and Seldom-Cited Influences." *Journal of the American Society for Information Science and Technology, 61* (2010): 1–12.

[Petersen 2008]
Petersen, K.; Feldt, R.; Mujtaba, S.; Mattsson, M. "Systematic Mapping Studies in Software Engineering." *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE).* University of Bari, Italy, June 2008.

[Pickard 1998]
Pickard, L. M.; Kitchenham, B.; and Jones, P. W. "Combining Empirical Results in Software Engineering." *Information and Software Technology, 40,* 14 (1998) 811–821.

[Seaman 2008]
Seaman, Carolyn B.; Shull, Forrest; Regardie, Myrna; Elbert, Denis; Feldmann, Raimund L.; Guo, Yuepu; & Sally Godfrey. "Defect Categorization: Making Use of a Decade of Widely Varying Historical Data." *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (ESEM '08). ACM, 2008.

[Shin 2013]
Shin, Y. & Williams, L. "Can Traditional Fault Prediction Models Be Used for Vulnerability Prediction?" *Empirical Software Engineering 18* (2013): 25–59.

[Sterman 2000]
Sterman, J.D. *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Irwin/McGraw-Hill, 2000.

[Van Eeten 2008]
Van Eeten, Michel J. G. & Bauer, Johannes M. *Economics of Malware: Security, Decisions, Incentives and Externalities.* Working paper DSTI/DOC(2008)1. Organisation for Economic Cooperation and Development, May 29, 2008.

[Weber 2005]
Weber, S.; Karger, P. A.; & Paradkar, A. “A Software Flaw Taxonomy: Aiming Tools at Security.” *ACM SIGSOFT Software Engineering Notes 30* (2005) 1-7.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	May 2014	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Data-Driven Software Assurance: A Research Study	FA8721-05-C-0003

6. AUTHOR(S)

Mike Konrad, Art Manion, Andrew Moore, Julia Mullaney, William Nichols, Michael Orlando, Erin Harper

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2014-TR-010

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Software vulnerabilities are defects or weaknesses in a software system that if exploited can lead to compromise of the control of a system or the information it contains. The problem of vulnerabilities in fielded software is pervasive and serious. In 2012, Software Engineering Institute (SEI) researchers began investigating vulnerabilities reported to the SEI's CERT® Division and determined that a large number of significant and pernicious software vulnerabilities likely had their origins early in the software development life cycle, in the requirements and design phases. A research project was launched to investigate design-related vulnerabilities and quantify their effects. The Data-Driven Software Assurance project examined the origins of design vulnerabilities, their mitigations, and the resulting economic implications. Stage 1 of the project included three phases: 1) conduct of a mapping study and literature review, 2) conduct of detailed vulnerability analyses, and 3) development of an initial economic model. The results of Stage 1 indicate that a broader initial focus on secure design yields substantial benefits to both the developer and operational communities and point to ways to intervene in the software development life cycle (or operations) to mitigate vulnerabilities and their impacts. This report describes Stage 1 activities and outlines plans for follow-on work in Stage 2.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Mapping study, software design, data analysis, literature review, system dynamics model, software assurance	115

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102