Assessing Security Vulnerabilities: An Application of Partial and End-Game Verification and Validation

Edward Snead Frazier II

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

April 21st, 2010
Blacksburg, VA

Keywords: Software Security, Vulnerabilities, Assessment, Constraints, Assumptions, Access Driven VV&T.

Assessing Security Vulnerabilities: An Application of Partial and End-Game Verification and Validation

by

Edward Snead Frazier II

ABSTRACT

Modern software applications are becoming increasingly complex, prompting a need for expandable software security assessment tools. Violable constraints/assumptions presented by Bazaz [1] are expandable and can be modified to fit the changing landscape of software systems. Partial and End-Game Verification, Validation, and Testing (VV&T) strategies utilize the violable constraints/assumptions and are established by this research as viable software security assessment tools.

The application of Partial VV&T to the Horticulture Club Sales Assistant is documented in this work. Development artifacts relevant to Partial VV&T review are identified. Each artifact is reviewed for the presence of constraints/assumptions by translating the constraints/assumptions to target the specific artifact and software system. A constraint/assumption review table and accompanying status nomenclature are presented that support the application of Partial VV&T. Both the constraint/assumption review table and status nomenclature are generic, allowing them to be used in applying Partial VV&T to any software system. Partial VV&T, using the constraint/assumption review table and associated status nomenclature, is able to effectively identify software vulnerabilities.

End-Game VV&T is also applied to the Horticulture Club Sales Assistant. Base test strategies presented by Bazaz [1] are refined to target system specific resources such as user input, database interaction, and network connections. Refined test strategies are used to detect violations of the constraints/assumptions within the Horticulture Club Sales Assistant. End-Game VV&T is able to identify violation of constraints/assumptions, indicating vulnerabilities within the Horticulture Club Sales Assistant. Addressing vulnerabilities identified by Partial and End-Game VV&T will enhance the overall security of a software system.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Dr. James D. Arthur, for his continued insight and patience throughout the completion of this work. Without his guidance and direction this research effort would not have been possible.

I would like to thank my entire committee, Dr. James D. Arthur, Mr. Randolph Marchany and Dr. Joseph Tront, for sacrificing their time to serve on my committee and answering any questions I may have.

I would like to thank Mr. Randolph Marchany and Mr. Wayne Donald for allowing me to use the Virginia Tech IT Security Laboratory during the completion of this work. I would also like to thank them for allowing me to be a Research Assistant in the IT Security Lab during completion of this work.

Many thanks to my fellow researchers in the Virginia Tech IT Security Lab for their constant support throughout this work: Ben Moyers and Lee Clagget. They were always available to bounce ideas around and provide feedback.

Finally, I would like to thank my family and friends for being a constant source of motivation, support and release. Without them this work truly would not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

The introduction and evolution of the internet as a communications medium has allowed for the mass interconnection of computer systems, making the systems accessible to almost any other computer system on the network. Being able to remotely access computer systems, along with sensitive data that traverses computer networks, make networked computer systems attractive to attackers. An increased awareness in software security can help mitigate or eliminate the effects of malicious attacks.

This research addresses the issue of software security and does so by utilizing a technique called Access-Driven Verification, Validation and Testing (VV&T) [2]. More specifically, Partial and End-Game VV&T from the Access-Driven VV&T specification are examined to determine if they are a viable option for assessing the security awareness of a software system. The application of Partial and End-Game VV&T is documented in order to identify methods by which Partial and End-Game VV&T can be applied to other software systems.

Section 1.1 provides motivation for applying and analyzing the VV&T methods. Section 1.2 elaborates on the problem being explored in this research. Section 1.3 presents issues encountered in trying to find a solution to this problem. Section 1.4 explains the solution approach to this problem. Section 1.5 details the organization of the remainder of this thesis.

1.1 Motivation

As the internet becomes more entrenched into everyday life, the sophistication and frequency of malicious attacks on software systems will increase. Data obtained from a well oriented malicious attack can have as much monetary value as if the attacker had robbed a bank. Unlike software systems, banks have vaults, cameras, and even security guards protecting them from attack. While it is near impossible to implement such methods for software systems, a security policy for a software system can be enforced through programming and other external methods, i.e. a firewall. Enforcing the security policies of software systems is vital to protecting the information stored within these systems. Information stored can include usernames, passwords, social security numbers, credit card numbers, and even the name and address of the user of the system. A leak of this information can lead to loss of money or identity theft.

It is estimated that economic cost of software security flaws (vulnerabilities) is around U.S. $180 billion a year [3]. In 2008 there were more than 320 million web-based attacks detected across the Virginia Tech network [4]. These numbers are alarming statistics. The Virginia Tech network represents only a small portion of the world's networked computers. The monetary loss per year due to software vulnerabilities alone is unacceptable. This research strives to mitigate the presence of software vulnerabilities, and in turn, reduce the effect of attacks on software systems.

The importance of software security has been recognized by the United States government as the President of the United States created a 'Cyber Czar' position to head a newly created White House office on cyber security [5]. Government systems are highly networked and store vast amounts of information critical to national infrastructure. File servers maintained by the government as well as government contractors contain information such as judicial, military and economic information. Security vulnerabilities in sensitive networks like the United States government can even lead to issues of national security. In 2009, computer files containing information pertaining to helicopters used to transport the President of the United States were found on a file server in Iran [6]. Software security attacks can be deployed by one nation against another and can be just as harmful as conventional weapons. In 2009, it was reported that sustained cyber attacks against United States and South Korean government agencies were possibly carried out by North Korean forces [7].

An alternative or even additional layer of security assessment can aid a software system in conforming to its security policy. The focus of this research is to test new software security assessment methods, which have been introduced, but not yet examined, for their feasibility in identifying software vulnerabilities within a system. Documentation produced in assessing Partial and End-Game VV&T can be used for future application of these methods.

1.2 Problem Statement

Several ways to implement security within a software system exist. One method, called penetrate-and-patch, involves penetrating a system after it has been released to the public and fixing any vulnerability found with a patch [8]. This method is largely ineffective due to the time it takes to find and patch vulnerabilities, lack of patch compliance by system administrators, and new vulnerabilities that the patches themselves can introduce [8]. Identifying security vulnerabilities after a system has been developed involves using code analysis tools to check for possible vulnerabilities in the source code of the system. After identifying the vulnerabilities, the code is patched and the process starts over again.

Another way to integrate software security is to apply software security principles throughout the design of the system. Having a security based mindset when developing software can identify potential vulnerabilities before a software system is developed. Incorporating security into the Software Development Life Cycle (SDLC) has shown to be less expensive and more effective than considering security as an afterthought [9]. Having a security mindset throughout the SDLC is a sensible approach, but has been largely ignored when developing software systems.

This research looks at the Partial and End-Game VV&T methods of software security assessment. Each of these methods uses violable constraints/assumptions to help mitigate the security risk within a software system [2]. The constraints/assumptions follow the Process/Object Model of Computing, which helps in pinpointing areas of vulnerability within a software system. Partial VV&T incorporates the SDLC in the assessment of a software system's security by analyzing the design artifacts produced during the development of the system. In this way, Partial VV&T is a hybrid between incorporating security throughout the SDLC and the penetrate-and-patch method. End-Game VV&T solely involves testing the system's executable. End-Game VV&T differs from the current penetrate-and-patch methods by using test strategies derived from the constraints/assumptions.

Taking a proposed method and applying it in a real world environment is the only definitive way to validate the usefulness of a method. The goal of this research is to test the Partial and End-Game VV&T security assessment tools, and as such the constraints/assumptions, to determine if they can be used as viable software security assessment tools for software systems. In addition to determining the feasibility of Partial and End-Game VV&T, this research strives to establish and document a process by which Partial and End-Game VV&T can be applied to a system. The constraints/assumptions are studied for their feasibility in real world application to determine if they alone are appropriate for assessing the security of a software system.

1.3 Problem Specific Issues

The problem statement for this research has several accompanying issues. Issues relating to selection of a software system, Partial and End-Game VV&T, and the constraints/assumptions exist. Resolving these issues is critical in achieving a solution to the problem statement. Issues specific to the problem statement are listed below.

Constraints on software system selection: Partial and End-Game VV&T add a constraint to the selection criteria for the software system. The software system needs to have completed the development process and have an accompanying set of design artifacts. While many software systems have accompanying development artifacts, they are not always available. They are not always available because development artifacts can contain proprietary or sensitive information that the authors of a software system are unwilling to divulge.

Translating constraints/assumptions: Generic constraints/assumptions are translated to target the selected software system. Generic terminology in the constraints/assumptions is able to be directly applied to the software system to some extent. However, it is possible that both the software system and the constraints/assumptions may reference the same type of security instance, but use different terminology to describe it. This difference in terminology can lead to the masking of a violation of a constraint/assumption if the security instance is not able to be identified.

Partial VV&T application method: No method exists with which to apply Partial VV&T. Therefore, a method for the application of Partial VV&T must be formulated. A method of application for Partial VV&T must be generalized across all development artifacts and eligible software systems. The method must be generalized for all development artifacts because the artifacts available for review will not always be the same. Generalizing the method for eligible software systems will allow the method formulated in this research to be applied to additional software systems.

End-Game VV&T application method: As with Partial VV&T, there is no documented application of End-Game VV&T to a software system. However, there exist base test strategies to be used in applying End-Game VV&T to a software system. Test strategies used in End-Game VV&T need to be refined to target the specific software system in question. Determining and executing methods to achieve the goal of each refined test strategy presents challenging issues. Documentation of the methods used in the refined test strategies can help guide future application of the base test strategies to additional software systems.

Issues relating to the selected software system: There are several issues with the selected software system. Proficiency in the language used to program the software system is necessary to maximize the results of Partial and End-Game VV&T. Knowledge of system functionality and interaction is also important to Partial and End-Game VV&T review. Without being familiar with functionality and interaction within a software system, Partial and End-Game VV&T cannot be properly applied.

System executable: The system executable needs to be recreated from the source code of the selected software system. Functionality of the recreated system executable needs to match the functionality that is described within the systems development artifacts. Database construction and secure network connections are used by the system and need to be created. The database used by the system needs to be loaded with system data that is provided alongside the development artifacts.

1.4 Solution Approach

A solution approach details steps that are taken en-route to finding a solution to a problem statement. Steps taken to solve the problem statement in this research begin with understanding all components involved in applying Partial and End-Game VV&T to a software system. The approach then moves to setting up and applying the Partial and End-Game VV&T methods to a software system. Finally, the results of Partial and End-Game VV&T are analyzed. The solution approach taken in this research is detailed as follows:

1. In-depth review and understanding of the constraints/assumptions used by Partial and End-Game VV&T.

   The constraints/assumptions in their current form are general in their description. Generality exists to allow the constraints/assumptions to be applied to many software systems. Therefore, the generality of the constraints/assumptions needs to be interpreted and translated to apply to a specific software system. Understanding the intent and purpose of the constraints/assumptions will benefit the interpretation and translation process when applying the constraints/assumptions to a specific software system.

2. In-depth review and understanding of Partial and End-Game VV&T.

   Partial and End-Game VV&T have each been introduced in previous work [2] and have no documented applicable process with which to follow. Neither Partial nor End-Game VV&T has been evaluated for their use as a software security assessment tool. This research strives to find a repeatable applicable process for Partial and End-Game VV&T

as well as determine their viability as software security assessment tools. Understanding Partial and End-Game VV&T is essential in finding an applicable process for these two software security assessment methods that follows the descriptions of Partial and End-Game VV&T in the previous work. Understanding the intent and purpose of Partial and End-Game VV&T will aide in the evaluation of the results of these two methods.

3. Select an appropriate software system with which to apply Partial and End-Game VV&T.

   Selecting an appropriate software system with which to test Partial and End-Game VV&T is not trivial. Partial and End-Game VV&T, along with the constraints/assumptions, are characterized in a manner to be applicable to a number of different software system configurations. However, a software system that embodies aspects most commonly found in a majority of software systems and that thoroughly tests the broad range of the constraints/assumptions is most desirable. For example, the majority of software systems in use today are internet based systems. Therefore, it is important to select a software system that utilizes a network interface.

4. Selecting and justifying software design artifacts from the selected system to be used during Partial VV&T.

   Accompanying a software system is a set of design artifacts that are produced during the design and development of the software system. These artifacts can vary in number, content and purpose. Previous work denotes artifacts that can be expected of a software system and which can be used in the Partial VV&T process. Different software development methods produce different sets of design artifacts with different nomenclature associated with each. It is imperative to find commonality between accepted software development methods and expected artifacts in order to appropriately asses the software system. Using this knowledge, artifacts from the selected system are either used during the Partial VV&T method or omitted based on their content and relevance to system design.

5. Apply Partial VV&T to the selected software system.

   Applying Partial VV&T to the selected software system involves finding a method with which to apply the Partial VV&T process, and obtaining feasible results while maintaining the intent and purpose of the Partial VV&T process. Finding a feasible method with which to apply Partial VV&T and assessing the results of this application are some of the major contributions of this work. Results obtained from the application of Partial VV&T to the selected software system can be used to determine if Partial VV&T is a sufficient software security assessment tool.

6. Refine test strategies to be used in applying End-Game VV&T.

   Refining test strategies derived from the constraints/assumptions is important to the End-Game VV&T process. These test strategies have been outlined in previous work [1] and can be further refined as to target the chosen software system. These test strategies target various aspects of the system in order to test each of the violable constraints/assumptions. Testing for the constraints/assumptions provides insight to any existing or potential security flaws in the software system.

7. Apply End-Game VV&T to the selected software system.

   After the test strategies are further refined to target the selected software system, End-Game VV&T can be applied. End-Game VV&T, unlike Partial VV&T, does not require a method of application. Instead, End-Game VV&T is applied by using test strategies that are derived to target specific aspects of the system and test for security flaws. Results from End-Game VV&T are guided by the violable constraints/assumptions.

8. Analyze the results and determine the validity of Partial and End-Game VV&T as software security assessment tools.

   The results of each of the VV&T methods being tested must be analyzed in order to determine the feasibility of each as a software security assessment tool. The two methods can then be compared against each other to determine which VV&T method, if either, produces better results and would be a more desirable software security assessment tool. It is also important to note the differences in the results of the two VV&T methods. After review of the two methods, the feasibility of each as a software security assessment tool can be assessed.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides an overview of background information needed to complete this work. Chapter 3 describes previous work done in this area of research and describes the software system being used in this research. Chapter 4 details the application of Partial VV&T to the software system. Chapter 5 describes the application of End-Game VV&T to the software system. Chapter 6 concludes the findings and contributions of this work and introduces future work in this area of research.

2. Background Information

This chapter provides background information for testing the viability of Partial and End-Game VV&T as security assessment methods. Background information included in this chapter helps substantiate relevancy of this research as well as provides sufficient knowledge for the completion of this research. Section 2.1 discusses the area of software security, what it encompasses, and why it is needed. Section 2.2 addresses the current software security assessment tools and their effectiveness. Section 2.3 describes various software development lifecycles that are used in creating software systems. Section 2.4 provides an overview of external components that are used by the software system under review.

2.1 Software Security

Software security can be defined as, “the idea of engineering software so that it continues to function correctly under malicious attack.”[10] Software security differs from the idea of application security in that application security aims at protecting software after the software has already been built [10]. This research focuses on testing software security assessment methods to determine their feasibility as viable software security assessment tools. Understanding the purpose of software security is important in applying a software security assessment tool to a software system.

2.1.1 Software Vulnerabilities

A main goal of software security is to prevent the presence of software vulnerabilities within a software system. A software vulnerability can be defined as, “any aspect of a computer system that allows for breaches in its security policy.”[11] Alternatively, “a vulnerability is a set of conditions that allows violation of an explicit or implicit security policy."[12] It is well recognized that software vulnerabilities are at the root of a majority of security incidents, before and after the introduction of the Internet, with the former pertaining mainly to operating systems [11].

Software vulnerabilities are prominent and broad in spectrum, which has prompted the development of taxonomies to classify software vulnerabilities and the threats they pose. Understanding software vulnerabilities is important in understanding the threats they represent [12]. Two early attempts to classify software vulnerabilities are the RISOS study [13] and the Program Analysis (PA) study [14]. The RISOS study focuses on classifying security flaws in operating systems and the PA study focuses on operating systems and programs [12, 15]. These studies are very similar to each other in that the classes of security flaws could be mapped to one another [12, 15]. Other studies that classify vulnerabilities in the UNIX operating system and network [16] and classify computer program flaws [17], are based on the findings of the RISOS and PA studies [12, 15]. Understanding classification schemes of software vulnerabilities is important to this research because the violable constraints/assumptions used in the application of VV&T are classified as they pertain to the Process/Object Model of Computation [18].

In recent years, the number of software vulnerabilities discovered in software applications is far greater than the number of software vulnerabilities found within operating systems [19]. As world-wide Internet usage continues to increase, so does the number of software applications existing as web-applications. Vulnerabilities within a web-application are more susceptible to attack because of their ability to be accessed by other computer systems over the Internet. Also, automated tools targeting web-application vulnerabilities make it easy to discover and infect thousands of websites [19]. A list of up to date vulnerability notes, which describes vulnerabilities along with their associated applications and severity level, can be found online at www.kb.cert.org/vuls.

2.1.2 Software Attacks

In order for attackers to take advantage of software vulnerabilities, attackers must exploit the vulnerabilities. A software exploit can be defined as, “a piece of software or a technique that takes advantage of a security vulnerability to violate an explicit or implicit security policy [12].” In the above definition, the term “security vulnerability” can be interchanged with the term software vulnerability, which is being used in this paper.

Exploitation of software systems can occur either locally or remotely via a network. Software exploits can be used to perform malicious attacks on a software system. Some of the more common attacks on software systems are denial-of-service, man in the middle, and spoofing related attacks [20]. A denial-of-service attack prevents legitimate users of a system from accessing system resources. For example, a denial-of-service attack can prevent a user from accessing a web page or their email. Man in the middle attacks are performed by placing a malicious user in-between two legitimate users. The malicious user then authenticates themselves to the legitimate users and network traffic is funneled through the malicious user, allowing them to change information at will. Spoofing related attacks occur when an attacker makes information provided to a legitimate user appear to be something it is not. Items such as email and IP addresses can be spoofed by an attacker.

Prevention of these attacks is becoming ever more difficult due an increase in attack sophistication with a decrease of knowledge needed to carry out these attacks [21]. Figure 1 shows the knowledge required by an intruder decreasing over time as the sophistication of the attack has increased. Another aspect leading to an increase in software attacks is the increasing complexity of software systems. Increasing the complexity of a software system adds more code and components. This inevitably leads to an increase in software vulnerabilities and thus gives an attacker more ways to exploit a software system.

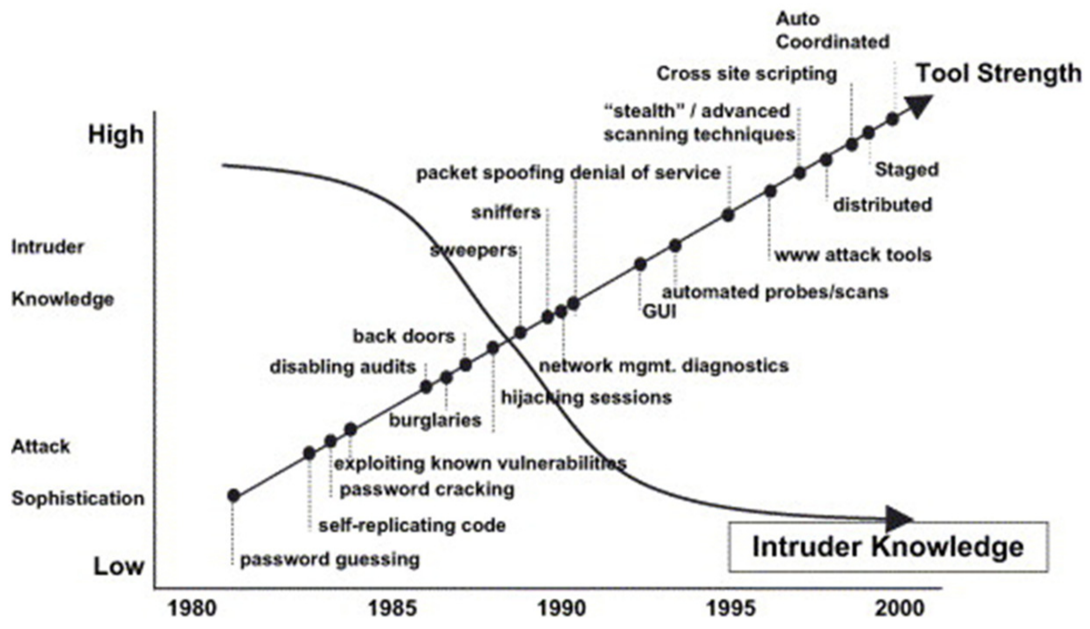


Figure 1. Attacker sophistication vs. intruder knowledge [21].

There are ways to mitigate the effects of these attacks. Software and hardware boundaries such as firewalls and Intrusion Detection Systems (IDSs) are a popular way of mitigating unwanted software intrusion. Firewalls can be placed between a user's incoming and outgoing network traffic in order to filter and deny access to unauthorized communications. IDSs are similar to firewalls except they do not filter network traffic. Instead, IDSs monitor a network for malicious activity based on known attack signatures and issues an alert or an alarm when malicious activity is detected.

User accounts and cryptographic protocols can be used to prevent attacks on private information. User names and passwords can be used to restrict access to sensitive data held in bank accounts, social networking sites, and other user access based systems. Cryptographic protocols can be used to keep private information from being read by attackers. Cryptographic protocols use keys and hashes, among others, to encrypt data and ensure confidentiality and/or integrity. However, the best way to prevent malicious software attacks from occurring is to seek and eliminate the software vulnerabilities that these attacks exploit. This is best done before a software system is deployed, but that is not always the case and gives an attacker ample opportunity to exploit the vulnerability before it is ever found.

2.2 Software Security Assessment Tools

Software security assessment tools aim to identify software vulnerabilities within a software system so that they may be corrected before the software can to be exploited and attacked.   Many security assessment techniques already exist and are readily used. Most security assessment techniques are code based analyzers that follow certain rules and principals in order to identify security vulnerabilities within the source code of a system. Due to the fact that all programmers have different ways of approaching problems, it is easy to see that a code analyzer would not be able to catch all vulnerabilities present within a system.

The purpose of using code based analyzer tools is to identify common coding problems before a system is released [22]. These common coding problems can lead to security vulnerabilities such as buffer overflow. The first code analyzing tool, called ITS4, was released in 2000 by Reliable Software Technologies, now Citigal [22]. ITS4 is based on syntactical rules that identify locations where possible security flaws could exist. Another example of a code analyzer that can be used to help with security assessment is Hammurapi. This tool looks at larger applications and is able to produce a software system specific report on its findings [23]. The report generated by the tool addresses violations of programming practices and if followed, can reduce the overall security risk of the software system being analyzed. However, these tools do not fix security flaws that are found. Instead, it is left to the developers to determine the severity of the risk, if any, and whether the security flaw should be fixed or not.

Security assessment can also take place throughout the development lifecycle of the software. One way of doing this is to incorporate a vulnerability matrix, property-based testing, and model-based security specification and verification to identify vulnerabilities as they emerge [24]. In this method, the vulnerability matrix contains a taxonomy and classification of software vulnerabilities and exploits. Properties in the property-based testing of this method are used to determine whether the current state of the execution of the system is in violation. If so, then there is a security flaw in the system. Model-based security specification and verification use models of the system, updated as the system develops under the design and requirements phases of the software development lifecycle, to more broadly and locally test the system for security properties.

Another method of focusing on security throughout the software development lifecycle is Microsoft's Trustworthy Computing Security Development Lifecycle (SDL) [25]. The SDL is used to develop more secure software systems, at the time of their deployment, by introducing security oriented activities and threat models throughout the software development lifecycle of the system. Implementing the SDL within the Microsoft organization has led to drastic reductions in software security flaws and software vulnerabilities upon the systems deployment [25]. The SDL approach of incorporating a security mindset along with security assessment measures throughout the design phases of a software system shows the importance and significance of implementing software security within the software development lifecycle.

The software security assessment approach being explored in this research varies from these two approaches in several ways. The application of End-Game VV&T is much like that of code-based analyzer tools in that End-Game VV&T is applied to the source code of the system. However, End-Game VV&T is not automated, as the code-based analyzers are. Instead, End-Game VV&T uses the violable constraints/assumptions in order to guide penetration testing of the system in checking for software vulnerabilities. Partial VV&T is also performed after the system has been developed but, Partial VV&T incorporates design artifacts produced during the software development lifecycle of the system in pinpointing software vulnerabilities. This tool is different from others in that it looks at the development and evolution of the system in finding software vulnerabilities whereas other tools simply look at the finished product, i.e. the source code.

2.3 Software Development Lifecycle

The software development lifecycle (SDLC) is a defined method of designing and developing adequate software systems. The SDLC has several phases of design, implementation, and testing that are recommended for a software system to go through during its development. Several models of the SDLC can be used. These models include the waterfall model, the incremental model and the spiral model [26]. Descriptions of the models are taken from Pressman's "*Software Engineering: A Practitioner's Approach*" [26].

2.3.1 Waterfall Model

The waterfall model, also called the classic model, is the most basic form of the SDLC models. This model follows a sequential approach to the software development process. Conventionally there are five phases in the waterfall model; requirements, design, implementation, verification, and maintenance. The phases within this model can be seen in Figure 2.

The requirements phase of the waterfall model involves conferring with the customer about system functionality, behavior and performance. These requirements are then translated during the design phase into system architecture and detail that provide baseline structure during the coding process. The implementation phase involves translating the design of the system into machine readable code, creating an executable of the system. Verification of the coded system is the next phase in the waterfall model. In this phase the coded system produced in the implementation phase is checked for desired functionality, behavior and performance. Finally, the maintenance phase incorporates all aspects the system needs to keep it operational and up to date after its deployment.

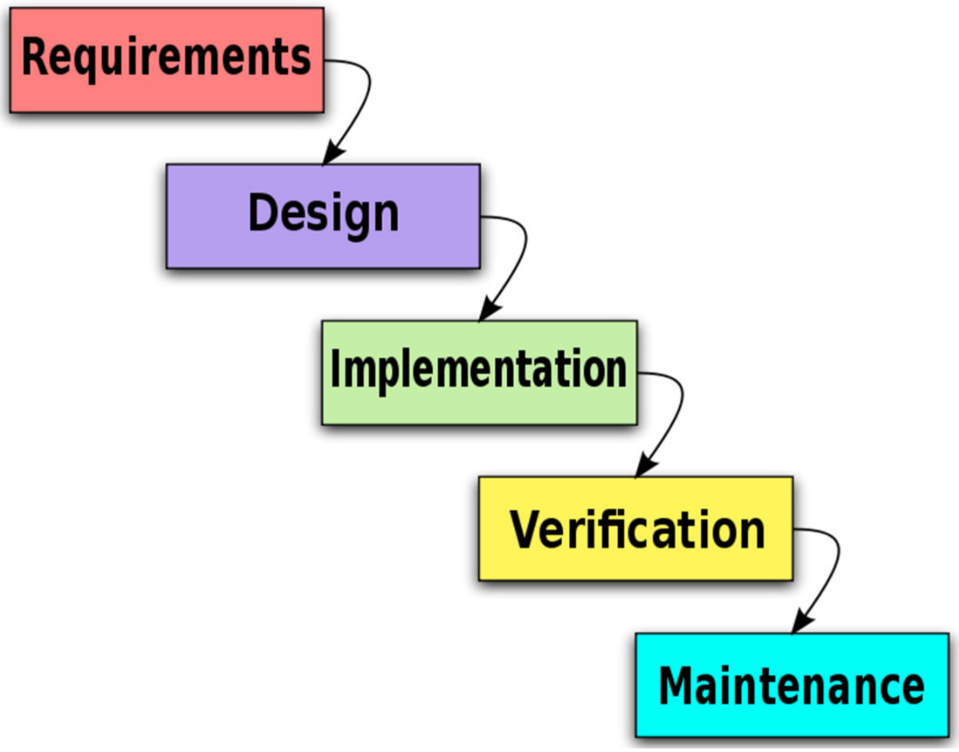


Figure 2. Phases of the waterfall model [27].

2.3.2 Incremental Model

A variation on the SDLC waterfall model is the incremental model. The incremental model uses similar phases to those of the waterfall model. The incremental model is an iterative process that focuses on delivering an operational product with each increment. A key difference, between the waterfall model and the incremental model, is the lack of a maintenance phase within the incremental model. The maintenance phase is only necessary after the final iteration of the incremental model. All phases of the incremental model follow the same purpose as those of the waterfall model. An example of the incremental model can be seen in Figure 3.

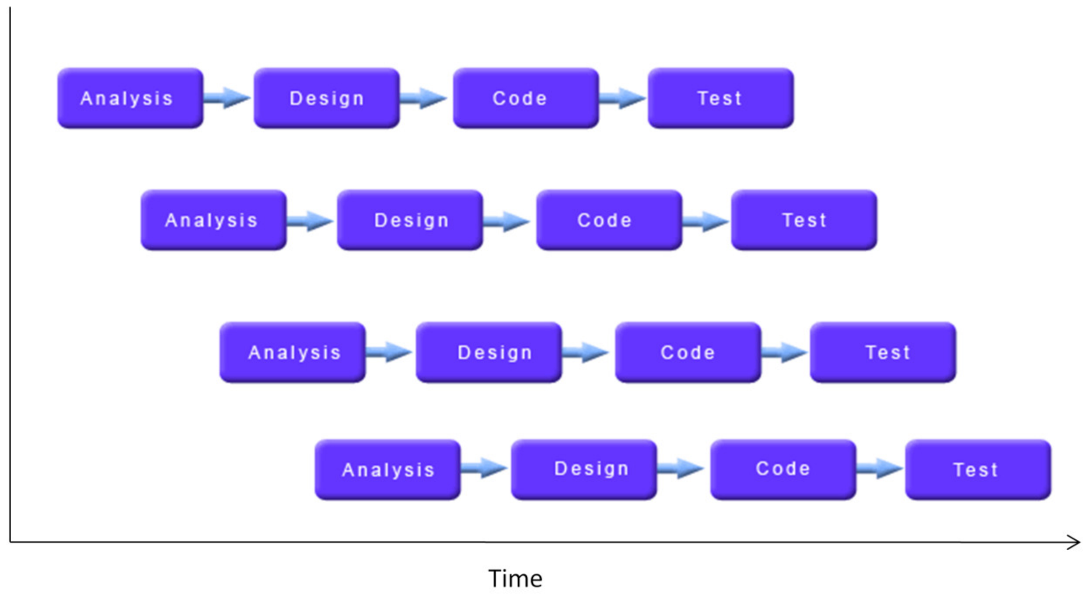


Figure 3. Incremental model phases over time. Adapted from [28].

2.3.3 Spiral Model

The spiral model is an iterative and evolutionary based design model. Unlike the incremental model, the spiral model repeats the lifecycle process after the previous iteration of the lifecycle has been completed. Also, the spiral model does not incorporate the typical five phase approach found within the waterfall and incremental methods. Instead, the spiral model is composed of task regions which can vary in number and purpose. The spiral model was first proposed in 1988 by Boehm [29]. A major difference between his proposed method and those of the waterfall and incremental models is the risk analysis stage. The risk analysis stage of the proposed spiral model, Figure 4, allows for risk assessment of the software system after each stage of development.

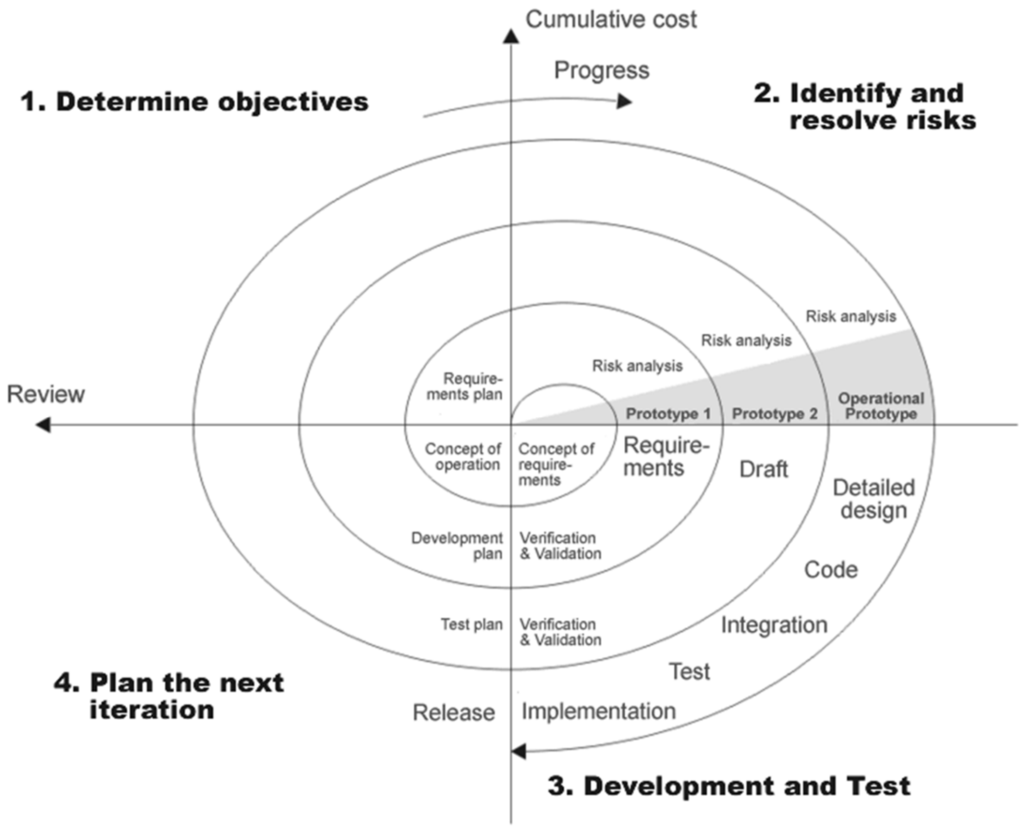


Figure 4. Spiral model showing the development lifecycle through separate task regions [29].

2.4 Elements Specific to Software System under Review

To properly assess the security of the software system we have chosen to examine, several aspects related to the system must be discussed. This section includes information relevant to the systems external components. External components include payment methods, data storage, and the language in which the system was programmed. Functionality of the external systems and protocols are addressed by several of the constraints/assumptions and can cause the software system under review to either comply or violate constraints/assumptions. An external component that violates a constraint or assumption can lead to a vulnerability in the component and compromise the security of a system as a whole. The following are external components examined in this research.

2.4.1 HTTPS

The selected software system utilizes the Hypertext Transfer Protocol Secure (HTTPS) protocol to securely send and receive data. A private portion of the software system has a user login that is secured using the HTTPS protocol. Also, the Paypal external component of the software system uses the HTTPS protocol when conducting transactions. HTTPS is a protocol that was developed in 1994 by Netscape Communications and is used for tasks such as payment transactions [30]. HTTPS involves using the original Hypertext Transfer Protocol (HTTP) over a Transport Layer Security (TLS) connection. HTTP is an application-level protocol that is widely used in transferring information across the World-Wide-Web [31]. The application layer is the highest level on the Open System Interconnection Reference Model (OSI Model). Both HTTP and HTTPS are used when accessing a website by the website's Uniform Resource Locator (URL) and can be uniquely identified as HTTP URLs begin with http:// and HTTPS URLs begin with https://. To help further differentiate HTTP from HTTPS when issuing and receiving requests, the default ports through which data flows are separate. The default port for HTTP is 80 [31] whereas the default port for HTTPS is 443 [32].

HTTP has been in use with the world-wide-web (WWW) global initiative since 1990 [31]. The current standard for HTTP is HTTP/1.1. An HTTP session is comprised of a sequence of request and response messages. In a request message, a client can request from the server a variety of resources. The server then sends a response message containing the resources requested, or an error message if appropriate. The request and response messages, along with any appropriate status or error messages, follow the standards set forth in Request For Comments (RFC) 2616 [31].

2.4.1.1 TLS

The security advantage of HTTPS over ordinary HTTP alone is that the HTTP requests and responses are sent over a secure connection. The current HTTPS standard focuses on using HTTP over a TLS connection as opposed to the original standard of using HTTP over a Secure Socket Layer (SSL) connection [32]. As the TLS name implies, TLS operates at the transport layer of the OSI Model, unlike HTTP which operates in the application layer. TLS is built upon the SSL protocol, which also operates at the transport layer in the TCP/IP model [33]. Each of these protocols, TLS and SSL, help better secure data traversing across the internet. The standard defining the TLS protocol is documented in RFC 2246 [33] and will be used in describing the functionality of TLS.

TLS is a client/server model that uses a handshake when establishing a connection. In the first part of the handshake, the client sends the server a "hello message" that includes the TLS protocol version, a random number generated by the client, and a list of cipher suites and compression methods that the client can use. The server responds to the client "hello message" by sending its own sever "hello message". The server "hello message" contains the chosen TLS protocol version to be used during the connection, a random number chosen by the server, and a cipher suite and compression method chosen from the list sent in the client "hello message". The server has the option during the server "hello message" to send a session id if performing a resumed handshake. The server follows the server "hello message" by sending its "certificate message" and then sending a server "hello done message".

After the server "hello done message", the client sends a client "key exchange message" which can contain a pre-master secret, a public key, or nothing at all. This message prompts both the client and server to compute a master secret that is derived from the random numbers exchanged earlier and the pre-master secret, if there is one. The client and server then exchange "change cipher spec messages" as well as "finished messages". The "change cipher spec message" indicates that any data sent after the message will be authenticated and encrypted (if indicated). The "finished messages" sent by the client and server are authenticated and encrypted messages that contain a hash and a Message Authentication Code (MAC) over the previous handshake messages. If the receiving entity cannot decrypt and verify the sent "finished message" then the connection is deemed to have failed and the connection is torn down and rebuilt. Figure 5 shows an interaction diagram for a client and a server implementing a TLS connection.

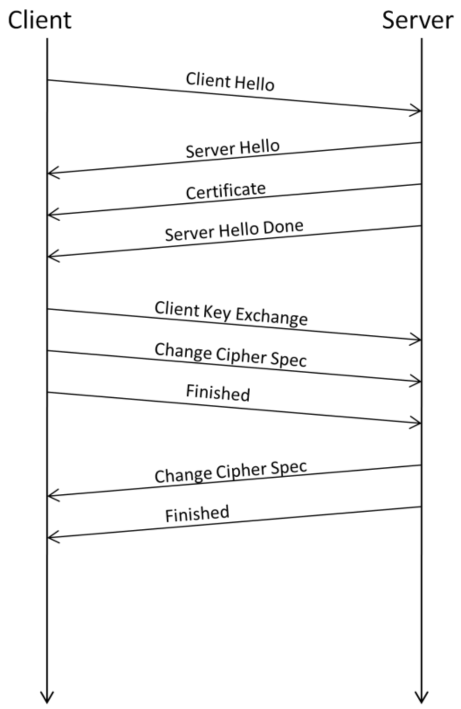


Figure 5. Interaction diagram of a TLS handshake.

For the server to be authenticated, the digital signature of the certificate issued by the server, in the server "certificate message", must be checked with the server certificate's issuing Certificate Authority (CA) chain. A CA chain is a chain of CAs that certifies each other until a trace back to a known and trusted CA is found. In addition to authenticating the server, the identity of the server must also be checked. This can be done by checking the URL with that of the URL embedded in the server's certificate. Comparing the URLs is a valid way of certifying the server's identity because only a trusted CA can embed a URL in the certificate. Once the TLS connection has been established between the client and the server, the HTTP request and response messages are able to start being sent.

2.4.1.2 Message Authentication Code (MAC)

For many operations in the TLS standard, a keyed MAC is required [33]. A keyed MAC means that the MAC is computed using a secret "key" that cannot be forged by entities not involved in the TLS connection. TLS, more specifically, uses a keyed-Hash Message Authentication Code, or HMAC. For the TLS handshake, the hash algorithms MD5 and SHA-1 are hardcoded into the handshaking protocol for use with generating the HMACs [33]. The description of the HMAC implementation is taken from RFC 2104 [34].

The output of a HMAC is essentially a hash of a hash. Hence, there is an inner hash function and an outer hash function. Associated with the inner and outer hash functions is an inner and outer pad that is used in conjunction with the secret key. The value of the inner pad (ipad) is the byte 0x36 and the value of the outer pad (opad) is the byte 0x5C. Each of these, ipad and opad, are B bytes in length where B is the length of the block size being used in the hash function. To get the B length for the ipad and opad, the byte value of each is simply repeated B times. An exclusive or (XOR) operation is performed between either the ipad or opad and the secret key before the execution of their respective hash functions. Below is a definition for generating the HMAC taken from [35].

$$\text{HMAC}(K,M) = H\,[\,(\,K^{+}\text{ XOR opad}\,)\,||\,H\,[\,(\,K^{+}\text{ XOR ipad}\,)\,||\,M\,]\,]$$

In this definition, K represents the value of the secret key to be used in the keyed hash generation. Furthermore, K^{+} represents the value of the secret key padded with zeros such that the length of K^{+} will be B bytes long. H is the embedded hash function used in the HMAC generation. Opad and ipad represent the values that were described above and along with K^{+} are both B bytes in length. M represents the input message to the HMAC (including any padding required by the embedded hash function). Lastly, the symbol || represents a concatenation operation, meaning that the value to the right of the operator is appended to the value to the left of the operator. A detailed view of the HMAC structure can be seen in Figure 6.

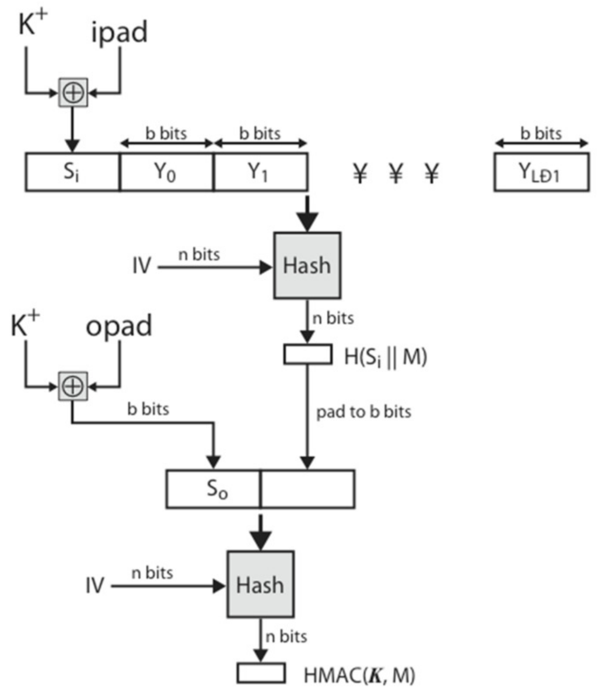


Figure 6. Description of a general HMAC structure [35]

2.4.2 Java

The selected software system is programmed using the Java programming language. Java has some inherent security properties. Understanding the Java programming language is essential to performing the code review of the selected software system. The Java programming language was developed by James Gosling and introduced by Sun Microsystems in 1995 [36, 37]. The Java syntax is mostly derived from the C and C++ programming languages with some of the aspects from the C and C++ languages being omitted and ideas from other programming languages being incorporated [37]. Likening the Java programming language to that of the C and C++ programming languages made for a gentler learning curve for new users of Java. Java is an object-oriented programming language, which is prevalent in the fact that every Java program has at least one class [36]. A class, in terms of programming, "is a definition template for structuring and creating objects [38]." Also, Java is a strongly typed language, helping it clearly distinguish between run-time and compile-time errors [37].

Java is a high-level programming language in that machine representation, or machine level code, is not available through the language itself [37]. Abstracting machine level code allows programmers to easily code functionality without having an extensive knowledge of machine representations. Extending on the borrowed C and C++ programming languages, several features to help enhance the security of Java programming were also added. Automatic storage management and garbage collection have been implemented in the Java programming language [37]. These two features help to prevent against excessive memory allocation and memory leaks. Memory leaks can occur in the C and C++ programming languages when memory is allocated by a program, but is not freed, or released, by the program. Another security measure implemented in the Java programming language, which is not found in the C and C++ programming languages, is that Java prevents array access without index checking [37]. To further ease memory management, there are no pointers in Java [37]. Pointers can be a major cause of security flaws in C and C++ as a pointer can be redirected to point to malicious code and the referencing of a null pointer will cause a program to crash.

In addition to added security features, Java has another feature that is separate from that of most other programming languages. The Java programming language is platform independent [39]. Platform independency is a very attractive feature of the Java programming language because it means that programs will only have to be written and compiled once and still run on multiple computer architectures. This can alternatively be termed a write on one, run on many (WORM) programming language [36]. To achieve platform independency, Java programs are compiled into bytecodes [39]. A bytecode can be defined as "an architecture neutral intermediate format designed to transport code efficiently to multiple hardware and software platforms [39]."

This bytecode is then run on the Java Virtual Machine. The Java Virtual Machine is responsible for the platform independency experienced by the Java programming language [40]. Even though the Java Virtual Machine is an abstracted computing machine, it includes an instruction set and manipulates memory during run-time, just as an actual computing machine does [40]. To run a Java program, the appropriate Java Virtual Machine for the architecture on which the program will be run must be installed. Once the Java Virtual Machine has been installed, the bytecode of the compiled Java program will be able to run on the system. The Java bytecode of a program is able to run on any system architecture that supports the Java Virtual Machine.

Java is the first programming language to have built-in support for networking applications [36]. This, combined with the platform independency of the Java programming language, has made Java an attractive option when dealing with networked applications. Java has become a proven and ideal language for developing "secure, distributed, network-based end-user applications [39]." Security features mentioned before, such as the absence of pointers in the Java, further secure the Java systems in networked environments whereas C or C++ systems would be more vulnerable.

2.4.3 MySQL

To store information, the selected software system uses a MySQL database. The MySQL database must be reviewed for possibly security vulnerabilities along with the rest of the software system. The acronym MySQL stands for My Structured Query Language [41]. MySQL has gained momentum as a free and easy database and is used by many websites. MySQL is used by popular websites such as Facebook and Google [42]. A database is a structured collection of data that can hold vast amounts of information [41]. Databases are controlled by database management systems. A DBMS allows data to be added, accessed, and processed in a computer database [41]. MySQL is a relational database management system (RDBMS), meaning that data is stored in separate tables rather than a traditional database management system (DBMS) [41]. The RDBMS term was first introduced by E. F. Codd [43].

A popularity selling point of the MySQL database management system is that it is open source. Open source means that anyone can download, use, and modify the software for free [41]. MySQL is licensed under the GPL (GNU General Public License) [44], which specifies the limitations of the open source content. While there are limitations under the GPL, open source software allows for a flexibility that is not found in other systems.

In addition to being open source, MySQL also has some security measures that are not always found in other database management systems. MySQL uses unencrypted connections by default. However, yaSSL is built into MySQL and can be enabled for secure connections [45]. Also, SSH can be used to encrypt the connection between the client and the MySQL server using port forwarding and tunneling [45]. Authentication is required to use the MySQL server and encompasses three scopes: Client Name/IP Address, Username, and Password [45]. While the MySQL server does require authentication, there is a security flaw with the root user of MySQL. The password for the MySQL root user is blank by default [45]. This leaves a database susceptible to malicious attacks and the root password should be set, to sufficient password strength, before a database is brought up and running.

Every user that has access to the MySQL database is stored in the user table of the MySQL database. As specified in the scope of user authentication, each user has a password which is also stored in the MySQL table of users. As a security measure, these passwords are not stored in plaintext. Instead, a hash value computed from the password is stored [45]. This prevents possible attackers from being able to see a user's plaintext password in the MySQL user table.

MySQL users are privileged. Each user can be granted or revoked of privileges that allow them to interact with database objects in certain ways [45]. User privilege information is stored in the grant table of the MySQL database [45]. Access control can also stem to user restrictions. These restrictions can include requiring users to connect to MySQL using only SSL connections and limiting the connections and/or queries per hour a user is allowed to issue [45]. A MySQL query is a way of both adding and accessing data to and from the database. Limiting connections and queries can help to prevent against denial of service attacks.

Data Encryption is also supported by MySQL. Encryption functions that are supported by MySQL are AES, Triple-DES, MD5, and SHA-1 [45]. These functions can be used to encrypt data that should not be stored in plaintext. A prime example of this would be if an application were to use a table for storing its own user names and passwords, then the application can encrypt the passwords before storing them into a table [45]. Future MySQL security features will include password and account management, vulnerability assessment, and auditing and logging of MySQL database activities [45]. Security features that are currently present in MySQL can also help further secure the applications that use MySQL for their database store.

2.4.4 Apache Tomcat

The selected software system is a dynamic online web application. Apache Tomcat is used to facilitate network access for the software system. Tomcat is discussed because of its relation with network interface constraints/assumptions. Apache Tomcat, Tomcat for short, is a Java 2 Platform Enterprise Edition (J2EE) web container and web server developed by Apache Software Foundation (ASF) [46]. A web container, with respect to J2EE, implements the web component of the J2EE architecture [47]. Implementing the web component of the J2EE architecture involves specifying a runtime environment for web components that includes security, deployment and other services [47]. A web container is provided by a J2EE server [47]. Since Tomcat implements both a web server and a web container, it is an attractive option for developing and deploying Java based web applications.

J2EE is a Java platform that focuses on developing and deploying business applications that have high availability and are easily integrated with other applications [46]. This platform carries with it both Java servlet and JavaServer Pages (JSPs) technologies [48]. There are two distinct components inside of Tomcat that support these two technologies: Catalina and Jasper. Catalina is a servlet container that provides support for servlet applications [49]. Jasper is a container that provides support for JSPs [49].

2.4.4.1 Java servlet

A Java servlet can be thought of as a Java applet that runs on the server side, as opposed to the client side [50]. Servlets are commonly used with web servers in order to take the place of Common Gateway Interface (CGI) scripts [51]. Servlets are a popular choice for building interactive web applications as they can offer similar functionality to CGI scripts without the performance limitations [52]. Advantages of servlets over other methods that enable dynamic web content include “portability, power, efficiency, endurance, safety, elegance, integration, extensibility, and flexibility [51].”

As with the Java programming language, Java servlets are server and platform independent[51]. Independency of servlets allows for easy transfer and scalability of a system. Java servlets maintain their portability by running on a server side Java Virtual Machine (JVM) in order to carry out functionality [51]. Additionally, running Java servlets in the domain of the server releases the requirement for web browsers to support Java, which is the case for Java applets [51]. Furthermore, Java servlets experience the portability and security that is afforded with the Java programming language.

Standalone servlet containers have built-in support for servlets and require no further installation or plug-ins [51]. Tomcat falls under the category of standalone servlet containers. There are, however, disadvantages of having built-in support for servlets. A new release of the web server must be released before the latest servlets are supported and vendors typically only support the JVM that they provide [51]. The current Java servlet spec that is in use with the Tomcat web server is version 2.5 [53].

2.4.4.2 JavaServer Pages

JavaServer Pages (JSP) is a tool used for the development of dynamic web sites and web applications and was first released as JSP 1.0 in 1999 [54]. JSP pages are not like Hyper Text Markup Language (HTML) pages in that HTML pages contain static content whereas JSP pages can change their content dynamically based on a number of variables [54]. However, JSP pages are a combination of standard markup language elements, such as HTML elements, and special JSP elements that allow the use of dynamic content [54]. The difference between JSP pages and Java servlets is that Java servlets have HTML code embedded inside of the servlet program code whereas for JSP pages, HTML code is a separate statement from that of the JSP elements [54].

In order to generate the dynamic content within a web page, the special JSP elements within the page are executed, upon request of the page, and then merged with the static content of the page and sent back to the browser used to access the page [54]. Dynamic generation of web pages allows for instant user feedback, which is important in business based web applications. Another important feature in most business based web applications is the use of a database. JSP has the ability to easily connect to a database, which makes this a very attractive option for developing dynamic web applications [55]. Figure 7 shows the interaction between a JSP file, Java servlets and a Tomcat web server.

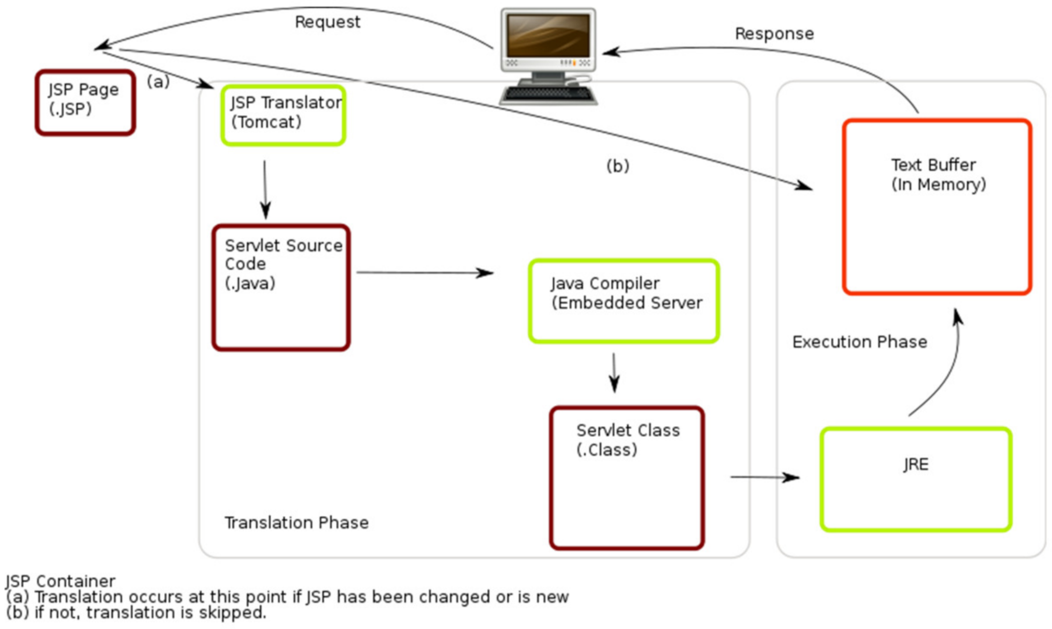


Figure 7. Interaction diagram of a JSP file with Tomcat and Java servlets [56].

First, a user requests a JSP file from the Tomcat server. The JSP elements within the JSP page are then translated by the Tomcat server and associated with servlet(s). The dynamic JSP element requests are then sent, in servlet form, to the Java Runtime Environment (JRE) where they are run on a JVM. The results of this are then combined with the static markup language content of the JSP page in the text buffer and reconstructed into a response to be sent to the user.

2.4.5 Paypal

Paypal is used by the selected software system to handle monetary transactions. Interactions between Paypal and the selected software system are important when reviewing the system against the constraints/assumptions. Paypal is an online monetary transaction service that was first launched in 1998 [57]. Since then, Paypal has become the world's fastest growing global currency exchange [57]. For most, Paypal is most closely associated with monetary transactions through the popular online auction site eBay. However, Paypal is offered to any willing users and offers pre-integrated solutions for business related web applications such as shopping carts and storefronts [57].

Paypal works fairly simply. It acts as an intermediary during a monetary exchange. To make a transaction, the monetary sender must have a Paypal account that has been credited with the agreed upon amount. This account can be credited through credit card, debit card or by any other means which allows a user to deposit funds instantly into an online account. The amount of the transaction is then transferred to the recipient of the monetary transaction. The recipient can receive this amount into a Paypal account, a bank account, or can receive a check from Paypal. Paypal is able to make revenue by charging transaction fees for each monetary transaction that it processes.

Security is a prime issue for a currency exchange website such as Paypal. To ensure confidentiality, Paypal uses the HTTPS protocol in conducting business. To further verify a user issuing a payment using Paypal, Paypal offers a security key. The security key creates random temporary security codes that a user can use to safe guard their Papal account when they login [58]. The security key is synchronized with the users account and the security code it generates is used as an extra authentication challenge at login.

3. Related Work

Chapter 3 introduces work that is related to and the inspiration for this research. This chapter starts by explaining the Process/Object Model of Computing and how it relates to this research in Section 3.1. Section 3.2 describes the set of violable constraints/assumptions and how they relate back to the Process/Object Model of Computing. Section 3.3 details the various Access-Driven Verification Validation and Testing strategies that can be used with the violable constraints/assumptions. Finally, section 3.4 describes the Horticulture Club Sales Assistant software system that is used to test Partial and End-Game VV&T.

3.1 Process/Object Model of Computing

The Process/Object Model of Computing relates software vulnerabilities to computer system resources [18]. To relate vulnerabilities to computer system resources, the Process/Object Model uses a high-level view of a software process (software application) executing on a computer system [18]. In this high-level view, the process is visualized as a black-box entity that accepts input, performs operations upon the input, and then produces output [2]. Figure 8 provides an illustration of how the Process/Object model views the execution of a software process. In this figure there are three resources associated with the execution of the software process: Memory, Input and Output (I/O), and Cryptographic resources.

Software processes access resources according to rules and limitations that are defined either by the resources themselves or by the operating system [18]. Rules and limitations provide constraints that a software process must follow to achieve proper functionality [18]. In addition to constraints, the software process makes assumptions about resources that it uses [1]. A vulnerable state can arise when a software process fails to follow a constraint put forth by a resource or makes an incorrect assumption about a resources' usage [18].

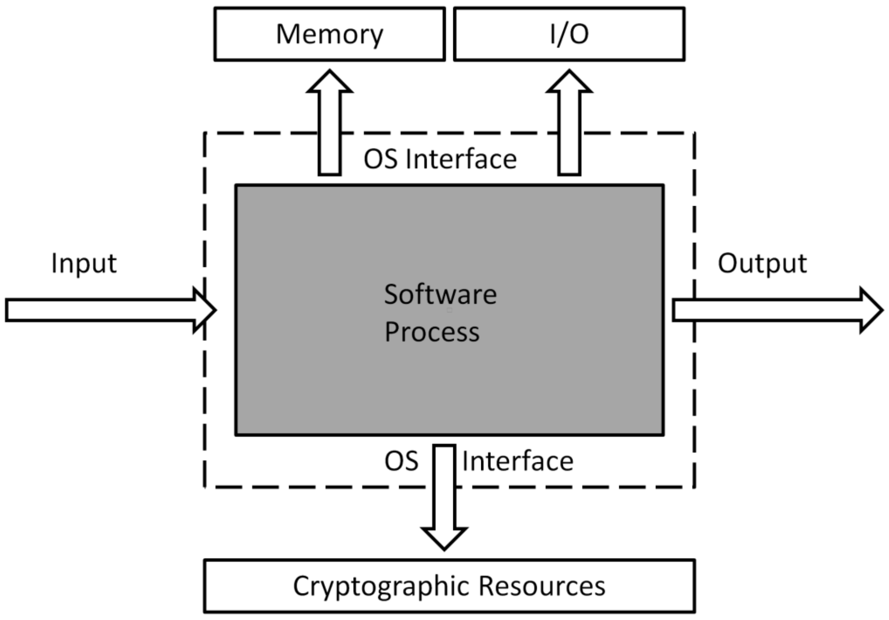


Figure 8. High level view of a software process executing on a computer system, adapted from [2].

3.1.1 Memory

The memory resource is used by the software process to store data, instructions and other execution specific parameters [1]. Anytime a software process creates a new variable, accesses a variable or deletes a variable the memory resource is used. Memory is used to temporarily store files and other data objects while the software process is executing on them. Constraints of the memory resource that can be violated include restricting the length of input to a data buffer and not allowing variables representing length or quantity to hold negative values. If the length of a data buffer is exceeded, then a buffer overflow vulnerability can occur. Allowing negative values to be stored in variables representing length or quantity can cause unpredictable behavior by the software process. Assumptions of the memory resource include not interpreting data on the memory as executable code and pointer variables must reference legal memory locations. Interpreting data on the memory as executable code can give an attacker unlimited access to a computer system. For pointer objects, referencing an illegal memory location can lead to system failure and denial-of-service.

3.1.2 I/O

The I/O resource provides the software process with functionality to receive and store input, and store and send output [1]. I/O resources are vulnerable to several types of attacks. If values accepted by the I/O resources are not properly checked, then fatal system crashes may occur resulting in denial-of-service attacks. Constraints of the I/O resource include access permissions of files used by the process and that the process will be provided with requested filesystem space. If permissions to files are not restricted to only required principals, attackers can modify files before a software process is able to use them. Fatal system crashes can occur if a software process is not provided with the filesystem space that it requests. Assumptions of the I/O resource include ensuring that data sent/received is not modified and that the network interface will be available if needed. If data is sent or received using the network interface then the data has the potential to be modified by an attacker. Unavailability of the network interface can lead to denial-of-service attacks for web based software systems.

3.1.3 Cryptographic Resources

Cryptographic resources allow for the software process to store secrets and ensure data integrity [1]. For the Process/Object Model, as well as this research, cryptographic algorithms and protocols are considered as resources [18]. Constraints of cryptographic resources include not using encryption to ensure data integrity and not using obfuscation instead of encryption to ensure confidentiality. If encryption is used to ensure data integrity, then an attacker can easily modify data being received or sent by the process in order to corrupt the data. Using obfuscation instead of encryption to ensure data confidentiality makes a system vulnerable as obfuscation of data is easily decipherable. Assumptions of cryptographic resources include data produced by the random number generator being unpredictable and the key length of cryptographic algorithms being sufficient. If data produced by a random number generator is predictable, then future numbers produced by the number generator can be guessed and cryptographic algorithms using this number can be broken. Keys of insufficient length can permit the keys to be easily guessed using brute force and compromise the cryptographic algorithms using the keys.

3.2 Constraints/Assumptions

The constraints/assumptions mentioned within the Process/Object Model can be used to form a taxonomy of vulnerabilities. This taxonomy is based on the relationship of the constraints/assumptions to the various resources described in the Process/Object Model [18]. Resources defined in the Process/Object Model are used as categories in the taxonomy and hence, the top-level categories of the taxonomy are Main Memory, Input/Output, and Cryptographic Resources [18]. Decomposing the taxonomy, each top-level category has two subcategories to better differentiate vulnerabilities. The taxonomy of vulnerabilities based on the Process/Object Model, including subcategories, can be seen in Figure 9.

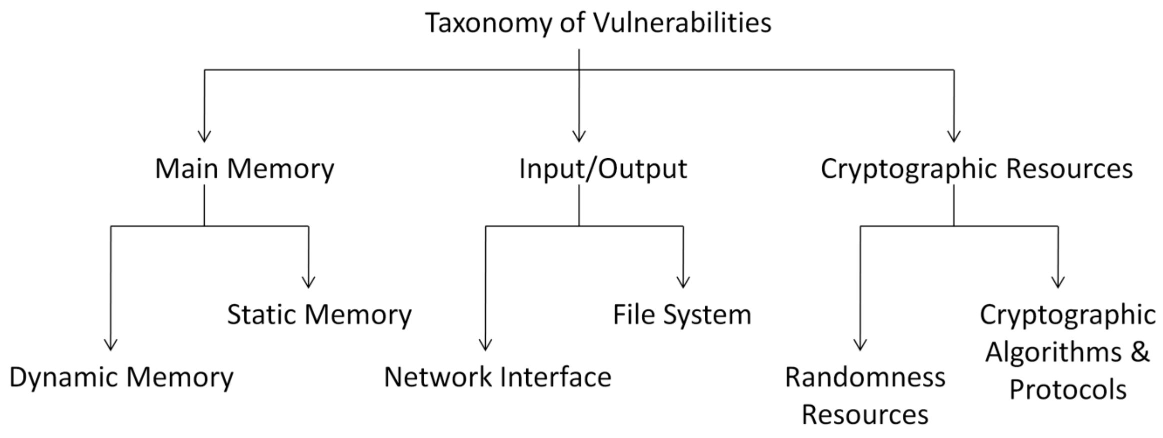


Figure 9. Taxonomy of vulnerabilities, adapted from [18].

Constraints/assumptions underlie the vulnerabilities that are classified in the taxonomy [18]. Vulnerable states can arise in a software system when the system does not enforce a constraint being assumed by a resource or makes incorrect assumptions about the resource [18]. A full listing of the violable constraints/assumptions can be found in [1]. Each of the categories from the Process/Object Model in the taxonomy has two subcategories with each subcategory having associated process objects that are exploitable. Description of the taxonomy is taken from [2].

3.2.1 Memory

The memory category of the taxonomy of vulnerabilities pertains to data stored by the software process, instruction for the software process, as well as other execution specific parameters. There are two subcategories to the memory category that can be considered as resources: dynamic memory and static memory.

3.2.1.1 Dynamic Memory

Dynamic memory is memory that can change size throughout the execution of a software process. Dynamic memory can be allocated any time during execution by the software process and likewise, can be freed by the software process. Process objects that incorporate dynamic memory include stack variables, environment variables, indexing integers, and heap-allocated variables. Dynamic memory vulnerabilities give an attacker the ability to manipulate data and instructions used by a software process. Unchecked variables used by the software process and stored on the dynamic memory introduce several vulnerabilities. Possible vulnerabilities include out of bound memory access, unexpected system behavior, and fatal system crashes.

3.2.1.2 Static Memory

Static memory is another memory resource that is a subcategory in the taxonomy of vulnerabilities. Static memory is different from dynamic memory in that the size of static memory cannot change during execution of the software process. The size of static memory is instead fixed before the software process begins execution. Process objects that incorporate static memory include global variables, and enumerated type definitions. Buffer overflow is an example of a vulnerability in the static memory resource. Buffer overflow occurs when data of greater length than a buffer is written to the buffer. This vulnerability can allow an attacker to possibly write instructions to the program stack.

3.2.2 I/O

Input/Output (I/O), as a resource to a software process, is used to receive input and communicate output [1]. I/O is comprised of two major components, which act as subcategories in this taxonomy: the filesystem and the network interface.

3.2.2.1 Filesystem

The filesystem is viewed by the software process as a system of connected directories containing metadata about files and files themselves. Process objects that exist in the file system subcategory include directories, files and permissions. If permissions are not properly assigned to files within the filesystem, vulnerabilities may exist. The filesystem can be exploited by an attacker modifying a file that the attacker should not have permission to do so. This modified file could include executable statements or other data that further compromises the security of a software system.

3.2.2.2 Network Interface

The network interface subcategory of I/O physically corresponds to the network interface card in a computer system [1]. However, communication through the network card by the process is handled via ports. This allows the software process to treat the network interface as a resource [1]. Process objects that are provided to the software process by the network interface include network access points (ports) and data packets. A vulnerability can occur if data received by the software process is modified by anyone other than the sender and used by the software process. An attacker can exploit this vulnerability to attack a software system anonymously by changing the data being sent to the software process while keeping the sender information the same.

3.2.3 Cryptographic Resources

Cryptographic resources are not like the previous two categories of resources in that there are no physical devices associated with cryptographic resources [1]. Instead, cryptographic resources are software components themselves that are used by software processes [1]. These software components are algorithms and protocols that are used to secure data by providing confidentiality, authentication, integrity, and non-repudiation [59]. There are two subcategories of cryptographic resources: Randomness resources and cryptographic algorithms and protocols.

3.2.3.1 Randomness Resources

Randomness resources are those that generate random numbers that are uniformly distributed over a set [2]. Random numbers are generated by providing a seed number to a Pseudo-Random Number Generator (PRNG) that in turn produces an unpredictable series of numbers [2]. Exploitable objects within the randomness resources include the PRNG seed and the series of random numbers produced by the PRNG [2]. Vulnerabilities can occur when the number produced by a PRNG is predictable. If the number produced by the PRNG is predictable, an attacker can predict future numbers produced by the PRNG and potentially break and cryptographic algorithms or protocols utilizing the number.

3.2.3.2 Cryptographic Algorithms and Protocols

Cryptographic algorithms and protocols provide the software process with confidentiality, integrity, authentication, and non-repudiation [1]. Algorithms and protocols in this subcategory include encryptions algorithms, authentication protocols, cryptographic checksums and hashing algorithms, and protocols and algorithms that promote non-repudiation [1]. The process objects most targeted for exploitation within this subcategory include encryption/cipher keys and their lengths [2]. A vulnerability can occur if a software process uses obfuscation instead of encryption to ensure data confidentiality. Obfuscation can easily be broken by brute force whereas encryption using keys of sufficient length and several iterations becomes increasingly infeasible to break.

3.3 Access-Driven Verification, Validation and Testing

Verification, Validation and Testing (VV&T) methods have been developed that use information from the Process/Object Model and the taxonomy of vulnerabilities, and provides base test strategies that test for the presence of vulnerabilities within a software process [1]. Verification of a software system answers the question: *Are we building the system right?* Validation of a software system answers the question: *Did we build the right product?* Testing of a software system assess the presence of desirable characteristics within the software system. Testing is utilized throughout both verification and validation activities but can be performed independently as well [2].

Three access-driven VV&T methods are Full VV&T, Partial VV&T, and End-Game VV&T. Each of these is guided by accessibility to the development process and availability of the development artifacts [2]. Of these three strategies, Full VV&T is the most beneficial [2]. This is because the constraints/assumptions can be enforced from the beginning of the software development process. Full VV&T is followed by Partial and End-Game VV&T in order of being most beneficial to finding software vulnerabilities. Full VV&T was not performed in this research. However, the effectiveness of both Partial and End-Game VV&T to detect vulnerabilities within a software system is explored. Information used in the discussion of the access-driven VV&T methods is taken from [2].

3.3.1 Full VV&T

Full VV&T enables software developers to build in an appropriate level of security into the software process because security concerns can be raised during concept definition. Being security conscious from the beginning of the software development lifecycle allows developers to incorporate more security oriented activities into the development lifecycle of the software system. Security related requirements can be added to the requirements specification and ultimately seed into the design documents as well as the code of the system. Having security incorporated into the design and requirement documents will lead to having security within the systems test plan and description documents. Furthermore, the existence of security within these documents means that the software system will be tested for its compliance with security related requirements during the validation of the system.

The violable constraints/assumptions can be rewritten as security requirements and be placed into the requirements specification. Having the constraints/assumptions exist as requirements for the software system will allow for tests of the constraint and assumption requirements to be constructed during the design phase. There is currently ongoing work on translating the violable constraints/assumptions into working security requirements for software systems.

3.3.2 Partial VV&T

Partial VV&T is performed after the software development process is completed and assumes that all artifacts generated during development of the process are still available for examination. Artifacts that are generated during the development process include the requirements specification, the design documents (both high-level and detailed), the source code, and the executable of the software system. There are two outlined objectives of Partial VV&T. One objective is to examine the generated requirements specifications and design documents for security related elements. Another objective is to use code analysis tools to identify security vulnerabilities within the software systems code.

Partial VV&T assumes that the software system is developed with minimal knowledge of the violable constraints/assumptions. Examination of development artifacts can be guided by the Process/Object Model. Content in requirements specifications or design documents relating to Process/Object Model resources can help focus document examination on security related issues.

Analyzing the source code during Partial VV&T can be performed either manually or using an automated process. Using an automated process to analyze the source code of a system is less effort intensive than visually inspecting the code. However, there exists no current tool that employs the constraints/assumptions technology. Therefore, for this research the source code was analyzed manually in order to analyze the code for the presence or omission of constraints/assumptions as well as any other software vulnerabilities.

In addition to the source code of a software process, the process executable is available to Partial VV&T. The process executable should only be used in Partial VV&T if the source code of the process is not available for examination. Having access to the source code of the process eliminates the need for having to design runtime tests for the process executable. Analyzing source code of a process is more revealing than testing the executable because it is easier to see where vulnerabilities might exist. Manually analyzing process source code and testing the process executable can be equally effort intensive; however, source code analysis can reveal vulnerabilities without extensive test cases and testing.

3.3.3 End-Game VV&T

End-Game VV&T is like Partial VV&T in that End-Game VV&T is performed after the software development process has completed. However, End-Game VV&T is used when the only development artifact available is the process executable. Lacking the verification and validation properties of its misnomer, End-Game VV&T is performed through testing only. Testing for the presence or omission of constraints/assumptions via End-Game VV&T is performed using base strategies that are derived to test for specific constraints/assumptions.

The base strategies are composed of three elements: a target, goal, and method [1]. The target specifies the process object(s) within the software system that the base strategy is focusing on. The goal of the strategy is to determine whether or not the software process violates the constraint or assumption associated with the target. Finally, the method is the approach taken to achieve the goal of the base strategy. The base strategy can be applied to any software process operating above the level of the operating system [1]. This is because the software process was viewed as a black-box entity in the Process/Object Model which eventually led to the development of generic base strategies. Base strategies can be refined to target specific software systems and be implemented in test cases to see if specific software process is vulnerable.

3.4 Horticulture Club Sales Assistant

This section presents an overview of the Horticulture Club Sales Assistant. The Horticulture Club Sales Assistant is a web-based plant purchasing system intended for use by the Horticulture Club of Virginia Tech. Partial and End-Game VV&T are applied to the Horticulture Club Sales Assistant to determine their feasibility as software security assessment methods.

3.4.1 Selection of Software System

A software system must meet three necessary criteria to be considered for review under Partial and End-Game VV&T. First, the development of the software system must have already been completed. Second, the software system must have been developed without prior knowledge of the violable constraints/assumptions. Finally, all artifacts produced during the development of the software system would need to still be accessible. All three criteria are requirements for applying Partial VV&T to a software system. However, only the first two criteria are requirements for applying End-Game VV&T to a software system.

In addition to the necessary criteria, additional features of a software system make a system more desirable for review under Partial and End-Game VV&. Because this is the first time the constraint/assumption technology has been applied to an actual software system, a system exercising the full range of the constraints/assumptions would produce more meaningful results. To test the full range of the constraints/assumptions, the system must be networked, store and manipulate information, and involve the transfer of sensitive data.

The Horticulture Club Sales Assistant program is a software system that was developed for the Horticulture Club of Virginia Tech. This system met the requirements of this research and is very accessible. Development of the system is complete and a full set of development artifacts that are accessible. The Horticulture Club Sales Assistant was developed without previous knowledge of the violable constraints/assumptions. The system has the desirable features of being networked, having a manipulative file store, and required the transfer of sensitive data.

3.4.2 System Development

The Horticulture Club Sales Assistant program was developed for the Horticulture Club of Virginia Tech in order to automate their annual plant sale. The main feature of this system differentiating it from traditional Horticulture Club plant sales would be the ability to place preorders online. Allowing online preorders involves the development of a networked web application that is able to access and manipulate inventory, accept and process monetary transactions, and store all order information.

Development of the Horticulture Club Sales Assistant follows the traditional waterfall model of software development. This is evident from the development artifacts of the system. Artifacts of the system include a product overview document, a requirements document, design documents, integration and testing documents, source code, and the executable binary of the system. Although the development of the system follows the traditional waterfall model of software development, the model is not followed directly and there are some differences. The implementation and verification stages of the waterfall model were combined during the development of the Horticulture Club Sales Assistant program and produced a single development artifact termed the integration and testing document. For the purposes of this research, the maintenance stage of the waterfall model is ignored as there are no associated artifacts available and no indication that maintenance of the system is ongoing.

The requirements document and design documents were produced during their respective development stages in the waterfall model. Development of this system produced one artifact not having an associated development stage in the waterfall model. The product overview document was produced prior to the requirements stage of the waterfall model. Not having an associated development stage within the waterfall model does not preclude the product overview document from use in this research. The product overview document, while produced outside of the waterfall model, can still contain pertinent information when reviewed under Partial VV&T.

3.4.3 System Architecture

The architecture of the Horticulture Club Sales Assistant program complements the testing of Partial and End-Game VV&T. The system is a web application for viewing and purchasing plants from the Horticulture Club at Virginia Tech. As such, the system implements a server/client interaction with the system acting as the server. A diagram depicting the server/client interaction is shown in Figure 10. The system is coded using the Java programming language and server functionality was provided by using an apache Tomcat web server.

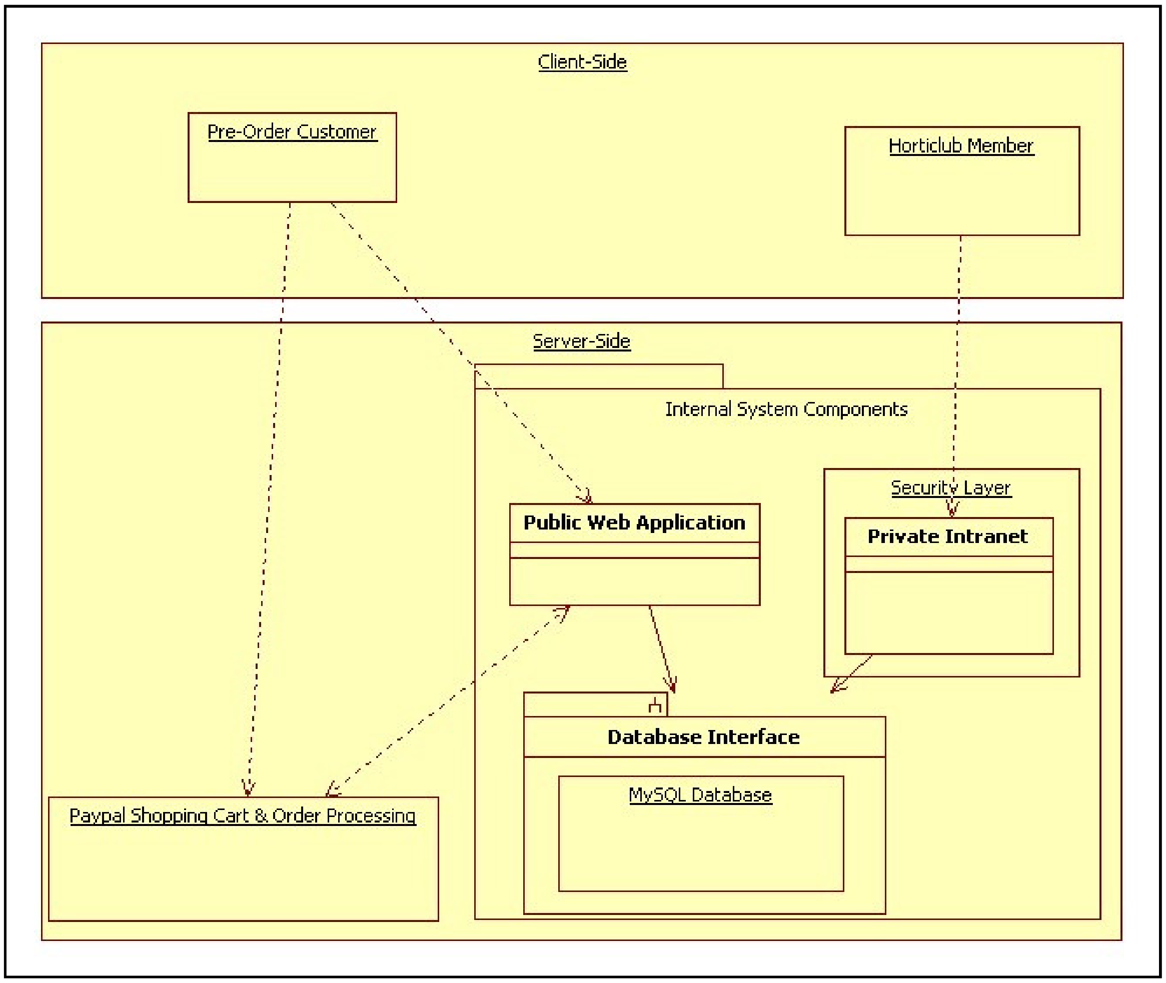


Figure 10. System Architecture of the Horticulture Club Sales Assistant.

3.4.3.1 Client Side

Users of this system act as the client in the server/client interaction, Figure 11. This system has two separate types of users: “Pre-Order Customer” and “Horticlub Member”. The users access the server side of the Horticulture Club Sales Assistant through different interfaces. The “Horticlub Member” is privileged and able to access private sections on the server side. “Pre-Order Customers” have no privileges and access the server side through a public interface.

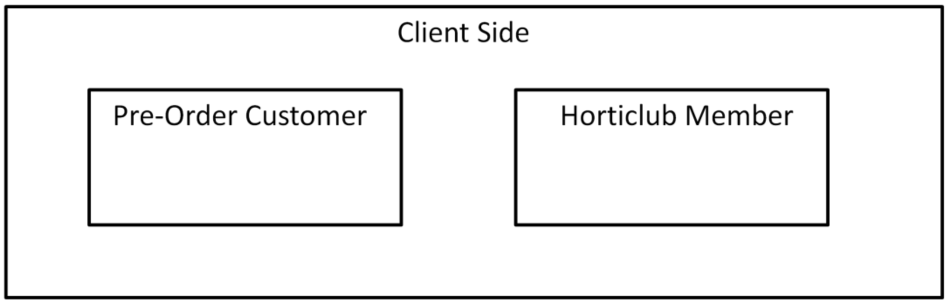


Figure 11. Client side depiction of the Horticulture Club Sales Assistant

A “Pre-Order Customer” is a general user who is using the system to either browse or purchase from the Horticulture Club. “Pre-Order Customers” are guided to a system welcome page and can navigate through inventory using simple search features. Items can be purchased by “Pre-Order Customers” from the public portion of the server side. A “Horticlub Member” is a member of the Horticulture club that has some form of administrative access to the system. Horticlub Members are able to access the private portion of the system and manage inventory, view sales, create reports and manage other “Horticlub Members”.

3.4.3.2 Server Side

The two main interfaces of the server are the ‘Public Web Application’ and the ‘Private Intranet’. A depiction of server side components is shown in Figure 12. The ‘Public Web Application’ is accessed by a ‘Pre-Order Customer’ from the client side of the system. Navigating through the ‘Public Web Application’ requires access to the ‘MySQL Database’ through the ‘Database Interface’. The ‘Database Interface’ abstracts database connections and allows for the database to be interchanged with minimal effort. Paypal is used by the ‘Public Web Application’ to handle monetary transactions needed for the pre-order functionality within the system.

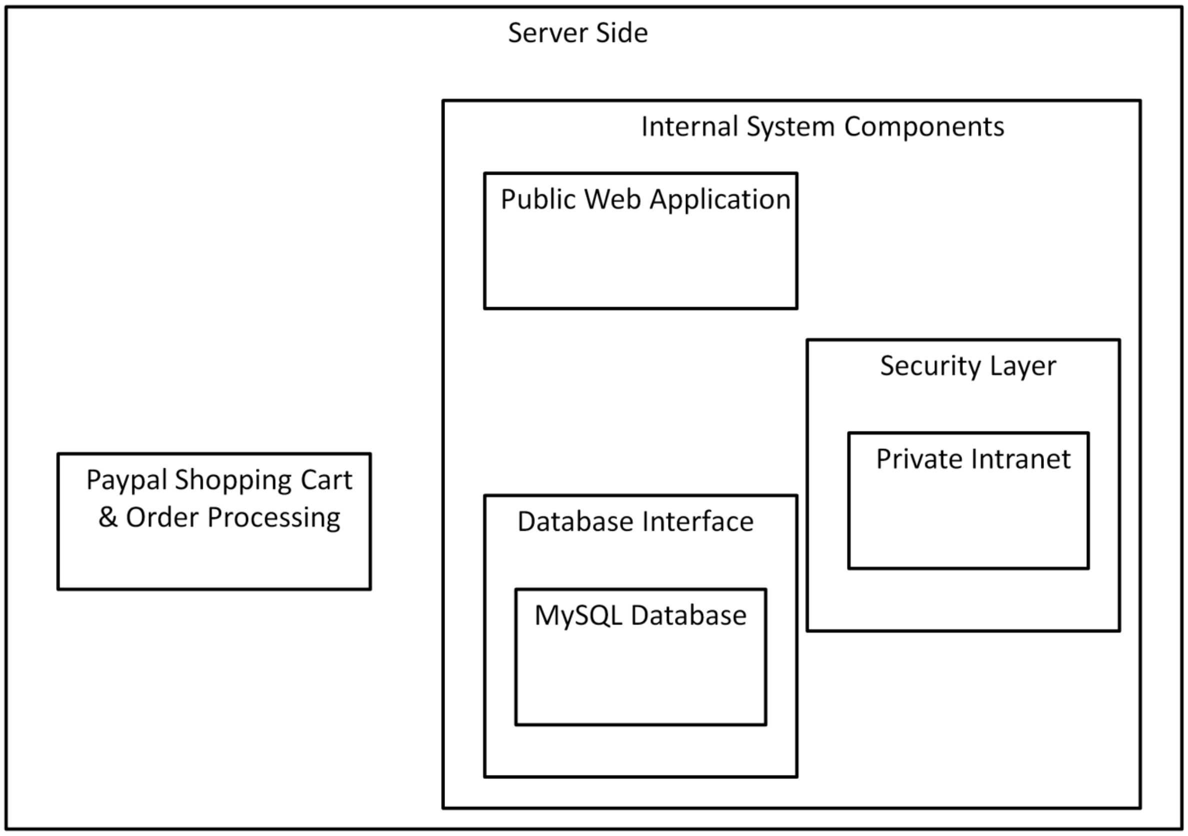


Figure 12. Server side depiction of the Horticulture Club Sales Assistant.

The 'Private Intranet' component also accesses the system database. 'Horticlub Member' users access the 'Private Intranet' by going through a 'Security Layer'. The 'Security Layer' forces user authentication as well as input parsing to protect the system from malicious input. In addition to the pre-order functionality, the system also supports point-of-sale functionality. Point-of-sale is implemented through the 'Private Intranet' and is controlled by a 'Horticlub Member'. Administrative functions such as user and inventory management are performed through the 'Private Intranet'.

4. Partial VV&T

Chapter 4 describes the application of Partial VV&T to the Horticulture Club Sales Assistant software system. Section 4.1 provides an overview of the development artifacts available from the Horticulture Club Sales Assistant program. Section 4.2 introduces a methodology of application for applying Partial VV&T to a software system. Section 4.3 describes the Partial VV&T analysis of each development artifact and notes important changes among each artifact. Section 4.4 details the results of applying Partial VV&T to the system. Section 4.5 discusses potential improvements to the Partial VV&T application process.

4.1 Artifact Overview

Partial VV&T is performed by applying violable constraints/assumptions to development artifacts of a software system in order to determine if any software vulnerabilities exist within the system. Several different methods are available for designing and developing a software system. As such, each development process can produce different sets of development artifacts. Frequency of artifact generation during the development process, as well as the content of an artifact, is at the discretion of software developers. Artifacts available to this research include a product overview document, a requirements document, design documents, integration and testing documents, source code, and the executable binary of the system.

Generation of the product overview document for the Horticulture Club Sales Assistant comes before that of the requirements document during the development of the system. This document provides a general overview of the goals of the system being developed as well as a list of proposed features of the system. Listed goals and proposed features of the system can be used to infer issues relating to constraints/assumptions. While the product overview might not contain as much security related information as the requirements and design documents, review of this document can still reveal constraint/assumption related issues that can be revisited when reviewing more development intense documents.

A requirements document is an artifact that is included in a majority of software development processes and is included for review under Partial VV&T. The requirements document for the Horticulture Club Sales Assistant goes into more detail about the functionality of the system than the product overview document. Expected system behavior in response to user actions is described within the document. Database and web application interfaces are preliminarily structured. Use cases for the system are included in this document as well as a list of expected variables used to execute use cases and other system functionality.

As with the requirements document, design documents are commonly associated with the development of a software system. Therefore, it is prudent that any available design documents produced during the development of the Horticulture Club Sales Assistant be reviewed under Partial VV&T. The Horticulture Club Sales Assistant had two design documents associated with its development. One design document contains a high-level design of the system and provides descriptions of each high-level component. The high-level description is decomposed and described at a lower level in the second design document. This lower-level design document contains detailed use cases, data flow diagrams, a data dictionary, and detailed component description.

The integration and testing document for the Horticulture Club Sales Assistant was produced during coding and testing of the system. This document notes the differences between the design documents and the source code. Differences between the design documents and the source code of the system arise from problems encountered when translating the design to functional code. Changes in the design of the system can lead to an introduction of software vulnerabilities to the system. Therefore, the integration and testing document of the system is reviewed under Partial VV&T.

Lastly, the source code of the Horticulture Club Sales Assistant program is reviewed under Partial VV&T. While the executable binary of the system can be reviewed under Partial VV&T, review of the source code eliminates the need to review the executable binary [2]. Source code review does not include review of the source code of external components used by the system. However, it is possible that the source code of external components could introduce software vulnerabilities to a system. This possibility is explored when reviewing security implications of external components as they arise throughout the development artifacts.

4.2 Methodology of Application

Prior to this research, no method exists with which to apply Partial VV&T to a software system. Several approaches are explored in finding a method that supports an intuitive and thorough application of Partial VV&T. The approach must involve both the violable constraints/assumptions and the development artifacts in producing viable results that can be used to identify software vulnerabilities. Concept approaches of application are attempted before being dismissed, allowing for an acceptable approach for applying Partial VV&T to emerge.

One failed approach takes all available development artifacts and treats them as one entity. The development artifacts are reviewed for content relating to the violable constraints/assumptions. A list is created containing the constraints/assumptions as well as any instances of the constraints/assumptions found during review of the development artifacts. Instances found pertaining to the constraints/assumptions are placed in the list under their respective constraint/assumption.

The reason this approach failed is the results of applying Partial VV&T to the system are not artifact specific. In this manner, vulnerabilities cannot be observed as they appear during the development process. This makes it more difficult to identify the origin of the vulnerability and correct it. Review of the development artifacts in the failed approach, however, is able to identify major instances where the software system either conforms to or violates the violable constraints/assumptions.

This failed attempt is a “stepping stone” to the final solution approach used to apply Partial VV&T. In the solution approach, each document is reviewed individually and has its own associated list in which to place constraint/assumption related instances. An advantage to this approach is that any discovered vulnerabilities can be traced back to an originating design document. To review a development artifact two things are needed: a list in which to track instances of the constraints/assumptions, and a nomenclature for classifying the status of the software system as it pertains to constraints/assumptions.

4.2.1 Artifact Constraint/Assumption Review Table

The review list for each development artifact takes the form of a table with three columns, Table 1. The left most column of the table is reserved for constraints/assumptions. Each constraint/assumption is listed in the table allowing each one to be reviewed for each development artifact. There are as many rows in the table as there are constraints/assumptions. The middle column of the table is used to denote related instances within the development artifact that are relevant to the associated constraint/assumption. For the related instance column, within a constraint/assumption row, there may exist sub rows to represent multiple related instances of a constraint/assumption. Finally, the right most column denotes the status of a software system related instance to its respective constraint/assumption.

Table 1. Example Constraint/Assumption Review Table.

Development Artifact		
Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
Constraint/Assumption 1	Constraint/Assumption Related Instance	Status
	Constraint/Assumption Related Instance	Status
Constraint/Assumption 2	Constraint/Assumption Related Instance	Status
	Constraint/Assumption Related Instance	Status

An advantage to reviewing artifacts in this format is the expandability of the table. Expandability allows this approach to applying Partial VV&T to grow as the constraint/assumption technology grows. To add a new constraint/assumption to the review table, a new row for the constraint/assumption can simply be added. It is also possible to add additional columns to the table, if needed.

4.2.2 Status Nomenclature

The most important column in the artifact review table is the "Constraint/Assumption Status" column. This column represents the status of the software system as it pertains to the constraint/assumption for the given development artifact. The status of a constraint/assumption related instance can change throughout the course of the development artifacts. The nomenclature must appropriately identify the software system's status. The following phrases are used in the third column of the constraint/assumption review tables to describe the software system's status.

N/A to Process describes a constraint/assumption that is not applicable to the software system being reviewed. For example, if a software system does not utilize random numbers, then constraints/assumptions pertaining to random number generation are not applicable to the software system.

N/A to "Development Artifact" describes a constraint/assumption that is not applicable to the development artifact being reviewed. In the review tables for the development artifacts, "Development Artifact" is replaced with the actual artifact's name. An example of this status is a development artifact that lends no inclination to the use of one-time pads by the software system; however, there is evidence that cryptographic protocols are used in the software system. Therefore, the use of one-time pads cannot be ruled out.

Constraint/Assumption Not Met describes a constraint/assumption that is not met by the software system. Either the software system does not enforce a constraint assumed by a resource or the software system makes an incorrect assumption about a resource's usage. For example, there is a constraint that states any integer variable used to hold a quantity or length must not hold a negative number. The constraint is not met if the integer variable is allowed to hold a negative number.

Related Instance => Constraint/Assumption describes a constraint/assumption that is not mentioned by a development artifact, but is implied through wording in the artifact. For example, there is a violable assumption that the software process will be able to use the network interface to send and receive data. If a software system is noted as being a networked process or web application, it can be implied that the software system will ensure that the network interface is accessible.

Constraint/Assumption Met describes a constraint/assumption that is met by the software system. For example, there is a constraint that states any integer variable used to hold a quantity or length must not hold a negative number. The constraint is met if the integer variable is not allowed to hold a negative number.

4.3 Partial VV&T Analysis

Partial VV&T analysis of the Horticulture Club Sales Assistant program involves reviewing five development artifacts; a product overview document, a requirements document, a design document, an integration and testing document, and the source code. Each of these artifacts is reviewed using constraint/assumption review tables. Review of the development artifacts provides insight into where software vulnerabilities exist, if any, and where they originate during the development process. In this section, examples from the artifact constraint/assumption review tables are highlighted and explained. Full constraint/assumption review tables can be found in Appendix A.

4.3.1 Product Overview Document

The product overview document for the Horticulture Club Sales Assistant program is a relatively small document. However, keywords in the document provide significant insight into what can be expected of the software system. From these keywords, knowledge of accompanying constraint/assumption related instances can be inferred. Inferring knowledge of constraint/assumption related instances is acceptable at this stage of the development process as no detailed descriptions of system requirements or functionality have been put forth. Keywords the product overview document include "Automated Inventory Tracking" and "Credit Card Capability". The full constraint/assumption review table for the product overview document is shown as Table A-1 in Appendix A.

4.3.1.1 Dynamic Memory

From the keyword "Automated Inventory Tracking", several features of the system can be implied that have dynamic memory constraint/assumption related instances. A subsection of dynamic memory constraints/assumptions from the constraint/assumption review table for the product overview document is shown in Table 2. "Automated Inventory Tracking" implies that storage will be provided and managed by the system. In order for the system to store this information, input of data will need to occur. The word "Inventory" implies that quantities of objects will be stored and managed. Functionality implied from the "Automated Inventory Tracking" keyword relates to several dynamic memory constraints/assumptions.

Table 2. Dynamic Memory Constraints/Assumptions from review of the Product Overview Document.

Dynamic Memory – Constraints/Assumptions	Related Instance	Constraint/Assumption Status
10. Data accepted by the process must not be interpreted as a format string by the I/O routines.	Being a process that allows online ordering, the process will have input capability and will need to prevent against format strings.	Related Instance => Constraint/Assumption
11. The value of an integer variable/expression (signed and unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer variable.	The process will implement automated inventory tracking which implies that numerical data will be stored that will require boundary checks.	Related Instance => Constraint/Assumption
12. An integer variable/expression used by the process as the index to a buffer must only hold values that allow it access to the memory locations assigned to the buffer.		N/A to Product Overview
13. An integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values.	The implementation of automated inventory tracking as well as the process being an online ordering system indicate that quantities will be used and need to be protected with boundary checks.	Related Instance => Constraint/Assumption

Three of the dynamic memory constraints/assumptions shown in Table 2 have a "Related Instance => Constraint/Assumption" status. This status is given based on functionality implied from the "Automated Inventory Tracking" keyword. The constraint/assumptions cannot be given "Constraint/Assumption Met" statuses because there is evidence from the product overview document that these constraints/assumptions are met. Likewise, they cannot be given a "Constraint/Assumption Not Met" status. If vulnerabilities arise relating to the dynamic memory constrains/assumptions in Table 2, the origins of the vulnerabilities can be traced back to the product overview document.

Dynamic memory constraint/assumption number 12 in Table 2 is given a "N/A to Product Overview" status. It cannot be implied from the product overview document that the software process will or will not use indices when accessing buffers. Because it is possible that the software process may use indices when accessing buffers, the dynamic memory constraint/assumption is only not applicable to the product overview document instead of being not applicable to the whole process.

4.3.1.2 Static Memory

It cannot be implied from the product overview document that the Horticulture Club Sales Assistant uses static memory. Because it cannot be determined, the static memory constraints/assumptions in the review table for the product overview document are given the status “N/A to Product Overview”.

4.3.1.3 Filesystem

Examples of filesystem constraints/assumptions from the product overview constraint/assumption review table are shown in Table 3. Related instances to filesystem constraints/assumptions 2 and 3 in Table 3 are inferred from the product overview document. The product overview document discusses the generation and use of sales reports by the software process. Filesystem constraint/assumption number 2 in Table 3 can be implied from the fact that the software process will generate and use files. Generating and using files implies that appropriate access permissions will be used. Because use of privileges is implied, filesystem constraint/assumption number 2 is given a “Related Instance => Constraint/Assumption” status. This status also applies to filesystem constraint/assumption number 3 in Table 3. Dynamically creating new files within a software process implies that each new file will have a unique filename.

Table 3. Filesystem Constraints/Assumptions from review of the Product Overview Document

Filesystem - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
2. Access permissions of the files/directories being used by the process are such that only the required principals have access to them.	Any Data being stored by the process that is deemed to be access sensitive should incorporate appropriate access permissions.	Related Instance => Constraint/Assumption
3. A file being created by the process does not have the same name as an already existing file.	If multiple sales reports are made and stored by the process then the filenames of the sales reports should not be the same.	Related Instance => Constraint/Assumption
6. A file being used by the process cannot be observed/modified/replaced while the process is in execution.	Being a web application, almost all modification of program files will occur while the process is in execution (online).	N/A to Process
7. A file/directory being used by the process and stored on the file-system (information used by the process over multiple runs) cannot be observed/modified/replaced in-between these runs.	Being a dynamic online web application, files will be updated continually.	N/A to Process

Filesystem constraints/assumptions 6 and 7 in Table 3 are given the status of "N/A to Process". The product overview document mentions that the Horticulture Club Sales Assistant will be a dynamic online web application. Being a dynamic online web application, files used by the software process are able to be changed while the process is in execution and between runs of the process.

4.3.1.4 Network Interface

The Horticulture Club Sales Assistant is described in the product overview document as being a dynamic web application. Being a web application implies several network interface related instances. Table 4 shows example network interface related instances from the product overview constraint assumption review table. Web applications incorporating monetary transactions pose a significant security risk if the data is able to be intercepted and read.

Network interface constraint/assumption 3 in Table 4 is implied from the product overview document keyword “Credit Card Capability”. The related instance has a status of “Related Instance => Constraint/Assumption” because there is no indication that data sent over the network is protected.

Table 4. Network Interface Constraints/Assumptions from review of the Product Overview Document.

Network Interface - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
3. The data sent by the software process via the network interface will not be read/modified before it reaches its destination.	The process will be networked and should be protected from possible man-in-the-middle attacks.	Related Instance => Constraint/Assumption
4. The software process will be able to utilize the network interface to send and receive data.	The process will need to be able to utilize the network in order to function properly.	Related Instance => Constraint/Assumption

Network interface constraint/assumption 4 in Table 4 also has a status of “Related Instance Constraint/Assumption”. There is no mention in the product overview document of ensuring access to the network interface. However, it is implied through keywords such as “Credit Card Capability” that developers will ensure network access and be able to handle cases where the network interface is not available.

4.3.1.5 Randomness Resources

There are no randomness resources related instances in the product overview document. “Credit Card Capability” of the Horticulture Club Sales Assistant implies some sort of data security. However, the way data is secured cannot be determined from the product overview document. Use of passwords by the Horticulture Club Sales Assistant is not mentioned in the product over view document either. All randomness resources constraints/assumptions are given the status of “N/A to Product Overview”.

4.3.1.6 Cryptographic Algorithms and Protocols

The "Credit Card Capability" keyword implies sensitivity and confidentiality. As such, associated constraints/assumptions from the cryptographic algorithms and protocol section can be implied. Because the algorithm or protocol being used is not mentioned and cannot be inferred from the text, most of the cryptographic algorithms and protocol constraints/assumptions are not applicable to the product overview document. Examples of cryptographic algorithms and protocols related instances from the product overview document are shown in Table 5.

Table 5. Cryptographic Algorithms and Protocols Constraints/Assumptions from review of the Product Overview Document.

Cryptographic Algorithms and Protocols - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
4. The process cannot use encryption to ensure data integrity.	The process will implement credit card capability and must ensure the integrity of all monetary transactions.	Related Instance => Constraint/Assumption
8. The process cannot use obfuscation instead of encryption to ensure confidentiality.	All monetary transactions by the process must ensure confidentiality.	Related Instance => Constraint/Assumption

Both constraint/assumption related instances in Table 5 are derived from the "Credit Card Capability" keyword. Integrity of monetary transactions is implied because if data integrity is ignored, then account information can be manipulated and the system functionality compromised. Confidentiality protects account information from being read by potential attackers. All other cryptographic algorithms and protocols constraints/assumptions are deemed "N/A to Product Overview". These constraints/assumptions are not applicable because there is no mention of cryptographic algorithms or protocols being used by the Horticulture Club Sales Assistant in the product overview document.

4.3.2 Requirements Document

The requirements document expands on the product overview document. Features proposed in the product overview document are detailed further and additional features are discussed. Paypal is used by the Horticulture Club Sales Assistant to facilitate monetary transactions. A MySQL database is used by the Horticulture Club Sales Assistant to store inventory and sales information. A data dictionary is included in the requirements document and lists expected variables needed to implement functionality of the Horticulture Club Sales Assistant. The full constraint/assumption review table for the requirements document is shown in Table A-2 in Appendix A.

4.3.2.1 Dynamic Memory

The presence of the data dictionary in the requirements document introduces several dynamic memory related instances. Example dynamic memory related instances from the requirements document constraint/assumption review table are shown in Table 6. The data dictionary details variable types, lengths and expected values to be used in the coding of the Horticulture Club Sales Assistant. Related instances for dynamic memory constraints/assumptions 1 and 11 in Table 6 are implied by the data dictionary. Knowing expected data values and lengths allows for appropriate boundary checks to be put in place. These boundary checks will prevent dynamic memory constraints/assumptions 1 and 11 from being violated.

Table 6. Dynamic Memory Constraints/Assumptions from review of the Requirements Document.

Dynamic Memory - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
1. Data Accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to the buffer.	Data Dictionary specifies data types for proposed variables to be used by the process. This also can be used to specify any needed buffer lengths by knowing the data types and their expected use.	Related Instance => Constraint/Assumption
7. A pointer variable being used by the process references a legal memory location.		N/A to Requirements Specification
11. The value of an integer variable/expression (signed and unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer variable.	The Data Dictionary specifies the integer variables to be used by the process and can allow for appropriate boundary checks.	Related Instance => Constraint/Assumption

Dynamic memory constraint/assumption 7 in Table 6 is "N/A to Requirements Specification." The data dictionary describes expected variables used in the system, but does not state whether or not any of the variables will be pointer variables. The use of pointer variables by the Horticulture Club Sales Assistant cannot be implied and cannot be ruled out. Therefore, pointer related dynamic memory constraints/assumptions are considered "N/A to Requirements Specification."

4.3.2.2 Static Memory

It cannot be implied from the requirements document that the Horticulture Club Sales Assistant uses static memory. Because it cannot be determined, the static memory constraints/assumptions in the review table for the requirements document are given the status "N/A to Requirements Specification".

4.3.2.3 Filesystem

The requirements document reveals details of the Horticulture Club Sales Assistant that allows the software process to meet two filesystem constraints/assumptions. Table 7 shows the two filesystem constraints/assumptions from the requirements document constraint/assumption review table. Authorization of users is a feature introduced in the requirements document. More specifically, authorized users will have restricted access to system functionality such as inventory management and sales report generation. Restricting access of files to only required principals meets the two constraints/assumptions.

Table 7. Filesystem Constraints/Assumptions from review of the Requirements Document.

Filesystem - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
1. Access permissions assigned to newly created files/directories are such that only the required principals have access to them.	Authorized users of the process will have delegated privileges that will restrict the access of files to only required principals	Constraint/Assumption Met
2. Access permissions of the files/directories being used by the process are such that only the required principals have access to them.	Authorized users of the process will have delegated privileges that will restrict the access of files to only required principals	Constraint/Assumption Met

4.3.2.4 Network Interface

The related instance for network interface constraint/assumption 2 from the requirements document constraint/assumption review table, Table 8, is affected by the requirements document. The requirements document briefly details a private and public portion of the Horticulture Club Sales Assistant. Allowing public access to the Horticulture Club Sales Assistant increases the possibility of an illegitimate client or peer accessing the system. However, the data dictionary in the requirements document describes the expected formats and lengths for data received by the process. Because the formats and lengths are known, network interface constraint/assumption 2 in Table 8 is implied.

Table 8. Network Interface Constraints/Assumptions from review of the Requirements Document.

Network Interface - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
2. The data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.	Part of the process is able to be accessed by the public, meaning there is no access restriction among who can use the public portion of the process. The data dictionary can aid in ensuring data is of the correct format and length.	Related Instance => Constraint/Assumption

4.3.2.5 Randomness Resources

A major change between the product overview document and the requirements document is the change in the status of randomness resources constraints/assumptions and cryptographic algorithms and protocols constraints/assumptions. Expanding on the keyword “Credit Card Capability”, the requirements document states that PayPal will be used for processing payment information. Implementation of the HTTPS protocol used by PayPal requires the use of randomness resources. However, this is not the responsibility of the software process. The strength of HTTPS is dependent upon end-to-end user support of the protocol. All randomness resources constraints/assumptions in the requirements document review table pertaining to random numbers, accessing entropic data and estimating entropy are “N/A to Process”.

Table 9. Randomness Resources Constraints/Assumptions from review of the Requirements Document.

Randomness Resources – Constraints/Assumptions	Related Instance	Constraint/Assumption Status
5. User selected passwords/keys will have sufficient amount of entropy.	The process will require a login of a user name and a password.	Related Instance => Constraint/Assumption

Randomness resources constraint/assumption 5 from the requirements document review table, Table 8, is given the status “Related Instance => Constraint/Assumption”. The constraint/assumption is implied due to the authorized user feature mentioned in the requirements document. To authorize users, a username/password combination will be used.

Entropy of the passwords is not specified in the requirements document but is implied as it is expected to appear in later design artifacts.

4.3.2.6 Cryptographic Algorithms and Protocols

Two Cryptographic algorithms and protocols constraints/assumptions are met during review of the requirements document, Table 10. Cryptographic algorithms and protocols constraints/assumptions 4 and 8 in Table 10 are met by the use of the HTTPS protocol by the Horticulture Club Sales Assistant. The HTTPS protocol ensures data integrity through authentication and confidentiality is ensured using a hashed message digest. External components of the Horticulture Club Sales Assistant (Paypal) also use the HTTPS protocol.

Table 10. Cryptographic Algorithms and Protocols Constraints/Assumptions from review of the Requirements Document.

Cryptographic Algorithms and Protocols - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
4. The process cannot use encryption to ensure data integrity.	The HTTPS protocol is used by the process and is useful in the prevention of data tampering. This is done using a hashed message digest and not encryption.	Constraint/Assumption Met
8. The process cannot use obfuscation instead of encryption to ensure confidentiality.	The HTTPS protocol is used by the process and provides encryption for data. Encryption provides confidentiality	Constraint/Assumption Met

4.3.3 Design Document

The design document of the Horticulture Club Sales Assistant program provides several details of the system that have an effect on the status of several constraints/assumptions. According to the design document, the Horticulture Club Sales Assistant is coded using the Java programming language and Apache Tomcat is used to provide server functionality. Specific features of the software system mentioned in the design document allows related instances in the design document constraint/assumption review table to be more process specific. The full design document constraint/assumption review table can be seen as Table A-3 in Appendix A.

4.3.3.1 Dynamic Memory

One feature of the Java programming language is that it does not use pointer variables. Dynamic memory constraints/assumptions relating to pointer variables, Table 11, are given the status "N/A to Process". Other programming languages such as C or C++ that utilize pointer variables would require further review of these dynamic memory constraints/assumptions. A status of "N/A to Process" indicates that the dynamic memory constraints/assumptions in Table 11 will not be revisited unless another development artifact presents evidence indicating that further review of the constraints/assumptions is necessary.

Table 11. Dynamic Memory Constraints/Assumptions from review of the Design Document.

Dynamic Memory – Constraints/Assumptions	Related Instance	Constraint/Assumption Status
7. A pointer variable being used by the process references a legal memory location.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
8. A memory pointer returned by the underlying operating system does not point to zero bytes of memory.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
9. A pointer variable being used by the process cannot reference itself.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process

Another feature of the Java programming language is that Java incorporates an automatic memory management system. This system obviates the need for the software system to allocate and free memory manually. Two dynamic memory constraints/assumptions are affected by this feature, Table 12. Dynamic memory constraint/assumption 4 in Table 12 is implied because of the way the Java automatic memory management system handles requests for memory. The software system does not allocate memory directly and thus, must go through the automatic memory management system for any memory requests. The memory management system returns an error if there is no memory available for allocation. Dynamic memory constraint/assumption 4 is not met if this error is not addressed. Because the Horticulture Club Sales Assistant is programmed using Java, knowledge of the memory management system is implied.

Table 12. Dynamic Memory Constraints/Assumptions from review of the Design Document.

Dynamic Memory - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
4. The process will be provided with the dynamic memory that it requests.	Java implements an automatic memory management system. This system throws an error when the process is not able to be provided with memory that it requests.	Related Instance => Constraint/Assumption
6. Data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.	Java implements an automatic memory management system. This includes automatic garbage collection. Garbage collection occurs when reference to a process object no longer exists. To ensure data cannot be accessed, the memory should be overwritten before being freed.	Related Instance => Constraint/Assumption

Dynamic memory constraint/assumption 6 in Table 12 also pertains to the automatic memory management system of the Java programming language. In addition to handling memory allocation, the automatic memory management system also handles automatic garbage collection. For Java, an object is considered for garbage collection when references to the object no longer exist. Memory used by the program should be overwritten before it is eligible for garbage collection. Because knowledge of the Java automatic memory management system is implied, dynamic memory constraint/assumption 6 in Table 12 is implied.

4.3.3.2 Static Memory

It cannot be implied from the design document that the Horticulture Club Sales Assistant uses static memory. Because it cannot be determined, the static memory constraints/assumptions in the review table for the design document are given the status “N/A to Design Document”.

4.3.3.3 Filesystem

Filesystem constraint/assumption 4, Table 13, is affected by review of the design document. The design document details files to be used by the process. Only picture files are to be uploaded to the Horticulture Club Sales Assistant. Picture files are used for inventory purposes. Knowing the file type allows for restrictions to be placed on the file upload functionality. No evidence exists in the design document indicating that restriction of file uploading occurs. Therefore, filesystem constraint/assumption 4 in Table 13 is given the status “Related Instance => Constraint/Assumption.”

Table 13. Filesystem Constraints/Assumptions from review of the Design Document.

Filesystem - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
4. A filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permissions.	Files being used by the process are expected to be picture and database files. Picture files are uploaded to the process and can be enforced. The database should not affect this constraint/assumption.	Related Instance => Constraint/Assumption

4.3.3.4 Network Interface

Further elaboration in the design document details how the Horticulture Club Sales Assistant meets network interface constraints/assumptions 1 and 3 in Table 14. A public and private portion of the Horticulture Club Sales Assistant is described in the requirements document. In the design document the HTTPS protocol is specified for the private portion of the Horticulture Club Sales Assistant. The HTTPS protocol is also used by Paypal in executing monetary transactions. Using the HTTPS protocol allows the Horticulture Club Sales Assistant to ensure that data being transferred is neither read nor modified by anyone other than the intended recipient.

Table 14. Network Interface Constraints/Assumptions from review of the Design Document.

Network Interface - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
1. The data received by the software process through the network interface is neither read nor modified by anyone other than the intended recipient.	The software process uses the HTTPS protocol to secure the private web portion of the system as well as monetary transactions. HTTPS protects data from being read and modified.	Constraint/Assumption Met
3. The data sent by the software process via the network interface will not be read/modified before it reaches its destination.	The software process uses the HTTPS protocol to secure the private web portion of the system as well as monetary transactions. HTTPS protects data from being read and modified.	Constraint/Assumption Met
5. The byte order of numerical data accepted from the network interface is same as that of the host machine.	Java uses network byte order, meaning there is no need for conversion.	Constraint/Assumption Met

Network interface constraint/assumption 5 in Table 14 is met by the Java programming language. Java uses network byte order to store data and does not require conversion when receiving data over the network.

4.3.3.5 Randomness Resources

Only one difference appears in the randomness resources section between the requirements document and design document review tables. The design document states that users will log into the Horticulture Club Sales Assistant using their Virginia Tech user ID and password. Because the origin of the username password combinations is known, randomness resources constraint/assumption 5 in Table 15 is met. Virginia Tech has requirements for password generation that provide a sufficient amount of entropy [60].

Table 15. Randomness Resources Constraints/Assumptions from review of the Design Document.

Randomness Resources - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
5. User selected passwords/keys will have sufficient amount of entropy.	The user login will utilize a Virginia Tech user ID and password. The entropy of the password must fall under Virginia Tech requirements.	Constraint/Assumption Met

4.3.3.6 Cryptographic Algorithms and Protocols

Cryptographic algorithms and protocols constraint/assumption 6 in TABLE is "N/A to Process" after review of the design document. Review of the design document no evidence of the use of one time pads by the Horticulture Club Sales Assistant. Details provided in the design document allow the status "N/A to Process" to be applied to this constraint/assumption. All encryption and data security in the Horticulture Club Sales Assistant is provided by the HTTPS protocol.

Table 16. Cryptographic Algorithms and Protocols Constraints/Assumptions from review of the Design Document.

Cryptographic Algorithms and Protocols - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
6. The process cannot use one time pads to encrypt a large quantity of data.	There is no indication of the use of one time pads by the process	N/A to Process

4.3.4 Integration and Testing Document

Few differences are noted between reviews of the design document and the integration and testing document. The integration and testing document was generated after the coding of the software system in order to note changes to the design during implementation as well as give an overview of the implementation. An admittance of the integration and testing document is that the software system is weak on input validation. Neglecting input validation can lead to a number of constraints/assumptions not being met. The full integration and testing document constraint/assumption review table can be seen as Table A-4 in Appendix A.

4.3.4.1 Dynamic Memory

Review of the integration and testing document does not produce any status changes among the dynamic memory constraint/assumption related instances. However, the integration and testing document does mention that input validation in the Horticulture Club Sales Assistant is lacking. Dynamic memory constraints/assumptions that may be affected by this admittance are shown in Table 17. Dynamic memory constraints/assumptions in Table 17 retain their implied status even though input validation for the Horticulture Club Sales Assistant is noted as lacking. Status of these dynamic memory constraint/assumption related instances remain unchanged because it is not known to what extent input validation is lacking. Earlier design documents mention the use of a security layer by the Horticulture Club Sales Assistant and the integration and testing document does not mention that the security layer will no longer be used. Until it is known where input validation is lacking or it is implied that a security layer is no longer being used by the Horticulture Club Sales Assistant, the dynamic memory constraints/assumptions in Table 17 retain their "Related Instance => Constraint/Assumption" status.

Table 17. Dynamic Memory Constraints/Assumptions from review of the Integration and Testing Document.

Dynamic Memory - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
2. The process will not interpret data present on the dynamic memory as executable code.	The security layer of the process will parse all data input to remove malicious information such as code injection.	Related Instance => Constraint/Assumption
10. Data accepted by the process must not be interpreted as a format string by the I/O routines.	The security layer of the process will parse all data input to remove malicious information such as code injection. However, the process is noted as lacking input validation.	Related Instance => Constraint/Assumption
13. An integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values.	The process will use quantities for keeping track of inventory. However, form validation is noted as lacking, leading to the possibility of negative quantities to be input to the process.	Related Instance => Constraint/Assumption

4.3.4.2 Static Memory

It cannot be implied from the integration and testing document that the Horticulture Club Sales Assistant uses static memory. Because it cannot be determined, the static memory constraints/assumptions in the review table for the integration and testing document are given the status "N/A to integration and testing Document".

4.3.4.3 Filesystem

Image files are able to be uploaded to the Horticulture Club Sales Assistant for inventory purposes. The integration and testing document mentions a naming convention for uploaded picture files that gives each uploaded file a unique filename. Assigning unique filenames to uploaded files meets filesystem constraint/assumption 3, Table 18. Also shown in Table 18, filesystem constraint/assumption 10 is no longer applicable to the process after review of the integration and testing document. Only picture files are uploaded to the Horticulture Club Sales

Assistant and files used by the system have not been described as having obscure or proprietary format.

Table 18. Filesystem Constraints/Assumptions from review of the Integration and Testing Document.

Filesystem - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
3. A file being created by the process does not have the same name as an already existing file.	Files that are uploaded to the process and named by the process use a unique item ID number as their filename.	Constraint/Assumption Met
10. A file having proprietary or obscure format cannot be understood or modified.	There is no indication of the process using proprietary or obscure formats for its files.	N/A to Process

4.3.4.4 Network Interface

Network interface constraint/assumption 2, Table 19, is not met after review of the integration and testing document. The integration and testing document describes the Horticulture Club Sales Assistant as lacking input validation. Dynamic memory constraints/assumptions affected by the lack of input validation are still implied due to the vagueness of the statement in the integration and testing document. Network interface constraint/assumption 2 cannot be implied as it encompasses all input to the Horticulture Club Sales Assistant. If input validation is lacking in any part of the Horticulture Club Sales Assistant then the constraint/assumption cannot be fully met. Hence, network interface constraint/assumption 2 is not met by the Horticulture Club Sales Assistant.

Table 19. Network Interface Constraints/Assumptions from review of the Integration and Testing Document.

Network Interface - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
2. The data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.	Part of the process is able to be accessed by the public, meaning there is no access restriction among who can use the public portion of the process. The process is noted as lacking input validation leading to the possibility of incorrect formats or lengths of data to be accepted and used by the process.	Constraint/Assumption Not Met

4.3.4.5 Randomness Resources

Review of the integration and testing document does not introduce any new or affect any current randomness resources constraint/assumption related instances. A majority of randomness resources constraints/assumptions are not applicable to the Horticulture Club Sales Assistant. After reviewing the integration and testing document, there is no indication that user selected passwords do not still fall under Virginia Tech requirements.

4.3.4.6 Cryptographic Algorithms and Protocols

Like randomness resources constraints/assumptions, review of the integration and testing document does not introduce and new or affect any current cryptographic algorithms and protocols constraint/assumption related instances. Use of the HTTPS protocol relegates many cryptographic algorithms and protocols constraints/assumptions not applicable to the Horticulture Club Sales Assistant. However, use of the HTTPS protocol allows the Horticulture Club Sales Assistant to ensure data integrity and confidentiality.

4.3.5 Code Review

Code review for the Horticulture Club Sales Assistant program is performed manually. Reviewing source code manually is necessary when using the constraints/assumptions technology as currently no code analyzer exists that incorporates this technology. Files analyzed in the code review consist of JSP files and Java class files used by the Horticulture Club Sales Assistant. For the code review, source code files are partitioned into resource specific categories before they are reviewed. For example, files pertaining to database management are grouped together. Grouping of files allows for a more resource concentrated constraint/assumption review of a file. However, files are reviewed for all constraint/assumption related instances. The full source code constraint/assumption review table can be seen as Table A-5 in Appendix A.

4.3.5.1 Dynamic Memory

Several dynamic memory constraints/assumptions implied by review of the integration and testing document are either met or not met by code review, Table 20. The lack of input validation mentioned in the integration and testing document is evident throughout the source code. Dynamic memory constraints/assumptions 2 and 13 are not met due to the lack of input validation. In some parts of the system there is no input validation, making it possible to introduce executable code on the dynamic memory. Also, there are no constraints for integer variables being used to represent inventory. A negative number can easily be inserted to represent a quantity in the Horticulture Club Sales Assistant's inventory.

Table 20. Dynamic Memory Constraints/Assumptions from review of the Source Code.

Dynamic Memory - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
2. The process will not interpret data present on the dynamic memory as executable code.	Input to the process is not thoroughly checked, allowing for the possibility of such things as code or command injection into variables being stored on dynamic memory. When these variables are used, the data present on the dynamic memory will be interpreted as executable code.	Constraint/Assumption Not Met
4. The process will be provided with the dynamic memory that it requests.	Java implements an automatic memory management system. This system throws an error when the process is not able to be provided with memory that it requests. However, the process does not ensure that it has been provided with requested memory before proceeding with execution.	Constraint/Assumption Not Met
6. Data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.	Java implements an automatic memory management system. This includes automatic garbage collection. Garbage collection occurs when reference to a process object no longer exists. To ensure data cannot be accessed, the memory should be overwritten before being freed. There is no evidence of overwriting memory before it is eligible for garbage collection.	Constraint/Assumption Not Met
10. Data accepted by the process must not be interpreted as a format string by the I/O routines.	Methods used for input and output by the system do not allow the use of format strings	Constraint/Assumption Met
13. An integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values.	There do not exist any non-negativity checks for integers being used to indicate quantity within the source code.	Constraint/Assumption Not Met

Dynamic memory constraint/assumption 10 in Table 20 is met by the code review. For input to be interpreted as a format string, the system must output the input using a method that supports format strings, i.e. "printf(output)". The Horticulture Club Sales Assistant does not use any output methods which support format strings. Because the Horticulture Club Sales Assistant does not use methods supporting format strings, it is inherently protected against interpreting input as format strings and meets dynamic memory constraint/assumption 10.

Dynamic memory constraints/assumptions 4 and 6 are not met after code review. The Java programming language utilizes an automatic memory management system. Knowledge of the memory management system has been implied throughout the development artifacts. After code review, there is no sign that data being deleted is protected by writing junk values to the memory. Writing junk values to memory prevents data from being able to be read after it is deleted. The Horticulture Club Sales Assistant does not anticipate a scenario where Java's automatic memory management system will throw an exception because there is no available memory to allocate. Not handling this exception gracefully will cause the Horticulture Club Sales Assistant to crash.

An example of the source code not meeting a constraint/assumption can be shown using dynamic memory constraint/assumption 13 from Table 20. Code not meeting dynamic memory constraint/assumption 13 is highlighted in the code snippet below.

```
else
{
      CSalesItem salesItem = new CSalesItem();
      salesItem.setItemNumber(currItemNumber);
      String q = request.getParameter("quantity");

      if(q != null)
            salesItem.setQuantity(Integer.parseInt(q));
      else
            salesItem.setQuantity(1);

      salesItem.setUnitPrice(db.getPriceByID(salesItem.getItemNumber()));

      currSale.addItem(salesItem);
}
```

A simple boundary check within the code would make this portion of the source code meet dynamic memory constraint/assumption number 13 from Table 20. Shown below is the same portion of code, but the highlighted section represents additions that would make the original code meet the constraint/assumption.

```
else
{
      CSalesItem salesItem = new CSalesItem();
      salesItem.setItemNumber(currItemNumber);
      String q = request.getParameter("quantity");

      if(q != null && (Integer.parseInt(q) >= 0))
            salesItem.setQuantity(Integer.parseInt(q));
      else
            salesItem.setQuantity(1);

      salesItem.setUnitPrice(db.getPriceByID(salesItem.getItemNumber()));

      currSale.addItem(salesItem);
}
```

4.3.5.2 Static Memory

After review of the source code for the Horticulture Club Sales Assistant, there is no evidence of static memory being used. Static memory constraints/assumptions, Table 21, are deemed "N/A to Process" after code review.

Table 21. Static Memory Constraints/Assumptions from review of the Source Code.

Static Memory - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
1. Data accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to buffer on the static memory.	The process source code does not utilize static memory.	N/A to Process
2. Data held on the static memory cannot be observed while the process is in execution.	The process source code does not utilize static memory.	N/A to Process

4.3.5.3 Filesystem

Code review has an impact on several filesystem constraint/assumption related instances, Table 22. Filesystem constraint/assumption 4 in Table 22 is not met by the Horticulture Club Sales Assistant due to a lack of input validation. Uploaded files are not checked for their file type or file extension leading to the possibility that a user (that does not have execute privileges) can upload an executable file to the system. The user can then access the file via web browser and run the executable. Because file types of uploaded files are not checked, the Horticulture Club Sales Assistant also does not meet filesystem constraint/assumption 5 in Table 22.

Filesystem constraints/assumptions 8 and 9 are similar to dynamic memory constraints/assumptions 6 and 4 respectively. Filesystem constraint/assumption 8 is not met after code review because files used by the system are not overwritten with junk values to prevent them from being accessed after they are deleted. Filesystem constraint/assumption 9 is not met after code review due to a lack of exception handling by the Horticulture Club Sales Assistant. While a full system crash of the Horticulture Club Sales Assistant is unlikely, unhandled exceptions will crash a user’s current session and if filesystem space is not resolved, can lead to a potential denial of service of the Horticulture Club Sales Assistant.

Table 22. Filesystem Constraints/Assumptions from review of the Source Code.

Filesystem - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
4. A filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permissions.	There is no check on picture files being uploaded to the system. This can possibly lead to executable files being loaded and executed with root or administrative privileges.	Constraint/Assumption Not Met
5. A file created/populated by a principal other than the process and being used by the process will have expected format and data.	Image files being used for inventory purposes by the process are not checked for format. Files of any extension can be uploaded to the system and stored when only image files are expected. The uploading of files is protected via authorization.	Constraint/Assumption Not Met
8. Data held by files owned/used by the process must not be accessible after the process deletes them.	Data is deleted directly by the process, bypassing any trash or recycle folders. Data memory is then eligible to be overwritten. However, data deleted is not zeroed out by the process making recovery possible.	Constraint/Assumption Not Met
9. The process must be provided with the filesystem space that it requests.	There is no check to determine if memory is available within the filesystem before trying to write to it. However, the Java programming language will throw an exception if no memory is available and prevent a full system crash.	Constraint/Assumption Not Met

4.3.5.4 Network Interface

Table 23 shows new network interface constraint/assumption related instances introduced by the code review. All new related instances are for network interface constraint/assumption 4. The Horticulture Club Sales Assistant is a networked web application and must utilize the network interface to function properly. It is implied for network interface constraint/assumption 4 that the Horticulture Club Sales Assistant will have access to the network interface. Ensuring network access is external to the Horticulture Club Sales Assistant and the status of this related instance remains unchanged after code review.

Table 23. Network Interface Constraints/Assumptions from review of the Source Code.

Network Interface – Constraints/Assumptions	Related Instance	Constraint/Assumption Status
4. The software process will be able to utilize the network interface to send and receive data.	The software process is a networked web application running on a server implying that the network interface will be available.	Related Instance => Constraint/Assumption
	Database connections are not checked after they are requested in order to see if they exist.	Constraint/Assumption Not Met
	Database connections are not always closed which could lead to an exhaustion of network resources.	Constraint/Assumption Not Met

Source code for database connections in the Horticulture Club Sales Assistant introduces 2 new related instances for network interface constraint/assumption 4. Database connection handling in the source code is poor and does not meet network interface constraint/assumption 4. After requesting a database connection, code execution proceeds without checking to see if an actual database connection is returned. Assuming that a connection exists can lead to a system crash if a database connection request is not able to be fulfilled. Also, database connections are not always closed after they are opened. This can lead to an exhaustion of database connection resources and potentially denial of service.

4.3.5.5 Randomness Resources

There is no change in randomness resources constraints/assumptions between reviews of the integration and testing document and the source code. The HTTPS protocol is used by the Horticulture Club Sales assistant and utilizes random numbers. However, functionality of the HTTPS protocol is performed externally to the Horticulture Club Sales Assistant and is not found within the source code of the system.

4.3.5.6 Cryptographic Algorithms and Protocols

Cryptographic algorithms and protocols constraint/assumption 9, Table 24, is met by the Horticulture Club Sales Assistant. Passwords used for logging in to the private section of the Horticulture Club Sales Assistant are not stored. It is unavoidable that passwords must be stored in dynamic memory briefly during authentication. However, cryptographic algorithms and protocols constraint/assumption 9 is interpreted to describe the permanent storing of passwords in plaintext. Because the Horticulture Club Sales Assistant does not permanently store passwords, it meets cryptographic algorithms and protocols constraint/assumption 9.

Table 24. Cryptographic Algorithms and Protocols Constraints/Assumptions from review of the Source Code.

Cryptographic Algorithms and Protocols - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
9. The process cannot store keys/passwords in clear text.	Passwords are not stored by the software process.	Constraint/Assumption Met

4.4 Results

Application of Partial VV&T to the Horticulture Club Sales Assistant program was successful in identifying weaknesses within the system relating to the violable constraints/assumptions. Weaknesses within the system are potential software vulnerabilities that can be exploited by attackers. A benefit of Partial VV&T over other vulnerability identification methods is the ability to identify the origin of the vulnerability within the software development process.

The status of the many constraint/assumption related instances follow a similar trend throughout the review of the development artifacts. A software system becomes more refined through its development lifecycle giving greater insight into system functionality. Likewise, constraint/assumption related instances become more refined as Partial VV&T progresses through the development artifacts of a system. A majority of related instances of the Horticulture Club Sales Assistant are introduced with a status of “Related Instance =>

Constraint/Assumption." As system designs become more detailed through the development artifacts, the statuses of the related instances are able to be described as being met or not being met.

After applying Partial VV&T to the Horticulture Club Sales Assistant, it is clear that the software system is vulnerable to a number of possible attacks. The Partial VV&T review is successful in identifying constraint/assumption related instances that can lead to vulnerabilities. Partial VV&T can also aid in preventing these vulnerabilities. The review tables shown in Appendix A can identify the origins of a vulnerability associated with the constraints/assumptions. Being able to identify the origins of vulnerabilities within the software development lifecycle can aid software maintenance teams in eliminating or preventing software vulnerabilities. Constraints/assumptions not met by the Horticulture Club Sales Assistant after Partial VV&T review and associated vulnerabilities are as follows.

- **The process will not interpret data present on the dynamic memory as executable code.**

  Insufficient input validation of the Horticulture Club Sales Assistant presents the possibility that data placed on the dynamic memory will be interpreted as executable code. Potential vulnerabilities from not meeting this constraint/assumption include cross site scripting vulnerabilities and SQL injection attacks.

- **The process will be provided with the dynamic memory that it requests.**
  The Horticulture Club Sales Assistant does not gracefully handle the scenario of not being able to allocate memory. Exceptions thrown by Java's automatic memory management system can lead to a system crash if not handled. A system crash can potentially lead to a denial of service attack if settings leading to the crash are readily repeatable.

- **Data present on the dynamic memory cannot be observed while the process is in execution.**
  To prevent memory from being observed during execution, the Horticulture Club Sales Assistant must be prevented from running in a controlled environment where a hostile entity can potentially observe the memory during execution. The runtime environment for the Horticulture Club Sales Assistant is not described in development artifacts. Therefore it may be possible for a hostile entity to run and observe the Horticulture Club Sales Assistant in a controlled environment. Observing memory of the Horticulture Club Sales Assistant can provide a hostile entity with passwords and purchasing information.
- **Data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.**
  The Horticulture Club Sales Assistant does not write junk values to data variables before they are deleted to ensure that the data cannot be accessed after it is deleted. If a hostile entity is able to access the data after it is deleted, username and password combinations may be obtainable. Username password combinations can give an attacker full access to the Horticulture Club Sales Assistant.
- **An integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values.**
  Negative values are able to be input to the Horticulture Club Sales Assistant to represent quantity. Negative quantities can lead to errors in the purchasing system, potentially giving refunds to customers instead of charging them.
- **A filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permissions.**
  Uploaded files to the Horticulture Club Sales Assistant are not checked for their file type. An executable file can be loaded by a middle level user of the system and executed to elevate their privileges.

- **A file created/populated by a principal other than the process and being used by the process will have expected format and data.**
  The ability to upload files of any format can lead to several vulnerabilities within the system. Denial of service can be performed by shutting down the web server. Databases containing sales information can be printed out to users whom do not have sales report privilege.
- **Data held by files owned/used by the process must not be accessible after the process deletes them.**
  The Horticulture Club Sales Assistant does not ensure that files are unable to be accessed after they are deleted. Files containing sales information can potentially be accessed after they are deleted allowing a hostile entity to read sensitive information.
- **The process must be provided with the filesystem space that it requests.**
  The Horticulture Club Sales Assistant does not gracefully handle a scenario where it is not provided with the filesystem space that it requests. Exceptions thrown in the Java programming language if filesystem space is unavailable can crash the Horticulture Club Sales Assistant and lead to a denial of service attack.
- **The data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.**
  Formats and lengths of data received through the network interface are not checked before they are used by the Horticulture Club Sales Assistant. Lack of input validation can lead to cross site scripting vulnerabilities, SQL injection, and denial of service attacks.
- **The software process will be able to utilize the network interface to send and receive data.**
  Database connections are not appropriately handled by the Horticulture Club Sales Assistant. Inappropriately handled connections can lead to a waste of resources and system crash resulting in denial of service.

4.5 Potential Improvements to Partial VV&T Process

Potential improvements for applying Partial VV&T to a software system include using a different status nomenclature, additions to the review table, and introducing software specific constraints/assumptions. The nomenclature defining the status of constraint/assumption related instances is developed with the Horticulture Club Sales Assistant program in mind. This nomenclature may not suit the application of Partial VV&T to every software system. Also, instead of using words to indicate a system's status, colors may provide a stronger sense of urgency within the review tables. For example, a status of the color red might call more attention than a status of "Constraint/Assumption Not Met."

Adding columns to the review table might aid in the effectiveness of Partial VV&T application. Potential columns include page number and specific instance. In the review tables used by this research there is no indication of what page the related instance noted in the review table occurs on. Having the page number of the related instance might help software maintenance teams in finding a quicker reference to the origins of a software vulnerability. A specific instance column in the review table would hold similar contents as the related instance column that already exists. However, the specific instance column would include the exact wording or code instance encountered within the development artifact. In this research the related instance column is used in describing the instance and how it is applicable to the constraint/assumption. The specific instance column coupled with the related instance column and page number column would allow for quicker recognition of the origin of potential software vulnerabilities.

Another potential improvement to the application Partial VV&T is the addition of software system specific constraints/assumptions being added to the review table. These specific constraints/assumptions can essentially be security concerns expected to be addressed by the system. Software specific constraints/assumptions can be added by developers or reviewers that are knowledgeable of the system. Adding constraints/assumptions specific to the software system can ensure that security policies of the system are met along with original constraint/assumption related instances.

5. End-Game VV&T

Chapter 5 describes the application of End-Game VV&T to the Horticulture Club Sales Assistant. Section 5.1 highlights differences between End-Game VV&T and Partial VV&T. Section 5.2 outlines testing strategies used in applying End-Game VV&T. Section 5.3 describes the application of End-Game VV&T to the Horticulture Club Sales Assistant. Section 5.4 details results of applying End-Game VV&T to the Horticulture Club Sales Assistant. Section 5.5 discusses potential improvements to the End-Game VV&T application process.

5.1 Differences between Partial VV&T and End-Game VV&T

The major difference between Partial and End-Game VV&T is that End-Game VV&T is performed when only the executable binary of a system is available for review. Partial VV&T is used when other development artifacts are available for review along with the executable binary. Those additional artifacts can be used to guide an examination of the executable. Application of End-Game VV&T relies only on base test strategies that are developed to test for the presence of constraints/assumptions within the executable binary. Partial VV&T uses information gained from previous development artifacts to identify areas of concern within the executable binary and minimizes the need for test-based strategies.

For the Horticulture Club Sales Assistant, there is a special case to observe when applying End-Game VV&T. The executable binary of this system is compiled initially at runtime and then continuously thereafter if files are modified during runtime. No single executable file for the software system exists. Therefore, having access to the executable binary of the Horticulture Club Sales Assistant implies having access to the source code. Compiling source code at runtime allows web applications such as the Horticulture Club Sales Assistant to perform system maintenance without shutting down the system.

Having access to source code allows the code to be used in refining base test cases to target the executable binary. Likewise, the source code can be used as a reference during testing. Access to source code is not specified in the description of End-Game VV&T. However, neither are software systems that are compiled at runtime. In general, having source code during the application of End-Game VV&T is not required, but can only enhance the overall application of End-Game VV&T to the executable binary. Although access to source code is not included in the description of End-Game VV&T, source code of the Horticulture Club Sales Assistant is used in conjunction with the executable binary for this application of End-Game VV&T.

5.2 Testing Strategies

Test strategies are used in End-Game VV&T to test the software system for violations of constraints/assumptions. Each test strategy has a goal, a target, and a method. The goal of each test strategy is to test if a constraint/assumption has been violated. Targets of test strategies are objects or resources used by the software system. Finally, the method of a test strategy describes how to test the target in order to reach the goal. Base test strategies have been provided for the constraints/assumptions and are generic so that they can be applied, in general, to any software system. However, refining the test strategies to be software system specific permits more efficient testing. Source code for the Horticulture Club Sales Assistant program is used in refining and eliminating base strategies used in End-Game VV&T. Refined test strategies are adapted from the base test strategies and vary only in their target and goal.

5.2.1 Dynamic Memory

Navigating through the web pages of the Horticulture Club Sales Assistant, the use of dynamic memory within the software process is evident. The Horticulture Club Sales Assistant permits dynamic input and output, both of which require dynamic memory. From the system executable and referencing available source code, test strategies are refined that target dynamic memory aspects of the software system. Listed below are refined test strategies that test for dynamic memory constraints/assumptions in the Horticulture Club Sales Assistant.

Test Strategy #1

Goal: To test if the software process allows violation of the assumption: *the process will not interpret data present on the dynamic memory as executable code.*

Target: Inputs expecting strings.

Method: Attempt to pass partial or whole SQL commands as input to the system. The assumption is considered violated if the process accepts and executes the input.

Test Strategy #2

Goal: To test if the software process allows violation of the assumption: *the process will be provided with the dynamic memory that it requests*

Target: The memory resource being used by the process.

Method: Restrict the amount of memory available to the software process by running it in a controlled environment. The assumption is violated if the process terminates or freezes.

Test Strategy #3

Goal: To test if the software process allows violation of the assumption: *data present on the dynamic memory cannot be observed while the process is in execution.*

Target: Memory Heap.

Method: Attempt to read the contents of the dynamic memory while the process is in execution. The assumption is violated if privileged data can be read from the dynamic memory.

Test Strategy #4

Goal: To test if the software process allows violation of the assumption: *data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.*

Target: Memory Heap.

Method: Attempt to read the contents of memory freed by the process after instantiating garbage collection. The assumption is violated if the memory contains information from the software process and is able to be read.

Test Strategy #5

Goal: To test if the software process allows violation of the constraint: *data accepted by the process must not be interpreted as a format string by the I/O routines.*

Target: String inputs to the process.

Method: Provide the process with a format string as input. The constraint is violated if the process accepts the format string and outputs contents of the program stack.

Test Strategy #6

Goal: To test if the software process allows violation of the assumption: *the value of an integer/expression (signed & unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer variable.*

Target: Inputs of the process using integer values.

Method: Provide the process with values exceeding the maximum (minimum) values of integer variables as input. The assumption is violated if the process stores and uses these values.

Test Strategy #7

Goal: To test if the software process allows violation of the constraint: *an integer variable/expression used by the process to indicate length/quantity of an object must not hold negative values.*

Target: Inputs of the process indicating length/quantity.

Method: Provide the process with negative values as input. The constraint is violated if the process accepts and uses these values.

Dynamic memory test strategies used in End-Game VV&T do not represent all dynamic memory constraints/assumptions. Several dynamic memory constraints/assumptions are not tested for during the application of End-Game VV&T due to information gathered from the system executable. More specifically, it is known from both the system executable and the source code that the Horticulture Club Sales Assistant is programmed using the Java programming language. The Java programming language has an automatic boundary checking property. Automatic checking by Java of string objects being copied as input to the system prevents the possibility of buffer overflow. Also, the automatic boundary checking prevents an index of a buffer from being greater than the size of a buffer. It is not necessary to refine test strategies for the following constraints/assumptions because they are immediately met in the Horticulture Club Sales Assistant by the automatic boundary checking property of the Java programming language.

- *Data accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to the buffer*.
- *An integer variable/expression used by the process as the index to a buffer must only hold values that allow it access to the memory locations assigned to the buffer*.

Pointer variables can be used to reference invalid, illegal, and privileged data that can cause a software system to crash. However, the Java programming language does not use pointer variables. Because Java does not use pointers, it is not necessary to refine test strategies for the following constraints/assumptions as they are not applicable to the Horticulture Club Sales Assistant:

- *A pointer variable being used by the process references a legal memory location*.
- *A memory pointer returned by the underlying operating system does not point to zero bytes of memory*.
- *A pointer variable being used by the process cannot reference itself*.

Based on the source code, there is no evidence that the Horticulture Club Sales Assistant directly accesses environment variables. Because the software process does not directly access environment variables, it is not necessary to refine test strategies for the following constraint/assumption as it is not applicable to the Horticulture Club Sales Assistant:

- *Environment variables being used by the process have expected format and values.*

5.2.2 Static Memory

Static Memory constraints/assumptions are not tested during the application of End-Game VV&T to the Horticulture Club Sales Assistant. The system executable, along with the source code, does not indicate that the Horticulture Club Sales Assistant uses static memory. Because static memory is not used by the Horticulture Club Sales Assistant, it is not necessary to refine test strategies for the following constraints/assumptions as they are not applicable to the Horticulture Club Sales Assistant:

- *Data accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to the buffer on the static memory*.
- *Data held on the static memory cannot be observed while the process is in execution*.

5.2.3 Filesystem

The Horticulture Club Sales Assistant is capable of uploading picture files, generating sales reports, and has an inventory management system. The filesystem must be tested to ensure that file permissions are present and respected, and that files contained in the filesystem are of the expected format and length. Files stored on the filesystem are critical to correct functionality of the Horticulture Club Sales Assistant. Listed below are refined test strategies that test for filesystem constraints/assumptions in the Horticulture Club Sales Assistant.

Test Strategy #8

Goal: To test if the software process allows violation of the constraint: *access permissions assigned to newly created files/directories are such that only the required principals have access to them.*

Target: Newly created files.

Method: Examine the access permissions of newly created files. The constraint is violated if any principal other than the required principals has access to the files.

Test Strategy #9

Goal: To test if the software process allows violation of the constraint: *access permissions of the files/directories being used by the process are such that only the required principals have access to them.*

Target: Used files/directories.

Method: Examine the access permissions of files/directories used by the process. The constraint is violated if any principal other than the required principals has access to the files/directories.

Test Strategy #10

Goal: To test if the software process allows violation of the assumption: *a file being created by the process does not have the same name as an already existing file.*

Target: Newly created files.

Method: Anticipate the name of a future file created by the process. Create a file with the same name and place in the location of the future file. The assumption is violated if the process uses the file not created by the process or terminates suddenly.

Test Strategy #11

Goal: To test if the software process allows violation of the assumption: *a filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permission.*

Target: Files created by the process.

Method: Provide the process with a file containing executable code. The assumption is violated if the process executes code within the file that the user would otherwise not have the permissions to execute.

Test Strategy #12 **Goal:** To test if the software process allows violation of the assumption: *a file created/populated by a principal other than the process and being used by the process will have expected format and data.* **Target:** Files uploaded to the process. **Method:** Provide the process files with unexpected format or data. The assumption is violated if the process accepts and uses these files.
Test Strategy #13 **Goal:** To test if the software process allows violation of the constraint: *data held by files owned/used by the process must not be accessible after the process deletes them.* **Target:** Files deleted by the process. **Method:** Attempt to access data held in files after the process deletes the files. The constraint is violated if the data is able to be accessed and read.
Test Strategy #14 **Goal:** To test if the software process allows violation of the constraint: *the process must be provided with the filesystem space that it requests.* **Target:** Filesystem size. **Method:** Attempt to restrict the amount of filesystem space available to the process and exceed this limit by loading files too big to store on the reduced space. The constraint is violated if the process freezes or terminates abruptly.

The Horticulture Club Sales Assistant is a dynamic web application that is compiled at runtime. File modification can be performed while the system is executing and does not affect the execution of the process. Also, files used by the Horticulture Club Sales Assistant can be modified between runs of the process without affecting system functionality. Because the Horticulture Club Sales Assistant is a dynamic web application, it is not necessary to refine test strategies for the following constraints/assumptions as they are not applicable to the Horticulture Club Sales Assistant:

- *A file being used by the process cannot be observed/modified/replaced while the process is in execution.*
- *A file/directory being used by the process and stored on the filesystem (information used by the process over multiple runs) cannot be observed/modified/replaced in-between these runs.*

Referencing the source code, there is no evidence that files having obscure or proprietary format are used by the Horticulture Club Sales Assistant. Because files of obscure or proprietary format are not used, it is not necessary to refine test strategies for the following constraint/assumption as it is not applicable to the Horticulture Club Sales Assistant:

- *A file having proprietary or obscure format cannot be understood or modified.*

5.2.4 Network Interface

Because the Horticulture Club Sales Assistant is a dynamic web application, the system executable must both receive and send data using the network interface. Data received by the Horticulture Club Sales Assistant should be validated before being used. Purchases can be made using the Horticulture Club Sales Assistant, implying the transfer of sensitive data. Listed below are refined test strategies that test for network interface constraints/assumptions in the Horticulture Club Sales Assistant.

Test Strategy #15

Goal: To test if the software process allows violation of the assumption: *the data received by the software process through the network interface is neither read nor modified by anyone other than the intended recipient.*

Target: Data received by the software process.

Method: Intercept data being sent to the software process and attempt to read or modify it. The assumption is violated if the data is able to be read or the process uses the modified data.

Test Strategy #16

Goal: To test if the software process allows violation of the assumption: *the data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.*

Target: Data sent to the software process.

Method: Send data of unexpected format and length to the process. The assumption is violated if the process accepts and uses this data.

Test Strategy #17

Goal: To test if the software process allows violation of the assumption: *the data sent by the software process via the network interface will not be read/modified before it reaches its destination.*

Target: Data sent by the software process.

Method: Intercept data sent by the software process and attempt to read or modify it. The assumption is violated if the data is able to be read or the receiving process uses the modified data.

Test Strategy #18

Goal: To test if the software process allows violation of the assumption: *the software process will be able to utilize the network interface to send and receive data.*

Target: Network connections of the software process.

Method: Limit the availability of the network interface to the software process. The assumption is violated if the process freezes or terminates abruptly.

The Java programming language uses network byte order in representing data. Because network byte order is used by the Java programming language, it is not necessary to refine test strategies for the following constraint/assumption as it is immediately met by the Horticulture Club Sales Assistant:

- *The byte order of numerical data accepted from the network interface is same as that of the host machine*.

5.2.5 Randomness Resources

Usernames and passwords are used by the Horticulture Club Sales Assistant to grant access to the private extranet of the system. Passwords used to gain access to the Horticulture Club Sales Assistant should not be easily guessed or cracked. Listed below is a refined test strategy to find instances of the randomness resources constraint/assumption describing password strength.

Test Strategy #19

Goal: To test if the software process allows violation of the assumption: *user selected passwords/keys will have a sufficient amount of entropy.*

Target: User selected passwords.

Method: Identify the origins of the users selected passwords used by the software process. The assumption is violated if the passwords were not created under requirements that would enforce increased entropy.

Referencing the source code, pseudo random number generators (PRNGs) are not directly used by the Horticulture Club Sales Assistant. Because the Horticulture Club Sales Assistant does not directly use PRNGs, it is not necessary to refine test strategies for the following constraints/assumptions as they are not applicable to the Horticulture Club Sales Assistant:

- *The series of random data being produced by the PRNG is unpredictable (assume that seed is unpredictable).*
- *The seed being used by the PRNG is unpredictable.*
- *If two different seeds are provided to the PRNG, it is computationally infeasible to produce the same series of data both times.*
- *Given that the PRNG is continuously producing random data, it is computationally infeasible to produce the same sequence of random data after some time.*

Entropic data is neither accessed nor estimated by the source code of the Horticulture Club Sales Assistant. Because entropic data is neither accessed nor estimated, it is not necessary to refine test strategies for the following constraints/assumptions as they are not applicable to the Horticulture Club Sales Assistant:

- *The process will have easy access to entropic data on a computer system.*
- *The process will be able to accurately estimate the entropy of a data set.*

5.2.6 Cryptographic Algorithms and Protocols

Sensitive data is sent and received across the network by the Horticulture Club Sales Assistant. Transfer of sensitive data over the network implies that data integrity as well as data confidentiality will be ensured. Passwords should not be stored in plain text where they can easily be read and compromised. Listed below are refined test strategies used to find instances of cryptographic algorithms and protocols constraints/assumptions within the Horticulture Club Sales Assistant.

Test Strategy #20

Goal: To test if the software process allows violation of the constraint: *the process cannot use encryption to ensure data integrity.*

Target: Securely transmitted data.

Method: Determine if the securely transmitted data is using encryption only in order to ensure data integrity. The constraint is violated if only encryption is being used to ensure data integrity.

Test Strategy #21

Goal: To test if the software process allows violation of the constraint: *the process cannot use obfuscation instead of encryption to ensure confidentiality.*

Target: Data used by the process.

Method: Analyze data used by the software process and determine to what extents obfuscation is used to keep secrets. The constraint is violated if obfuscation is used to keep secrets.

Test Strategy #22

Goal: To test if the software process allows violation of the constraint: *the process cannot store keys/passwords in clear text.*

Target: Data store.

Method: Analyze the data store to determine if passwords are stored in clear text. The constraint is violated if the software process stores passwords in clear text.

The HTTPS protocol is used by the Horticulture Club Sales Assistant. While the HTTPS protocol is used by the process, the implementation of the HTTPS protocol is handled by the web server and web browsers. Therefore, the process itself is not directly responsible for the negotiation of the TLS connection used by the HTTPS protocol. The HTTPS protocol is the only cryptographic algorithm or protocol used by the Horticulture Club Sales Assistant. Because the software process is not responsible for establishing the TLS connection used by HTTPS, it is not necessary to refine test strategies for the following constraints/assumptions as they are not applicable to the Horticulture Club Sales Assistant:

- *Random data being used by the cryptographic algorithm/protocol is unpredictable.*
- *The length of the key being used by the cryptographic algorithm or protocol is sufficient.*
- *The hashing algorithm will not produce same hash for two different inputs.*
- *The process cannot use a key more than once for a stream cipher.*
- *The process cannot use one time pads to encrypt a large quantity of data.*
- *The process cannot use keys that are self reported by a client or a server.*

5.3 Application of End-Game VV&T

To apply End-Game VV&T, the refined test strategies are applied to the Horticulture Club Sales Assistant. The goal of each test strategy is to test for the violation of a specific constraint/assumption. Violating a constraint/assumption may indicate a vulnerability within the system. Testing of the Horticulture Club Sales Assistant is performed using a test server. Functionality of the Horticulture Club Sales Assistant is compared against functionality described in development artifacts to proper interaction by the system.

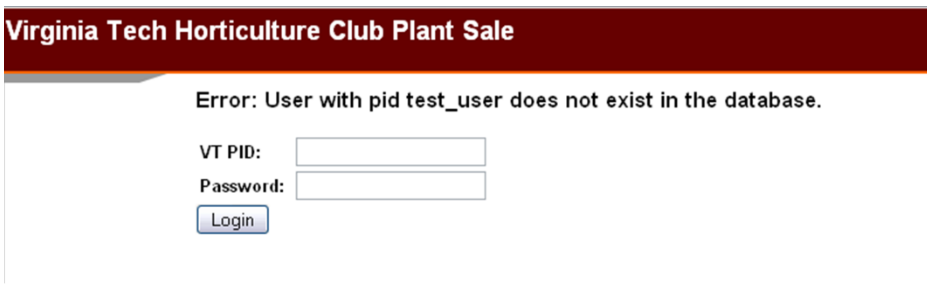
5.3.1 Test Strategy #1

Goal: To test if the software process allows violation of the assumption: *the process will not interpret data present on the dynamic memory as executable code.*

Target: Inputs expecting strings.

Method: Attempt to pass partial or whole SQL commands as input to the system. The assumption is considered violated if the process accepts and executes the input.

To apply this test strategy, focus is placed on the login page of the Horticulture Club Sales Assistant. The login page expects a username and a password as input to the process in order to authenticate a user. Usernames and passwords are authenticated together through an external means, but the user is authenticated to the software process by simply checking the username against a database of users. To check the database of users, the username is added to an SQL statement that queries the database. For reference, the resulting web page displayed after a login attempt from an invalid user is shown in Figure 13. In Figure 13, a username of *test_user* is used in generating the error page.

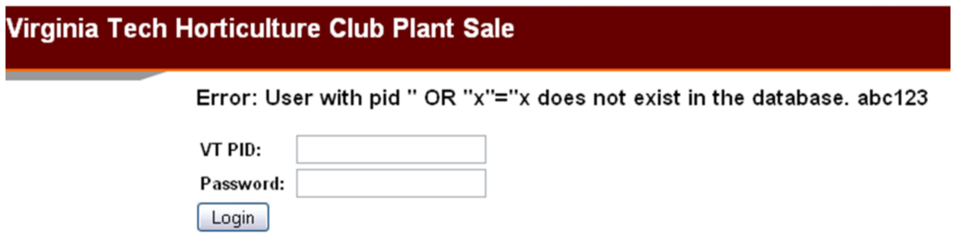


Virginia Tech Horticulture Club Plant Sale

Error: User with pid test_user does not exist in the database.

Figure 13. Invalid user login screen.

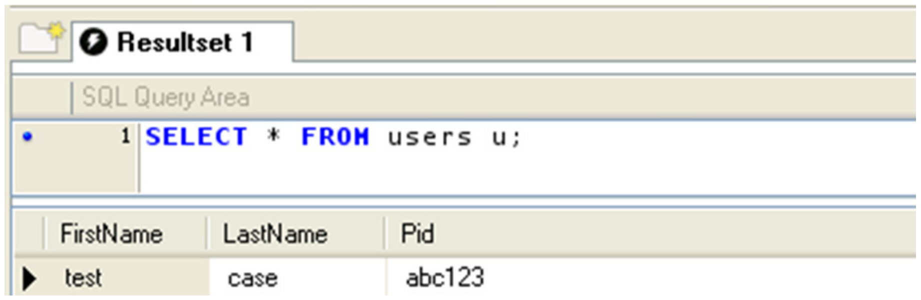
To attempt to place executable code on the dynamic memory, a username of *" OR "x"="x* is entered into the "VT PID" input option on the login screen and submitted for authorization. The resulting error page is shown in Figure 14. Providing the software executable with this input prompts the SQL query to return a valid user of the system, user "abc123". Username "abc123" is the first user name that appears in the table of users for the Horticulture Club Sales Assistant. Using the MySQL Query Browser [61], username "abc123" does exist in the table of users (Figure 15).



Virginia Tech Horticulture Club Plant Sale

Error: User with pid " OR "x"="x does not exist in the database. abc123

Figure 14. Invalid user login screen with SQL injection.



```
SELECT * FROM users u;
```

```
FirstName  LastName  Pid
test       case      abc123
```

Figure 15. Query browser showing users of the system executable.

The process accepts a partial SQL command as input, placing it in an object being held in dynamic memory. This object is then used in constructing a SQL query that is interpreted as executable SQL syntax. Because the Horticulture Club Sales Assistant interprets data present on the dynamic memory as executable code, the Horticulture Club Sales Assistant violates the assumption: *the process will not interpret data present on the dynamic memory as executable code*. Returning a valid user of the software system can be very helpful to attackers. Instead of having to crack, or guess, a username-password pair, half of an attackers work is already done. Knowing a valid username allows an attacker to focus all of their resources on finding a valid password to go along with the valid username.

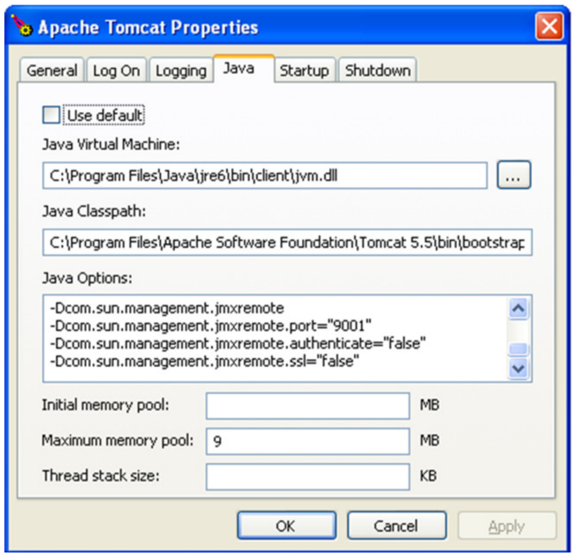
5.3.2 Test Strategy #2

Goal: To test if the software process allows violation of the assumption: *the process will be provided with the dynamic memory that it requests*

Target: The memory resource being used by the process.

Method: Restrict the amount of memory available to the software process by running it in a controlled environment. The assumption is violated if the process terminates or freezes.

To implement this test strategy, the amount of memory available to the Java Virtual Machine (JVM) is restricted. The JVM is invoked when the Tomcat web server is started. When invoking the JVM, Tomcat has the opportunity to pass startup parameters to the JVM. One of the parameters passed is the maximum amount of heap size available. Limiting the heap size of the JVM effectively limits the heap size of the software executable as the software executable is executed inside the JVM and can only access memory available to the JVM. The maximum heap size is set to 9MB as shown in Figure 16.



Use default

Java Virtual Machine:

C:\Program Files\Java\jre6\bin\client\jvm.dll

Java Classpath:

C:\Program Files\Apache Software Foundation\Tomcat 5.5\bin\bootstrap

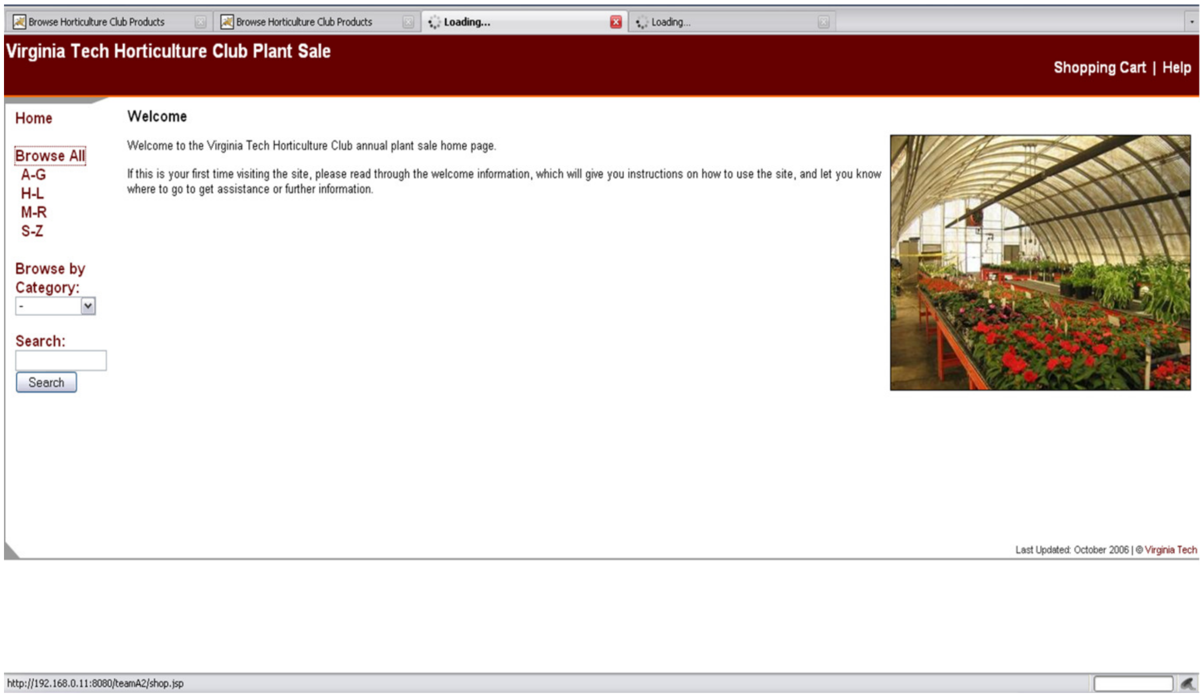
Java Options:

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port="9001"
-Dcom.sun.management.jmxremote.authenticate="false"
-Dcom.sun.management.jmxremote.ssl="false"
```

Initial memory pool:		MB
Maximum memory pool:	9	MB
Thread stack size:		KB

Figure 16. Memory settings for the JVM.

A maximum memory size of 9MB is determined from running Jconsole [62] in conjunction with the Horticulture Club Sales Assistant. Jconsole shows the heap usage of the Horticulture Club Sales Assistant running under normal conditions. A maximum of 9MB can easily be exceeded by opening several web pages at one time. To use as much dynamic memory as possible, web pages are directed to the “Browse All” inventory page of the Horticulture Club Sales Assistant. This requires the all inventory objects to be loaded into dynamic memory in order to display them. Accessing the “Browse All” web page is successful twice before freezing on the third try, Figure 17.



Virginia Tech Horticulture Club Plant Sale

Welcome

Welcome to the Virginia Tech Horticulture Club annual plant sale home page.

If this is your first time visiting the site, please read through the welcome information, which will give you instructions on how to use the site, and let you know where to go to get assistance or further information.

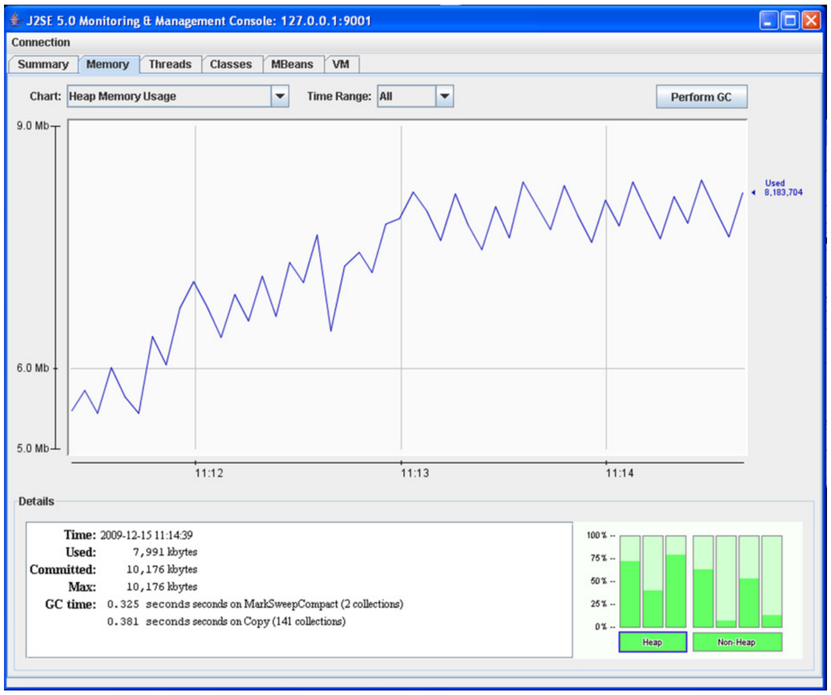
Last Updated: October 2006 | © Virginia Tech

http://192.168.0.11:8080/teamA2/shop.jsp

Figure 17. Frozen web page due to lack of memory resources.

Figure 17 shows the webpage in its frozen state while trying to access the “Browse All” web page of the Horticulture Club Sales Assistant. The first two tabs in the web browser in Figure 17 show that the “Browse All” web page has been accessed twice by the browser. The third tab shows that the “Browse All” web page is selected; however, the Horticulture Club Sales Assistant is unable to generate and send the requested data. The fourth tab in Figure 17 is an attempt to access the web browser’s home page after an attempt to access the “Browse All” web page has frozen. The home page of the web browser is unable to be accessed.

Data provided by Jconsole, Figure 18, shows that the memory usage never exceeds or nears the maximum of 9MB. Instead, access to the Horticulture Club Sales Assistant freezes with around 8MB of JVM memory being utilized. Two distinct jumps can be seen in memory usage that are fairly similar to each other, Figure 18. These jumps correspond to the two “Browse All” web page accesses. Following the trend of these memory jumps, another memory jump of this magnitude will exceed the 9MB memory threshold and is the reason why the Horticulture Club Sales Assistant froze when trying to access the “Browse All” web page.



```
     Time: 2009-12-15 11:14:39
     Used:      7,991 kbytes
Committed:     10,176 kbytes
      Max:     10,176 kbytes
 GC time:  0.325  seconds seconds on MarkSweepCompact (2 collections)
           0.381  seconds seconds on Copy (141 collections)
```

Figure 18. Jconsole display showing memory usage for Test Strategy #2.

Because the Horticulture Club Sales Assistant freezes when dynamic memory is not available, the Horticulture Club Sales Assistant violates the assumption: *the process will be provided with the dynamic memory that it request*. Violating this assumption makes the Horticulture Club Sales Assistant vulnerable to such attacks as denial of service. To perform denial of service on the Horticulture Club Sales Assistant an attacker can exhaust memory resources by opening memory intensive web pages of the Horticulture Club Sales Assistant.

To prevent against denial of service attacks, cleanup of unused resources should occur even if the system web page is still being viewed by a user. For example, after the inventory is requested from the database and displayed to the user in the “Browse All” memory page, the inventory should be removed from dynamic memory, allowing it to be allocated for other operations. “Out of memory” errors should be gracefully handled. Gracefully handling “out of memory” errors will allow the Horticulture Club Sales Assistant to execute upon newly received requests when dynamic memory becomes available instead of freezing or terminating.

5.3.3 Test Strategy #3

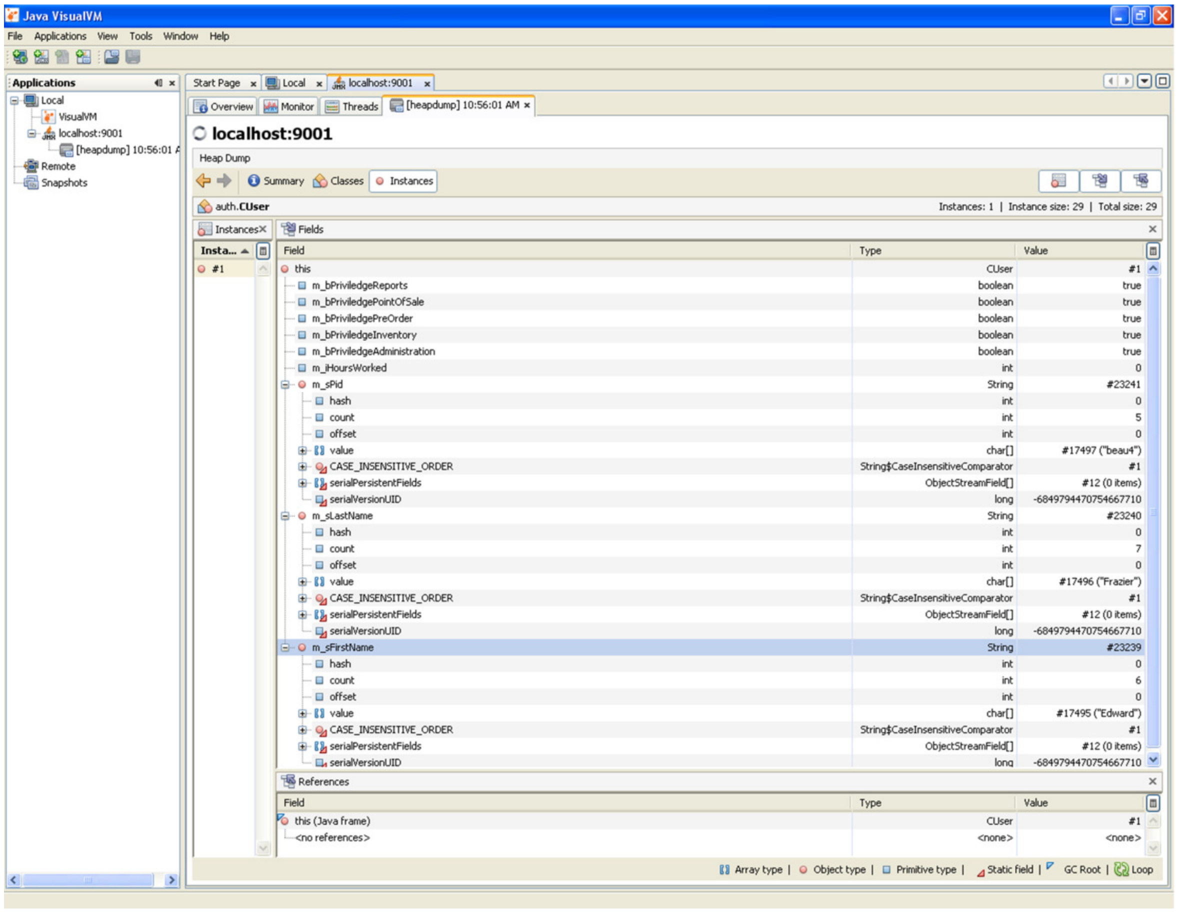
Goal: To test if the software process allows violation of the assumption: *data present on the dynamic memory cannot be observed while the process is in execution.*

Target: Memory Heap.

Method: Attempt to read the contents of the dynamic memory while the process is in execution. The assumption is violated if privileged data can be read from the dynamic memory.

To try and observe dynamic memory of the Horticulture Club Sales Assistant while the program is in execution, a heap dump is performed for the JVM. The heap dump is performed using Java VisualVM [63]; a program that enables monitoring of memory management for the JVM. Before a heap dump can be performed, the Horticulture Club Sales Assistant must access and store data on the dynamic memory. A user is authenticated to the system to place user specific data, such as permissions, onto the dynamic memory.

Figure 19 shows the heap dump performed after a user is authenticated to the Horticulture Club Sales Assistant. The heap dump lists instances of objects held in dynamic memory, most of which are objects of the Horticulture Club Sales Assistant. An instance of "auth.CUser" provides information about the authorized user. Viewable properties of the "auth.CUser" object include the user's username as well as their first and last name. The "auth.CUser" object also contains permissions of the user including whether or not the user has administrative privileges within the Horticulture Club Sales Assistant.



auth.CUser — Instances: 1 | Instance size: 29 | Total size: 29

Insta...
#1

Field	Type	Value
this	CUser	#1
m_bPriviledgeReports	boolean	true
m_bPriviledgePointOfSale	boolean	true
m_bPriviledgePreOrder	boolean	true
m_bPriviledgeInventory	boolean	true
m_bPriviledgeAdministration	boolean	true
m_iHoursWorked	int	0
m_sPid	String	#23241
hash	int	0
count	int	5
offset	int	0
value	char[]	#17497 ("beau4")
CASE_INSENSITIVE_ORDER	String$CaseInsensitiveComparator	#1
serialPersistentFields	ObjectStreamField[]	#12 (0 items)
serialVersionUID	long	-6849794470754667710
m_sLastName	String	#23240
hash	int	0
count	int	7
offset	int	0
value	char[]	#17496 ("Frazier")
CASE_INSENSITIVE_ORDER	String$CaseInsensitiveComparator	#1
serialPersistentFields	ObjectStreamField[]	#12 (0 items)
serialVersionUID	long	-6849794470754667710
m_sFirstName	String	#23239
hash	int	0
count	int	6
offset	int	0
value	char[]	#17495 ("Edward")
CASE_INSENSITIVE_ORDER	String$CaseInsensitiveComparator	#1
serialPersistentFields	ObjectStreamField[]	#12 (0 items)
serialVersionUID	long	-6849794470754667710

Field	Type	Value
this (Java frame)	CUser	#1
<no references>	<none>	<none>

Figure 19. Heap dump after logging into the system executable.

Because this data can be observed while the Horticulture Club Sales Assistant is in execution, the Horticulture Club Sales Assistant violates the assumption: *data present on the dynamic memory cannot be observed while the process is in execution.* If a hostile entity is able to gain access to the computer system on which the Horticulture Club Sales Assistant is running and perform a heap dump, sensitive data of the system may be compromised. Data stored on the dynamic memory can include usernames, passwords, full names and addresses of a user.

5.3.4 Test Strategy #4

Goal: To test if the software process allows violation of the assumption: *data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.*

Target: Memory Heap.

Method: Attempt to read the contents of memory freed by the process after instantiating garbage collection. The assumption is violated if the memory contains information from the software process and is able to be read.

Java VisualVM is also used during this test to try and read data stored on dynamic memory. In addition to being able to generate a heap dump, Java VisualVM can force the automatic garbage collection of the JVM to occur. Forcing automatic garbage collection allows garbage collection to occur by bypassing the automatic collection algorithm. It is necessary to force garbage collection for this test because of how memory is freed by Java processes. In Java, memory is not freed by a process directly. Instead, when an object is no longer referenced by a software process, the object becomes eligible for garbage collection. Only after garbage collection has been performed, has memory that was allocated to the software process been freed.

This test extends on the test performed in test strategy #3. Figure 19 shows data available in dynamic memory after authenticating a user to the Horticulture Club Sales Assistant. After authenticating the user, the user is “logged out” of the system to remove any references to the “auth.Cuser” object described in test strategy #3. With no remaining references, the garbage collection is invoked to free the dynamic memory from the Horticulture Club Sales Assistant.

A heap dump is performed after invoking garbage collection and shows that the instance of the “auth.CUser” object shown in Figure 19 no longer exists in dynamic memory. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the assumption: *data owned by the process and stored in the dynamic* *memory cannot be accessed after the process frees the memory*. The Horticulture Club Sales Assistant meeting this assumption provides protection from a hostile entity accessing sensitive information stored on dynamic memory after it has been deleted by the Horticulture Club Sales Assistant.

5.3.5 Test Strategy #5

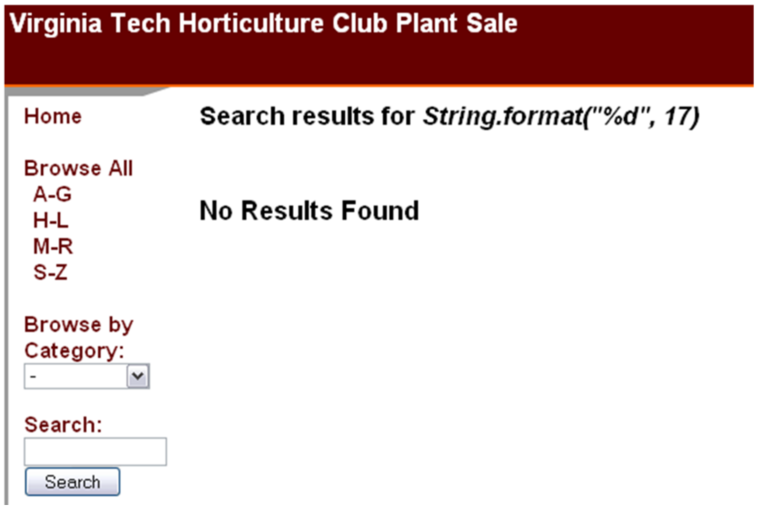
Goal: To test if the software process allows violation of the constraint: *data accepted by the process must not be interpreted as a format string by the I/O routines.*

Target: String inputs to the process.

Method: Provide the process with a format string as input. The constraint is violated if the process accepts the format string and outputs contents of the program stack.

Inputs of the Horticulture Club Sales Assistant expecting string inputs are targeted for this test. More specifically, inputs that are redisplayed to the client's web browser are targeted for this test. A textbox used to implement searching of inventory stored by the Horticulture Club Sales Assistant redisplays the text being searched for on the results page. The search text is displayed on the results page whether a match to the search query is found or not.

One way to use format strings within Java is to invoke the String.format() method [64]. This method returns a formatted string specified by arguments being passed to it. Testing to see if the software process accepts and uses this type of format string, a simple instruction of *String.format("%d", 17)* is given to the search input. Display of the number "17" is expected on the search results webpage if the software process accepts and interprets the input as a format string. However, the number "17" is not displayed and output displayed to the web browser is shown in Figure 20.


Virginia Tech Horticulture Club Plant Sale

Search results for *String.format("%d", 17)*

No Results Found

Figure 20. Format string test results using "String.format()".

Results from Figure 20 show that the software process does not interpret the input as a format string. Other variations of this type of formatted string input are attempted. Variations of the formatted string input include adding semicolons so the input will be interpreted as a command and adding escape characters to try and force the software process to interpret the input as a single command. For each attempt, the search results web page displays exactly what is entered in the search box and the process does not interpret the input as a format string. An additional test input of *"%d", 17* is also submitted to the software process. This test input tests for use of the printf() output method. Test input *"%d", 17* is not interpreted as a format string by the Horticulture Club Sales Assistant.

Referencing the source code, the Horticulture Club Sales Assistant is protected against format string attacks due to how the process handles inputs before redisplaying them to the web browser. When the process receives inputs from the user, they are retrieved using the getParameter() method. This method retrieves inputs as string objects and does not interpret the input as a formatted string[65]. After retrieving the data, the data is injected into an output string object. Combining data in this way does not allow for the format string syntax to be interpreted. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the constraint: *data accepted by the process must not be interpreted as a format string by the I/O routines.*

5.3.6 Test Strategy #6

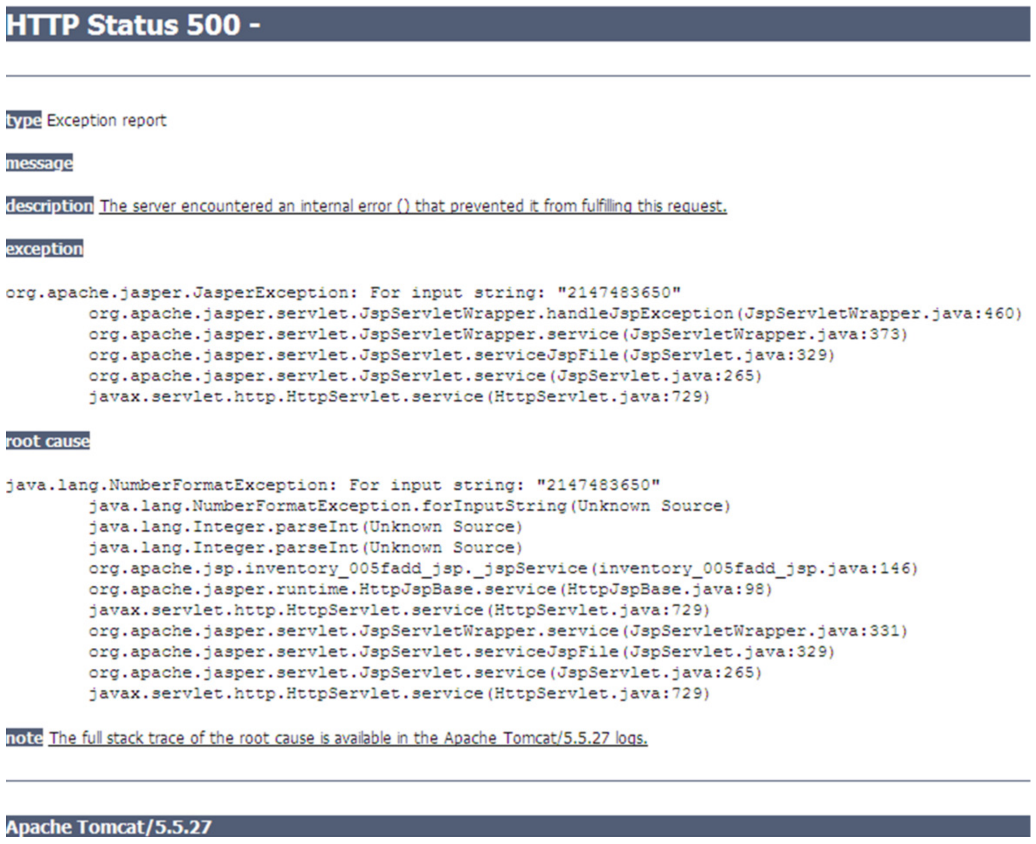
Goal: To test if the software process allows violation of the assumption: *the value of an integer/expression (signed & unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer variable.*

Target: Inputs of the process using integer values.

Method: Provide the process with values exceeding the maximum (minimum) values of integer variables as input. The assumption is violated if the process stores and uses these values.

Inputs of the Horticulture Club Sales Assistant expecting integer values are targeted for this test strategy. Inputs for the public portion of the Horticulture Club Sales Assistant allow accept both characters and numbers. However, the private portion of the Horticulture Club sales Assistant includes several inputs that expect only integers. The integer input targeted for this test exists on the web page for adding new inventory to the Horticulture Club Sales Assistant. To perform this test strategy, values exceeding the maximum and minimum value that can be stored inside an integer value are provided to the Horticulture Club Sales Assistant.

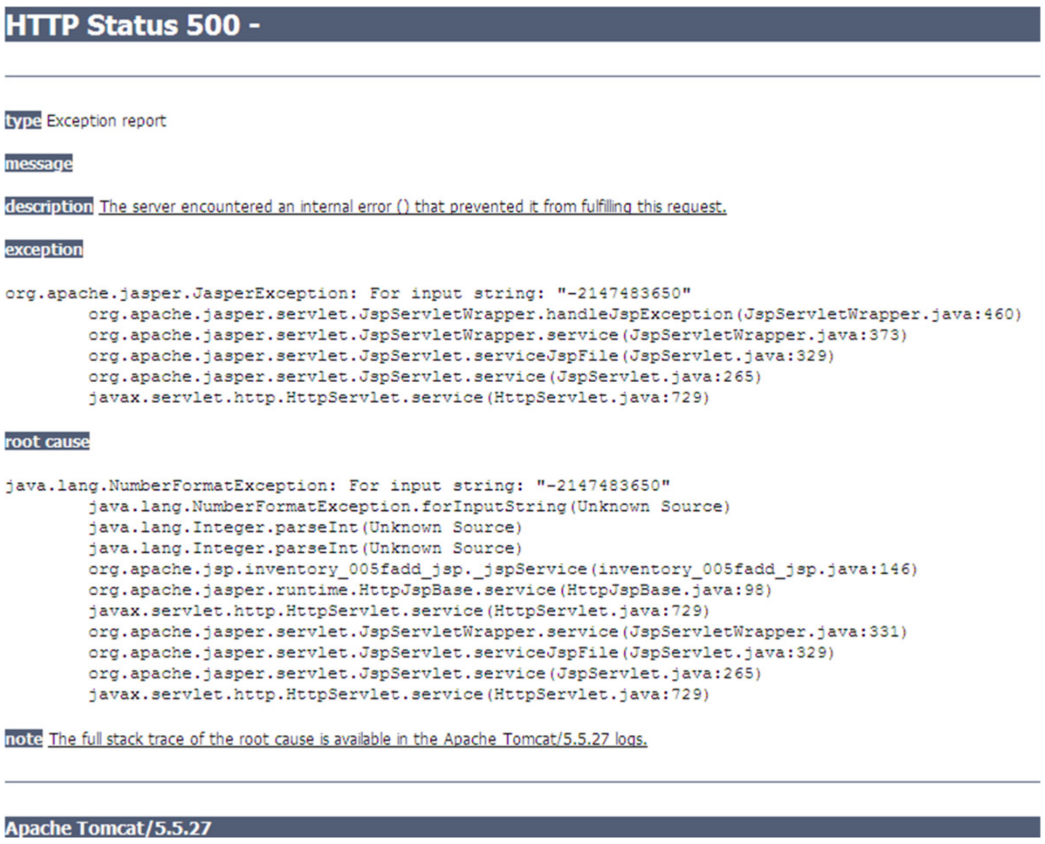
The maximum value that can be stored by an integer variable in the Java programming language is $2^{31} – 1$ (2147483647). Ensuring that the value given to the Horticulture Club Sales Assistant exceeds the maximum value that can be held by an integer, the Horticulture Club Sales Assistant is given a value of 2147483650. Submitting the new inventory object with the number exceeding the maximum integer value to the Horticulture Club Sales Assistant is responded to with the web page shown in Figure 21.



```
HTTP Status 500 -

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

org.apache.jasper.JasperException: For input string: "2147483650"
	org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:460)
	org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:373)
	org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:329)
	org.apache.jasper.servlet.JspServlet.service(JspServlet.java:265)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:729)

root cause

java.lang.NumberFormatException: For input string: "2147483650"
	java.lang.NumberFormatException.forInputString(Unknown Source)
	java.lang.Integer.parseInt(Unknown Source)
	java.lang.Integer.parseInt(Unknown Source)
	org.apache.jsp.inventory_005fadd_jsp._jspService(inventory_005fadd_jsp.java:146)
	org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:98)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
	org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:331)
	org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:329)
	org.apache.jasper.servlet.JspServlet.service(JspServlet.java:265)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:729)

note The full stack trace of the root cause is available in the Apache Tomcat/5.5.27 logs.

Apache Tomcat/5.5.27
```

Figure 21. Error report for exceeding the maximum integer value.

The Horticulture Club Sales Assistant is also tested for values exceeding the minimum value that can be stored by an integer variable. The minimum value that can be stored by an integer variable in the Java programming language is -2^{31} (-2147483648). Ensuring that the value given to the Horticulture Club Sales Assistant exceeds the minimum value, the Horticulture Club Sales Assistant is given a value of -2147483650. Submitting the new inventory object with the number exceeding the minimum integer value to the Horticulture Club Sales Assistant is responded to with the web page shown in Figure 22.



HTTP Status 500 -

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: For input string: "-2147483650"
	org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:460)
	org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:373)
	org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:329)
	org.apache.jasper.servlet.JspServlet.service(JspServlet.java:265)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
```

root cause

```
java.lang.NumberFormatException: For input string: "-2147483650"
	java.lang.NumberFormatException.forInputString(Unknown Source)
	java.lang.Integer.parseInt(Unknown Source)
	java.lang.Integer.parseInt(Unknown Source)
	org.apache.jsp.inventory_005fadd_jsp._jspService(inventory_005fadd_jsp.java:146)
	org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:98)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
	org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:331)
	org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:329)
	org.apache.jasper.servlet.JspServlet.service(JspServlet.java:265)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
```

note The full stack trace of the root cause is available in the Apache Tomcat/5.5.27 logs.

Apache Tomcat/5.5.27

Figure 22. Error report for exceeding the minimum integer value.

Exception reports are generated for both the maximum and minimum integer tests. These exceptions do not allow the Horticulture Club Sales Assistant to accept and use these values. Analyzing the database after receiving the exception reports shows that the data is not accepted by the Horticulture Club Sales Assistant and the new inventory is not added. Also, while the exceptions are not handled gracefully by the software process, they do not cause the Horticulture Club Sales Assistant to terminate (the web server can still be accessed). Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the assumption: *the value of an integer variable/expression (signed & unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer value.*

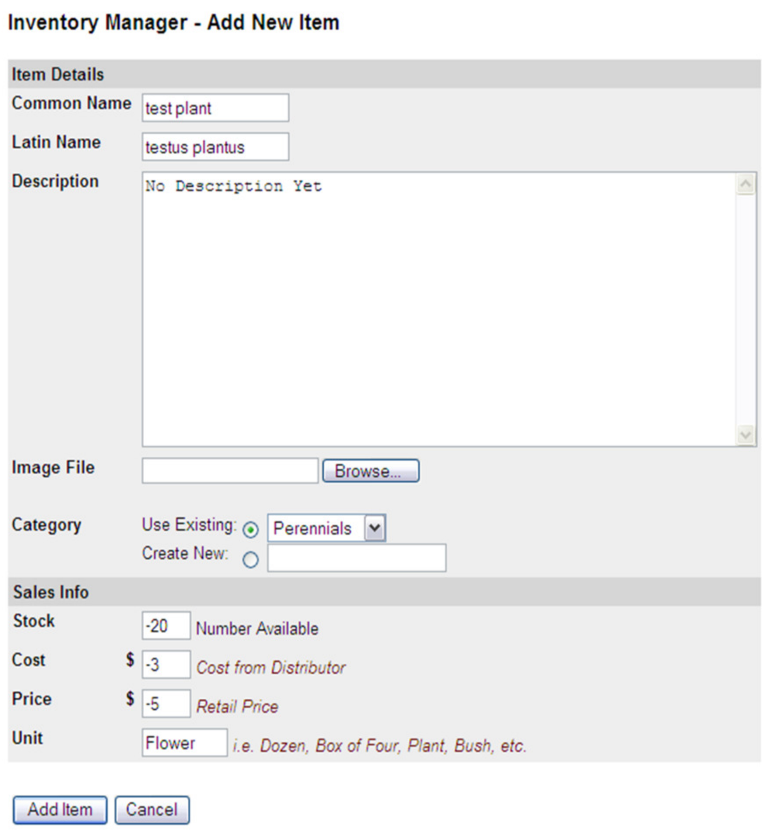
5.3.7 Test Strategy #7

Goal: To test if the software process allows violation of the constraint: *an integer variable/expression used by the process to indicate length/quantity of an object must not hold negative values.*

Target: Inputs of the process indicating length/quantity.

Method: Provide the process with negative values as input. The constraint is violated if the process accepts and uses these values.

The Horticulture Club Sales Assistant uses quantity to keep track of inventory. This test targets the web page for adding inventory to the Horticulture Club Sales Assistant. On this page there is an input for entering the quantity of the plant being added. The quantity input box is given a value of -20 as shown in Figure 23. To test other inputs on the page, the cost input of the form is given a value of -3 and the price input of the form is given a value of -5. These two inputs do not relate to length or quantity, however, they should be non-negative as they represent a unit of cost.


Inventory Manager - Add New Item

Item Details

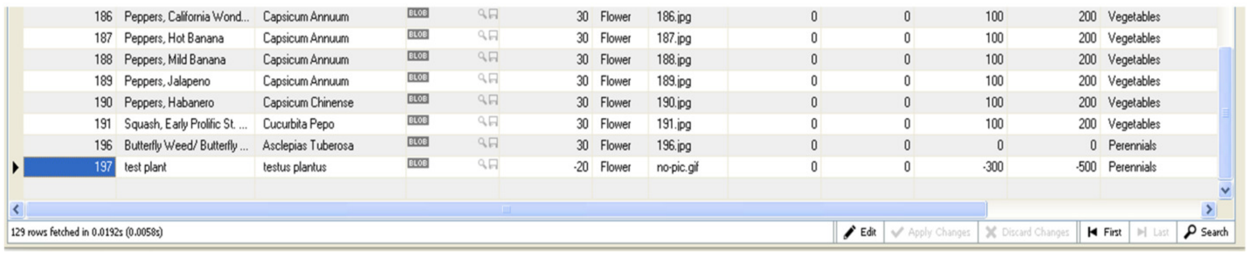
Common Name	test plant
Latin Name	testus plantus
Description	No Description Yet
Image File	
Category	Use Existing: [✓ Perennials] Create New:

Sales Info

Stock	-20	Number Available
Cost	$ -3	Cost from Distributor
Price	$ -5	Retail Price
Unit	Flower	i.e. Dozen, Box of Four, Plant, Bush, etc.

Figure 23. Add inventory form with negative input values.

Submitting these values to the software process forwards the web page back to the main administrative menu without any warning or error that negative values have been submitted to represent either quantity or cost. Figure 24 shows the newly added inventory item in the database. All three negative values are accepted and used by the software process. Because the Horticulture Club Sales Assistant accepts and uses these negative values, the Horticulture Club Sales Assistant violates the constraint: *an integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values*.



186	Peppers, California Wond...	Capsicum Annuum	BLOB		30	Flower	186.jpg	0	0	100	200	Vegetables
187	Peppers, Hot Banana	Capsicum Annuum	BLOB		30	Flower	187.jpg	0	0	100	200	Vegetables
188	Peppers, Mild Banana	Capsicum Annuum	BLOB		30	Flower	188.jpg	0	0	100	200	Vegetables
189	Peppers, Jalapeno	Capsicum Annuum	BLOB		30	Flower	189.jpg	0	0	100	200	Vegetables
190	Peppers, Habanero	Capsicum Chinense	BLOB		30	Flower	190.jpg	0	0	100	200	Vegetables
191	Squash, Early Prolific St. ...	Cucurbita Pepo	BLOB		30	Flower	191.jpg	0	0	100	200	Vegetables
196	Butterfly Weed/ Butterfly ...	Asclepias Tuberosa	BLOB		30	Flower	196.jpg	0	0	0	0	Perennials
197	test plant	testus plantus	BLOB		-20	Flower	no-pic.gif	0	0	-300	-500	Perennials

129 rows fetched in 0.0192s (0.0058s)

Figure 24. SQL database showing acceptance of negative values.

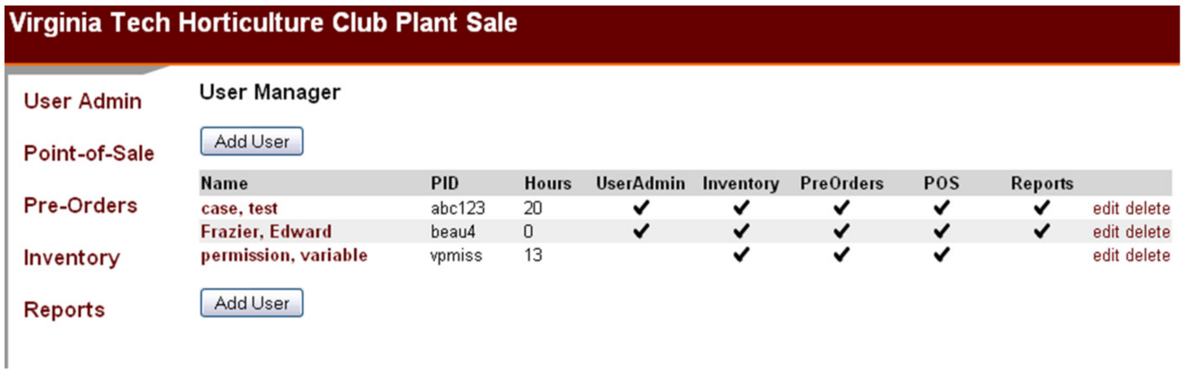
5.3.8 Test Strategy #8

Goal: To test if the software process allows violation of the constraint: *access permissions assigned to newly created files/directories are such that only the required principals have access to them.*

Target: Newly created files.

Method: Examine the access permissions of newly created files. The constraint is violated if any principal other than the required principals has access to the files.

Testing for violation of this constraint is directed at the private portion of the Horticulture Club Sales Assistant. All principals have access to the public portion of the Horticulture Club Sales Assistant and therefore, this constraint is irrelevant to the public portion of the Horticulture Club Sales Assistant. Newly created files of the private portion of the Horticulture Club Sales Assistant include picture files being uploaded when adding inventory and reports that can be generated based on sales figures for the Horticulture Club Sales Assistant. Each of these files can only be created by users of the Horticulture Club Sales Assistant with associated permissions. A sample table of users and associated permissions can be seen in Figure 25.


Virginia Tech Horticulture Club Plant Sale

User Admin
Point-of-Sale
Pre-Orders
Inventory
Reports

User Manager

Add User

Name	PID	Hours	UserAdmin	Inventory	PreOrders	POS	Reports	
case, test	abc123	20	✓	✓	✓	✓	✓	edit delete
Frazier, Edward	beau4	0	✓	✓	✓	✓	✓	edit delete
permission, variable	vpmiss	13		✓	✓	✓		edit delete

Add User

Figure 25. Sample table of users for the Horticulture Club Sales Assistant program.

Picture files being used by the software process can only be created, modified or deleted by users having the “Inventory” permission shown in Figure 25. All principals of the software process are able to read the newly created picture files as they are used in displaying inventory to the public web server. Unlike newly created picture files, newly created reports can only be read by users possessing the “Reports” permission shown in Figure 25. The “Reports” permission is required by any principal trying to create, modify, access, or delete a report file. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the constraint: *access permissions assigned to newly created files/directories are such that only the required principals have access to them*.

5.3.9 Test Strategy #9

Goal: To test if the software process allows violation of the constraint: *access permissions of the files/directories being used by the process are such that only the required principals have access to them.*

Target: Used files/directories.

Method: Examine the access permissions of files/directories used by the process. The constraint is violated if any principal other than the required principals has access to the files/directories.

As in Test Strategy #8, the focus of this test is on the private section of the Horticulture Club Sales Assistant. The public section of the Horticulture Club Sales Assistant allows read access to both the inventory database and picture files associated with inventory objects. This access is necessary in order to make the web application available to the public. Write access is also allowed by the public section of the Horticulture Club Sales Assistant to databases recording sales for the web application. However, the Horticulture Club Sales Assistant itself has the write access and this permission is not granted to the public user.

Files used by the private section of the Horticulture Club Sales Assistant are the database files storing users of the system, inventory, and recorded sales. Each of the files has an associated permission listed in Figure 25 restricting access to only required principals. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the constraint: *access permissions of the files/directories being used by the process are such that only the required principals have access to them*.

5.3.10 Test Strategy #10

Goal: To test if the software process allows violation of the assumption: *a file being created by the process does not have the same name as an already existing file.*

Target: Newly created files.

Method: Anticipate the name of a future file created by the process. Create a file with the same name and place in the location of the future file. The assumption is violated if the process uses the file not created by the process or terminates suddenly.

New files created by the Horticulture Club Sales Assistant are restricted to picture files that are uploaded when a new inventory object is added to the system. Test inventory items are provided as input to the system to try and determine the naming convention used for newly created files. It is observed that the names of newly created files increase by 1 numerically and retain the file extension of the uploaded file. Thus, the name of the next newly created file will be the name of the most recently created file incremented by 1, plus the new file's extension.

A file with the expected filename and extension of the future file is created and placed in the directory where the newly created file will be placed. Loading a new inventory object to the Horticulture Club Sales Assistant causes the Horticulture Club Sales Assistant to create a file with the same filename as the one already existing in the directory. The Horticulture Club Sales Assistant overwrites the file previously in the directory that has the same filename as the newly created file. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the assumption: *a file being created by the process does not have the same name as an already existing file*.

5.3.11 Test Strategy #11

Goal: To test if the software process allows violation of the assumption: *a filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permission.*

Target: Files created by the process.

Method: Provide the process with a file containing executable code. The assumption is violated if the process executes code within the file that the user would otherwise not have the permissions to execute.

Through violation of another assumption, filenames of any extension are able to be uploaded to the Horticulture Club Sales Assistant using the add inventory functionality. When adding a picture file, the extension of the uploaded file is not checked and the file is simply renamed with the same file extension and stored by the process. This allows for files with extensions such as .exe and .jsp to be uploaded to the system. The Horticulture Club Sales Assistant uses .jsp files in its execution. Therefore, .jsp files can be loaded and executed by the web server. Furthermore, browsing the public website and monitoring the status bar, or even using a packet sniffer on the public website to monitor HTTP requests, can provide the directory in which uploaded picture files are stored by the Horticulture Club Sales Assistant.

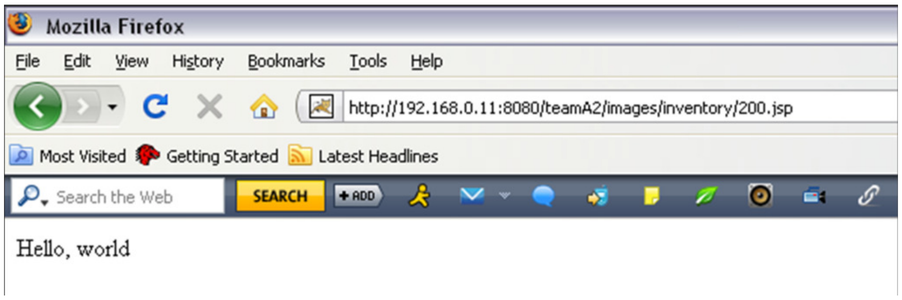
Using this information, a .jsp file can be uploaded to the Horticulture Club Sales Assistant and then accessed by directing the web browser to the URL of the file. To test this, two files are uploaded to the system with the .jsp extension. The first file contains simple HTML code:

```
<HTML>
<BODY>
Hello, world
</BODY>
</HTML>
```

The second file of the system contains the executable line of code:

```
<% System.exit(1); %>
```

Accessing the files directly over the web browser causes the web server to execute and display what is in the files. For the first file containing the HTML code, the web server reads the file and displays "Hello, world" to the web browser, shown in Figure 26.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
Most Visited Getting Started Latest Headlines
Search the Web SEARCH ADD

http://192.168.0.11:8080/teamA2/images/inventory/200.jsp

Hello, world

Figure 26. Web browser displaying hello world.

Contents of the first file are harmless to the system, but testing of the first file shows a significant security flaw in the Horticulture Club Sales Assistant. In testing the second file, the security flaw is magnified. The line of code *System.exit(1)* causes the JVM to stop running. When the JVM is not running, JSP files are not able to be processed and displayed to the web browser, turning the test file into an effective denial of service attack.

Tests show that a user is able to upload a file to the software process and run the file as executable code even though the user does not have (or should not have) permissions to load an executable file to the web server. Uploading of files for the Horticulture Club Sales Assistant is protected by user authentication; however, it must be assumed that a user/password combination can be compromised and that a valid user of the system could knowingly or unknowingly exploit this security flaw. Because a user can upload and execute a file for which they do not have access permissions, the Horticulture Club Sales Assistant violates the assumption: *a filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permissions*.

5.3.12 Test Strategy #12

Goal: To test if the software process allows violation of the assumption: *a file created/populated by a principal other than the process and being used by the process will have expected format and data.*

Target: Files uploaded to the process.

Method: Provide the process files with unexpected format or data. The assumption is violated if the process accepts and uses these files.

Picture files uploaded to the Horticulture Club Sales Assistant are targeted for this test strategy. The picture files are created outside of the Horticulture Club Sales Assistant and are then used by the Horticulture Club Sales Assistant when displaying inventory to the web browser. A similar approach to test strategy #11 is used and a file having a .jsp extension and containing the line of code *System.exit(1)* is uploaded to the Horticulture Club Sales Assistant. Differing from test strategy #11, the Horticulture Club Sales Assistant must access the file itself and not directed to by using the URL of the file.

When accessing the details of an inventory object, the software process attempts to display the picture file associated with the inventory object, if the file exists. After adding a new inventory object, an inventory details page is displayed showing the new inventory object along with its uploaded picture file, if any. When the Horticulture Club Sales Assistant attempts to render the test file as a picture, code in the file is executed instead. This is evident when the web browser freezes temporarily on the inventory details page and afterwards, any attempt to access system web pages is met with a network timeout error. The time out error can be explained by the software process executing the code *System.exit(1)*, causing the JVM of the web server to stop running and thus not allowing the web server to handle incoming requests.

Because the Horticulture Club Sales Assistant accepts and uses a file of invalid format and containing invalid data, the Horticulture Club Sales Assistant violates the assumption: *a file create/populated by a principal other than the process and being used by the process will have expected format and data*. This violation can be avoided by placing restrictions on the type of files that can be uploaded to the Horticulture Club Sales Assistant. Restricting expected picture files to have non-executable file extensions, such as .jpg and .bmp, will allow the process to function normally after accessing the files.

5.3.13 Test Strategy #13

Goal: To test if the software process allows violation of the constraint: *data held by files owned/used by the process must not be accessible after the process deletes them.*

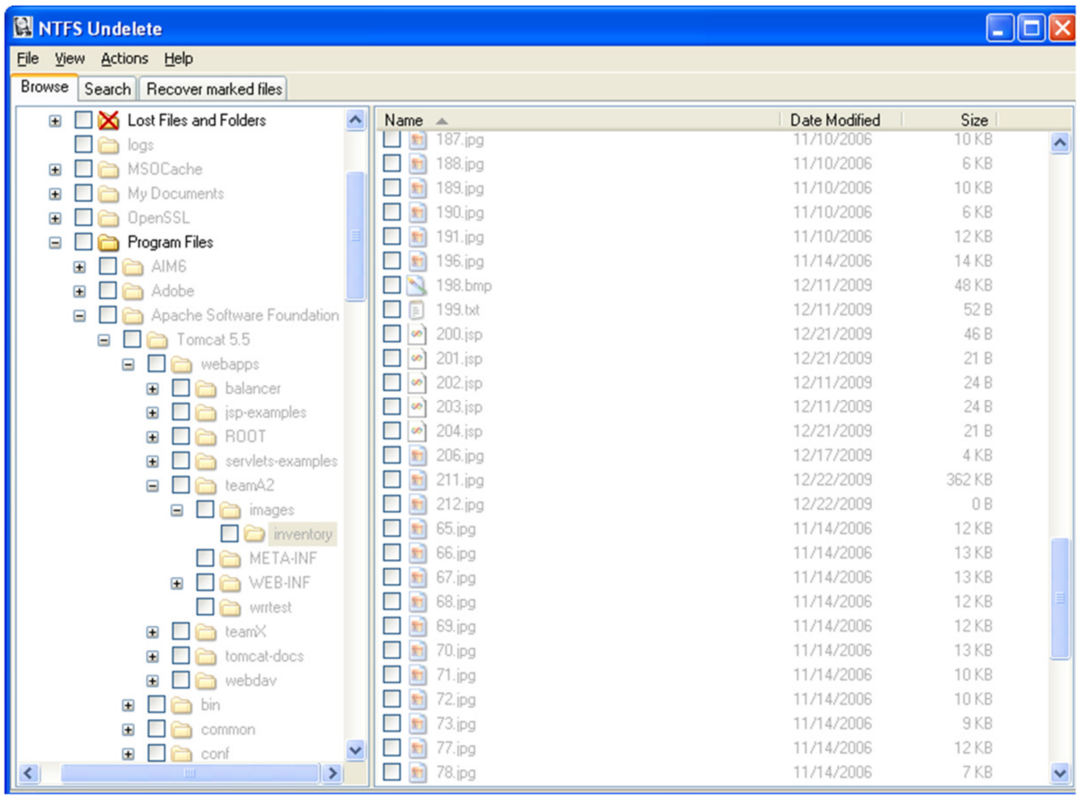
Target: Files deleted by the process.

Method: Attempt to access data held in files after the process deletes the files. The constraint is violated if the data is able to be accessed and read.

Picture files used for displaying inventory of the Horticulture Club Sales Assistant are targeted in this test strategy. Inventory objects of the Horticulture Club Sales Assistant have can be deleted by privileged users. Deletion of an inventory object from the Horticulture Club

Sales Assistant also means the deletion of the associated picture file, if any. To determine if files are able to be accessed after deletion, an automated data recovery tool is used. NTFS Undelete is a free data recovery tool that can be used to recover deleted files [66]. NTFS Undelete is installed before this test is performed to optimize results.

Before a file can be recovered, it first must be deleted. A test entry in the inventory database created during a previous test having an associated picture file with filename *210.jpg* is deleted using the inventory management system of the Horticulture Club Sales Assistant. After the inventory item is deleted, NTFS Undelete is immediately run to see if the deleted picture file can be recovered. Figure 27 shows the results of running NTFS Undelete after the test inventory item is deleted.



Name	Date Modified	Size
187.jpg	11/10/2006	10 KB
188.jpg	11/10/2006	6 KB
189.jpg	11/10/2006	10 KB
190.jpg	11/10/2006	6 KB
191.jpg	11/10/2006	12 KB
196.jpg	11/14/2006	14 KB
198.bmp	12/11/2009	48 KB
199.txt	12/11/2009	52 B
200.jsp	12/21/2009	46 B
201.jsp	12/21/2009	21 B
202.jsp	12/11/2009	24 B
203.jsp	12/11/2009	24 B
204.jsp	12/21/2009	21 B
206.jpg	12/17/2009	4 KB
211.jpg	12/22/2009	362 KB
212.jpg	12/22/2009	0 B
65.jpg	11/14/2006	12 KB
66.jpg	11/14/2006	13 KB
67.jpg	11/14/2006	13 KB
68.jpg	11/14/2006	12 KB
69.jpg	11/14/2006	12 KB
70.jpg	11/14/2006	13 KB
71.jpg	11/14/2006	10 KB
72.jpg	11/14/2006	10 KB
73.jpg	11/14/2006	9 KB
77.jpg	11/14/2006	12 KB
78.jpg	11/14/2006	7 KB

Figure 27. Results of NTFS Undelete after deletion of inventory item.

If the file is able to be recovered by NTFS Undelete, the file will be listed in the right hand pane. Of the files listed in the right hand pane of Figure 27, the file *210.jpg* is not one of them. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales

Assistant violates the constraint: *data held by files owned, used by the process must not be accessible after the process deleted them*.

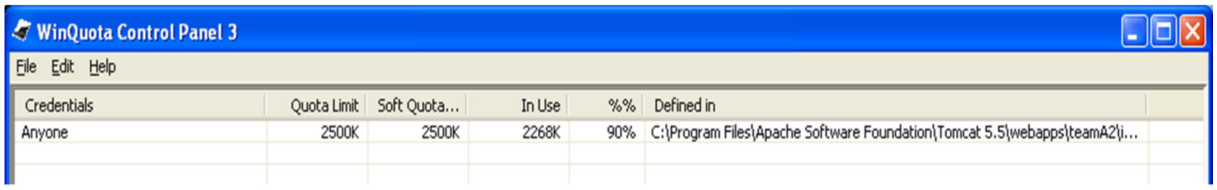
5.3.14 Test Strategy #14

Goal: To test if the software process allows violation of the constraint: *the process must be provided with the filesystem space that it requests.*

Target: Filesystem size.

Method: Attempt to restrict the amount of filesystem space available to the process and exceed this limit by loading files too big to store on the reduced space. The constraint is violated if the process freezes or terminates abruptly.

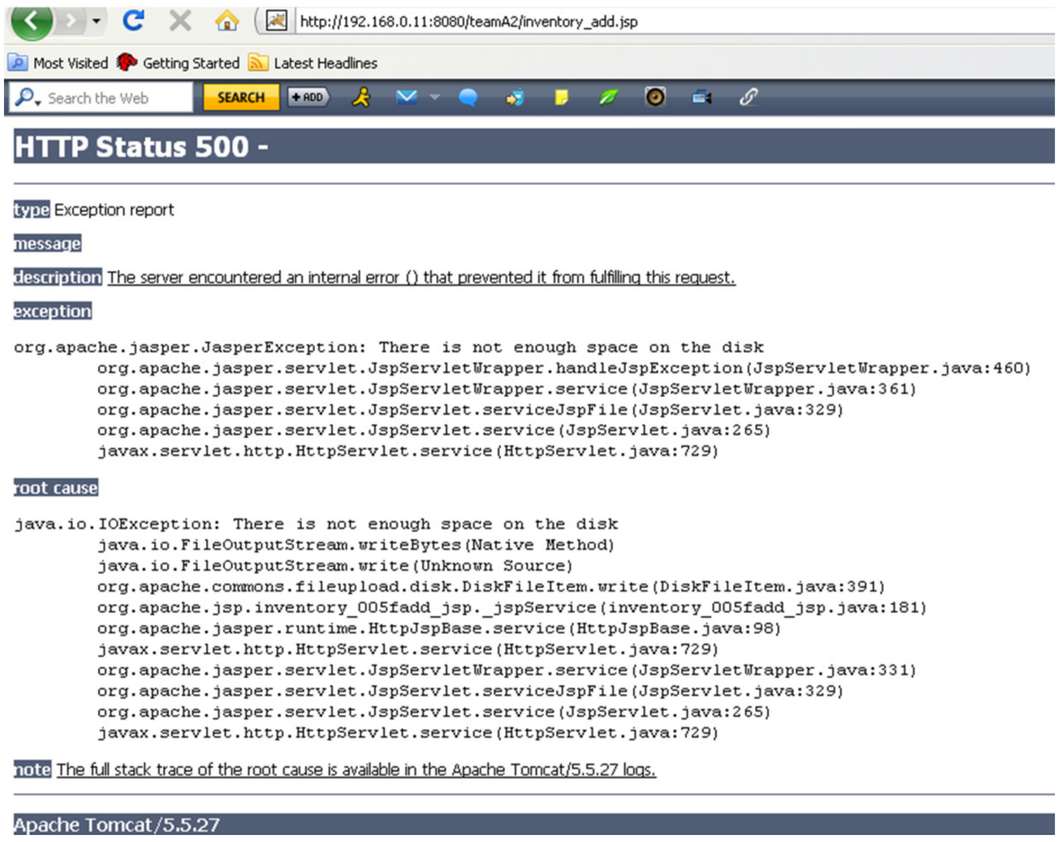
To restrict the amount of filesystem space available to the Horticulture Club Sales Assistant, a program called WinQuota [67] is used. Using WinQuota, the directory in which uploaded picture files are stored by the Horticulture Club Sales Assistant is targeted and the filesystem space available to the directory is restricted to 2500KB. Memory allotted to the directory is set just above the amount of memory currently used by the directory so that the filesystem space available will be exceeded by uploading a single file. Figure 28 shows the WinQuota program and the limits placed on the directory storing inventory image files.



Credentials	Quota Limit	Soft Quota...	In Use	%%	Defined in
Anyone	2500K	2500K	2268K	90%	C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\teamA2\i...

Figure 28. WinQuota showing image directory size restriction.

Exceeding filesystem space available to the directory requires uploading a file to the system whose size is greater than 232KB. A new inventory object is created for the Horticulture Club Sales Assistant with an image file exceeding 300KB in size. Using a file of this size guarantees that the filesystem space available to the directory will be exceeded. After submitting the inventory object to the Horticulture Club Sales Assistant, the web server responds with an error page, shown in Figure 29. The error page indicates that there is not enough space on disk to perform the requested operation.



HTTP Status 500 -

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: There is not enough space on the disk
	org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:460)
	org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:361)
	org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:329)
	org.apache.jasper.servlet.JspServlet.service(JspServlet.java:265)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
```

root cause

```
java.io.IOException: There is not enough space on the disk
	java.io.FileOutputStream.writeBytes(Native Method)
	java.io.FileOutputStream.write(Unknown Source)
	org.apache.commons.fileupload.disk.DiskFileItem.write(DiskFileItem.java:391)
	org.apache.jsp.inventory_005fadd_jsp._jspService(inventory_005fadd_jsp.java:181)
	org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:98)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
	org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:331)
	org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:329)
	org.apache.jasper.servlet.JspServlet.service(JspServlet.java:265)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
```

note The full stack trace of the root cause is available in the Apache Tomcat/5.5.27 logs.

Apache Tomcat/5.5.27

Figure 29. Error page indicating not enough space on disk.

Due to the web browser displaying this error message, it is clear that the Horticulture Club Sales Assistant does not handle the exception of not having enough space on disk gracefully. The Horticulture Club Sales Assistant does not terminate abruptly due to the exception because the homepage of the process is still able to be accessed after receiving the error page. While the exception is not handled gracefully by the software process, the process does not freeze or terminate abruptly when filesystem space has been exceeded. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the constraint: *the process must be provided with the filesystem space that it requests*.

5.3.15 Test Strategy #15

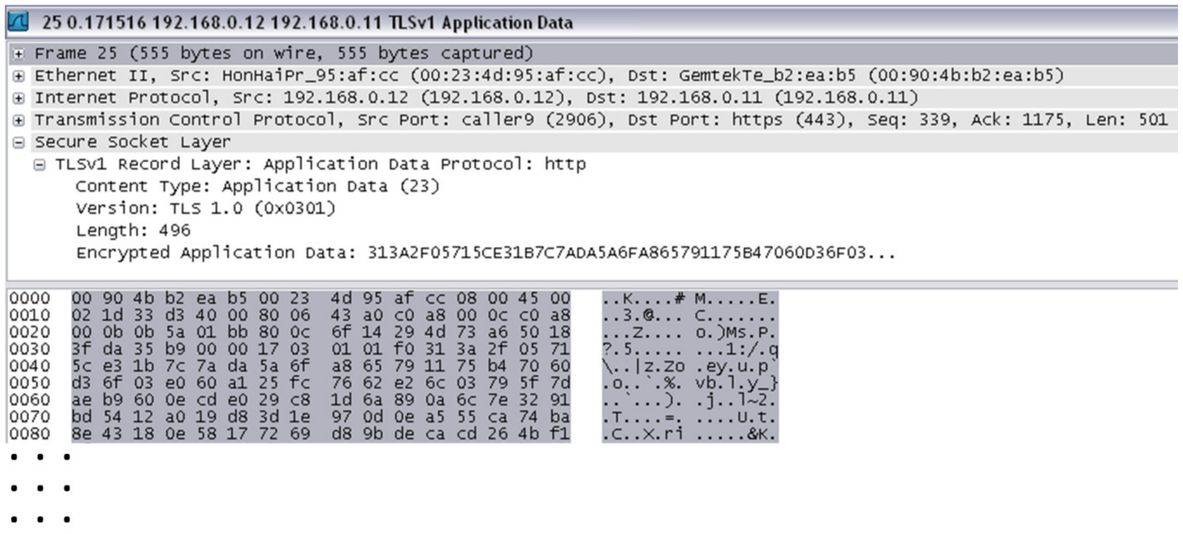
Goal: To test if the software process allows violation of the assumption: *the data received by the software process through the network interface is neither read nor modified by anyone other than the intended recipient.*

Target: Data received by the software process.

Method: Intercept data being sent to the software process and attempt to read or modify it. The assumption is violated if the data is able to be read or the process uses the modified data.

Both the public and private portions of the Horticulture Club Sales Assistant must be considered in this test. The public side of software process allows navigation and simple searching via the HTTP protocol. Because public functionality of the Horticulture Club Sales Assistant using the HTTP protocol does not include the exchange of sensitive data, it is not being considered for this test. More advanced functionality that includes the transfer of sensitive data for both the public and private portions of the software process requires a secure connection. This secure connection is established using the HTTPS protocol. Data received by the Horticulture Club Sales Assistant over this connection is targeted during this test.

Wireshark is used to capture data being sent to the process to determine if the data can be read or modified. Wireshark is a free network protocol analyzer that has the ability to capture data packets traversing a computer network [68]. Figure 30 shows a data packet that is captured using the Wireshark protocol analyzer. The destination IP address for the packet is 192.168.0.11. Testing is performed on a local subnet and 192.168.0.11 is the designated IP address for the Horticulture Club Sales Assistant. The HTTPS protocol is used in sending the data packet and can be determined from Figure 30 by the Application Data Protocol being HTTP. However, the HTTP data is sent over a TLS connection. Because the HTTP data is sent over a TLS connection, this packet is sent using the HTTPS protocol.



```
25 0.171516 192.168.0.12 192.168.0.11 TLSv1 Application Data
⊞ Frame 25 (555 bytes on wire, 555 bytes captured)
⊞ Ethernet II, Src: HonHaiPr_95:af:cc (00:23:4d:95:af:cc), Dst: GemtekTe_b2:ea:b5 (00:90:4b:b2:ea:b5)
⊞ Internet Protocol, Src: 192.168.0.12 (192.168.0.12), Dst: 192.168.0.11 (192.168.0.11)
⊞ Transmission Control Protocol, Src Port: caller9 (2906), Dst Port: https (443), Seq: 339, Ack: 1175, Len: 501
⊟ Secure Socket Layer
  ⊟ TLSv1 Record Layer: Application Data Protocol: http
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 496
      Encrypted Application Data: 313A2F05715CE31B7C7ADA5A6FA865791175B47060D36F03...

0000  00 90 4b b2 ea b5 00 23  4d 95 af cc 08 00 45 00   ..K....# M.....E.
0010  02 1d 33 d3 40 00 80 06  43 a0 c0 a8 00 0c c0 a8   ..3.@... C.......
0020  00 0b 0b 5a 01 bb 80 0c  6f 14 29 4d 73 a6 50 18   ...Z.... o.)Ms.P.
0030  3f da 35 b9 00 00 17 03  01 01 f0 31 3a 2f 05 71   ?.5..... ...1:/.q
0040  5c e3 1b 7c 7a da 5a 6f  a8 65 79 11 75 b4 70 60   \..|z.Zo .ey.u.p`
0050  d3 6f 03 e0 60 a1 25 fc  76 62 e2 6c 03 79 5f 7d   .o..`.%. vb.l.y_}
0060  ae b9 60 0e cd e0 29 c8  1d 6a 89 0a 6c 7e 32 91   ..`...). .j..l~2.
0070  bd 54 12 a0 19 d8 3d 1e  97 0d 0e a5 55 ca 74 ba   .T....=. ....U.t.
0080  8e 43 18 0e 58 17 72 69  d8 9b de ca cd 26 4b f1   .C..X.ri .....&K.
• • •
• • •
• • •
```

Figure 30. Packet capture of data received by the software process.

The Encrypted Application Data shown in Figure 30 is data being sent to the Horticulture Club Sales Assistant by a client web browser. While this data may be able to be modified, it will have no affect on the Horticulture Club Sales Assistant. The Horticulture Club Sales Assistant is protected against using modified data. The HTTPS protocol uses a message digest at the end of transmitted data to ensure that data has not been modified. If the data is detected as being modified, then the web server will discard the data packet and alert the client web browser. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the assumption: *the data received by the software process through the network interface is neither read nor modified by anyone other than the intended recipient*.

5.3.16 Test Strategy #16

Goal: To test if the software process allows violation of the assumption: *the data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.*

Target: Data sent to the software process.

Method: Send data of unexpected format and length to the process. The assumption is violated if the process accepts and uses this data.

Being an online web application, all data entered into the Horticulture Club Sales Assistant by a user is done via the network interface. As such, the results of Test Strategy #1 are applicable to this test strategy as well. In Test Strategy #1, SQL injection is used to cause unexpected behavior within the Horticulture Club Sales Assistant. The partial SQL command entered into the username input box is not in the format of an acceptable Virginia Tech PID [69], which is expected by the Horticulture Club Sales Assistant. Because the partial SQL command is both accepted and used, the Horticulture Club Sales Assistant violates the assumption: *the data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length*.

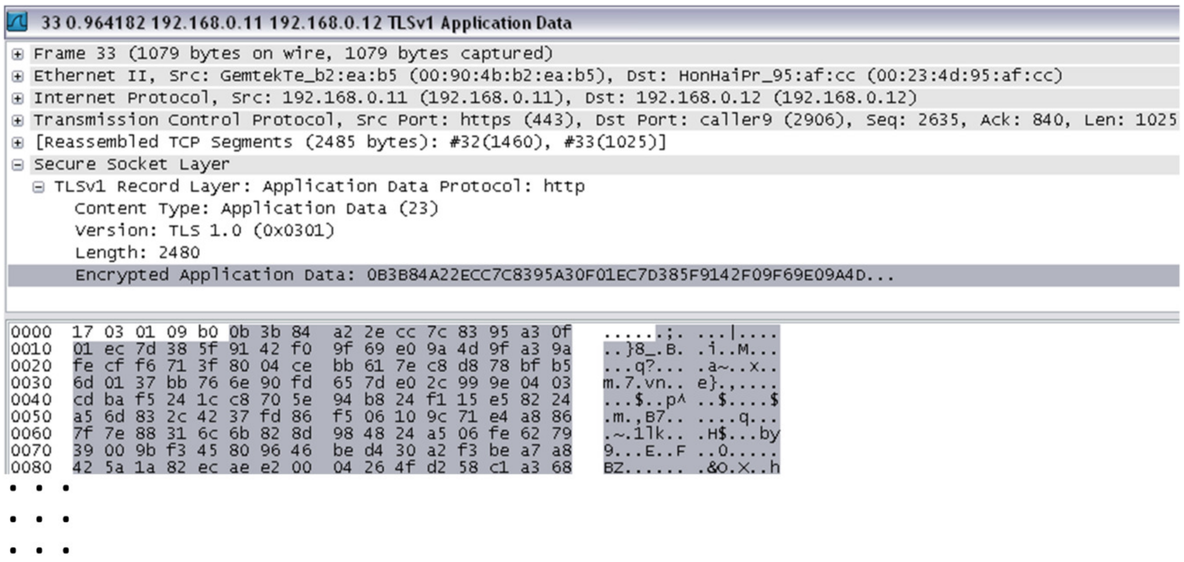
5.3.17 Test Strategy #17

Goal: To test if the software process allows violation of the assumption: *the data sent by the software process via the network interface will not be read/modified before it reaches its destination.*

Target: Data sent by the software process.

Method: Intercept data sent by the software process and attempt to read or modify it. The assumption is violated if the data is able to be read or the receiving process uses the modified data.

The same steps are taken for this Test Strategy as in Test Strategy #15. Figure 31 shows a data packet that is captured using the Wireshark network protocol analyzer. The packet shows that data is being sent from the Horticulture Club Sales Assistant, IP address 192.168.0.11, to the client. Data is sent using the HTTPS protocol which protects the data from being modified before it is received. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the assumption: *the data sent by the software process via the network interface will not be read/modified before it reaches its destination*.



```
33 0.964182 192.168.0.11 192.168.0.12 TLSv1 Application Data
⊞ Frame 33 (1079 bytes on wire, 1079 bytes captured)
⊞ Ethernet II, Src: GemtekTe_b2:ea:b5 (00:90:4b:b2:ea:b5), Dst: HonHaiPr_95:af:cc (00:23:4d:95:af:cc)
⊞ Internet Protocol, Src: 192.168.0.11 (192.168.0.11), Dst: 192.168.0.12 (192.168.0.12)
⊞ Transmission Control Protocol, Src Port: https (443), Dst Port: caller9 (2906), Seq: 2635, Ack: 840, Len: 1025
⊞ [Reassembled TCP Segments (2485 bytes): #32(1460), #33(1025)]
⊟ Secure Socket Layer
  ⊟ TLSv1 Record Layer: Application Data Protocol: http
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 2480
      Encrypted Application Data: 0B3B84A22ECC7C8395A30F01EC7D385F9142F09F69E09A4D...

0000  17 03 01 09 b0 0b 3b 84  a2 2e cc 7c 83 95 a3 0f   ......;. ...|....
0010  01 ec 7d 38 5f 91 42 f0  9f 69 e0 9a 4d 9f a3 9a   ..}8_.B. .i..M...
0020  fe cf f6 71 3f 80 04 ce  bb 61 7e c8 d8 78 bf b5   ...q?... .a~..x..
0030  6d 01 37 bb 76 6e 90 fd  65 7d e0 2c 99 9e 04 03   m.7.vn.. e}.,....
0040  cd ba f5 24 1c c8 70 5e  94 b8 24 f1 15 e5 82 24   ...$..p^ ..$....$
0050  a5 6d 83 2c 42 37 fd 86  f5 06 10 9c 71 e4 a8 86   .m.,B7.. ....q...
0060  7f 7e 88 31 6c 6b 82 8d  98 48 24 a5 06 fe 62 79   .~.1lk.. .H$...by
0070  39 00 9b f3 45 80 96 46  be d4 30 a2 f3 be a7 a8   9...E..F ..0.....
0080  42 5a 1a 82 ec ae e2 00  04 26 4f d2 58 c1 a3 68   BZ...... .&O.X..h
• • •
• • •
• • •
```

Figure 31. Packet capture of data sent by the software process.

5.3.18 Test Strategy #18

Goal: To test if the software process allows violation of the assumption: *the software process will be able to utilize the network interface to send and receive data.*

Target: Network connections of the software process.

Method: Limit the availability of the network interface to the software process. The assumption is violated if the process freezes or terminates abruptly.

Availability of the network interface to the Horticulture Club Sales Assistant for this test strategy is limited using the Windows Firewall available on the Windows XP operating system [70]. To make the network interface unavailable to the Horticulture Club Sales Assistant, the Windows Firewall is turned on and port 8080 is closed for use by the network interface. Port 8080 is the port used by the Tomcat web server for HTTP traffic and is used by the public portion of the Horticulture Club Sales Assistant. Attempts to access the software process after the port has been blocked by the Windows Firewall are unsuccessful. After unblocking port 8080, the home page of the Horticulture Club Sales Assistant is accessible. Because the home page is accessible, the Horticulture Club Sales Assistant does not freeze or terminate while the network interface is unavailable.

Database connections are also explored in this test strategy. Connection to the databases for the Horticulture Club Sales Assistant must be available to ensure correct operation of the Horticulture Club Sales Assistant. If connections are not properly closed by the process and are abandoned, database connection resources can be exhausted. After accessing 5 to 10 web pages, the Horticulture Club Sales Assistant is no longer able to respond to requests from the client web browser. Examining files specifying connection parameters for the Horticulture Club Sales Assistant, a maximum of 10 database connections are allowed at one time. There is no evidence that abandoned connections are freed, leading to an exhaustion of database connection resources. Exhaustion of database resources can lead to a denial of service against the Horticulture Club Sales Assistant. Because connection resources can be exhausted and thus limiting access to the network interface, the Horticulture Club Sales Assistant violates the assumption: *the software process will be able to utilize the network interface to send and receive data*.

5.3.19 Test Strategy #19

Goal: To test if the software process allows violation of the assumption: *user selected passwords/keys will have a sufficient amount of entropy.*

Target: User selected passwords.

Method: Identify the origins of the users selected passwords used by the software process. The assumption is violated if the passwords were not created under requirements that would enforce increased entropy.

Passwords chosen by users of the Horticulture Club Sales Assistant are associated with their Virginia Tech PID. As such, these passwords are subject to requirements set forth by the Virginia Tech password selection process [60]. Requirements placed on passwords create a sufficient amount of entropy such that passwords cannot be easily guessed and will have reasonable protection against brute force attacks. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the assumption: *user selected passwords/keys will have a sufficient amount of entropy*.

5.3.20 Test Strategy #20

Goal: To test if the software process allows violation of the constraint: *the process cannot use encryption to ensure data integrity.*

Target: Securely transmitted data.

Method: Determine if the securely transmitted data is using encryption only in order to ensure data integrity. The constraint is violated if only encryption is being used to ensure data integrity.

Data is transmitted securely by the Horticulture Club Sales Assistant using the HTTPS protocol. The HTTPS protocol uses a hashed message digest in addition to encryption to help insure data integrity. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the constraint: *the process cannot use encryption to ensure data integrity*.

5.3.21 Test Strategy #21

Goal: To test if the software process allows violation of the constraint: *the process cannot use obfuscation instead of encryption to ensure confidentiality.*

Target: Data used by the process.

Method: Analyze data used by the software process and determine to what extents obfuscation is used to keep secrets. The constraint is violated if obfuscation is used to keep secrets.

The Horticulture Club Sales Assistant does not use obfuscation in order to ensure confidentiality of data traversing the network. Instead, the Horticulture Club Sales Assistant uses the HTTPS protocol to ensure confidentiality. Furthermore, obfuscation is not used to ensure confidentiality of any items stored locally by the process. Based on the refined test strategy, no evidence is found that the Horticulture Club Sales Assistant violates the constraint: *the process cannot use obfuscation instead of encryption to ensure confidentiality.*

5.3.22 Test Strategy #22

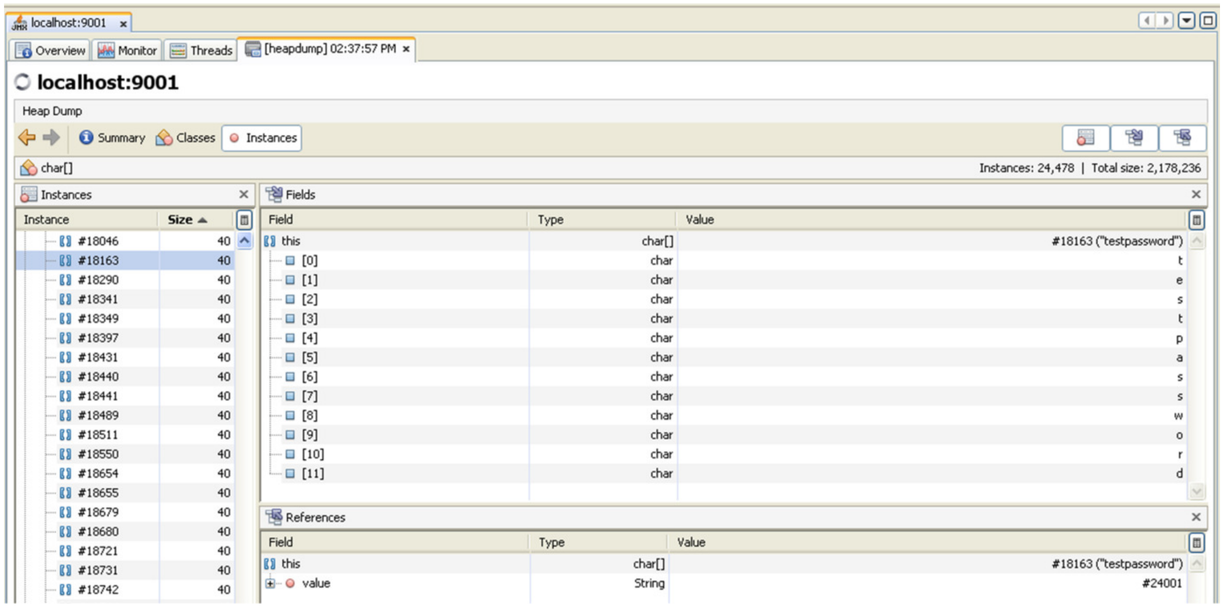
Goal: To test if the software process allows violation of the constraint: *the process cannot store keys/passwords in clear text.*

Target: Data store.

Method: Analyze the data store to determine if passwords are stored in clear text. The constraint is violated if the software process stores passwords in clear text.

Passwords stored by the Horticulture Club Sales Assistant are limited to the passwords associated with a Horticulture Club member's Virginia Tech PID. The database of user's for the Horticulture Club Sales Assistant does not store user passwords. User passwords can also be stored in dynamic memory. Using Java VisualVM, a heap dump is performed after a user is authenticated to the Horticulture Club Sales Assistant using the password "testpassword".

Figure 32 shows the password being stored in memory in clear text. Although the process does not store the password in any location other than the dynamic memory, the password is still stored in clear text. Because the password is stored in clear text, the Horticulture Club Sales Assistant violates the constraint: *the process cannot store keys/passwords in clear text*.



Instances: 24,478 | Total size: 2,178,236

Instance	Size
#18046	40
#18163	40
#18290	40
#18341	40
#18349	40
#18397	40
#18431	40
#18440	40
#18441	40
#18489	40
#18511	40
#18550	40
#18654	40
#18655	40
#18679	40
#18680	40
#18721	40
#18731	40
#18742	40

Fields

Field	Type	Value
this	char[]	#18163 ("testpassword")
[0]	char	t
[1]	char	e
[2]	char	s
[3]	char	t
[4]	char	p
[5]	char	a
[6]	char	s
[7]	char	s
[8]	char	w
[9]	char	o
[10]	char	r
[11]	char	d

References

Field	Type	Value
this	char[]	#18163 ("testpassword")
value	String	#24001

Figure 32. Heap dump revealing password stored in clear text.

5.4 Results

The application of End-Game VV&T to the Horticulture Club Sales Assistant is successful in identifying vulnerabilities of the system executable. Violations are found for dynamic memory, filesystem, network interface, and cryptographic algorithms and protocols constraints/assumptions. No violations found within the system executable are of static memory or randomness resources constraints/assumptions. Violations of the constraints/assumptions found within the system executable can imply the presence of software vulnerabilities, leaving the Horticulture Club Sales Assistant open to attack.

- Data present on the dynamic memory is interpreted as executable code by the Horticulture Club Sales Assistant. Interpreting data on dynamic memory as executable code can put an entire system at risk. Code statements can be inserted that crash the system and lead to a denial of service attack. Also, privileged data can be compromised by inserting code statements that when executed, display memory held in the program stack to the user.
- Out of memory errors are not appropriately handled by the Horticulture Club Sales Assistant. Out of memory errors should not cause the Horticulture Club Sales Assistant to crash. This vulnerability leads the Horticulture Club Sales Assistant open to denial of service attacks. If an attacker is able to cause the Horticulture Club Sales Assistant to exhaust memory resources, then the system may crash and not be able to recover when memory does become available.
- Data present on the dynamic memory can to be observed while the Horticulture Club Sales Assistant is in execution. If an attacker gains access to the computer system on which the Horticulture Club Sales Assistant is running, then an attacker will have the ability to view data stored on the dynamic memory. In the case of the Horticulture Club Sales Assistant, information stored in dynamic memory includes usernames, passwords, sale transaction numbers, names and addresses.
- Negative inventory, cost, and price values can be submitted to the Horticulture Club Sales Assistant. Representing inventory with negative values can cause problems for real world transactions. If the Horticulture Club Sales Assistant allows an object with negative inventory to be purchased, the selling entity might not be able to fulfill the purchase. Also, if an item has a negative price, purchasing the item might allow the buyer to receive a refund instead of being charged for the item.
- Files containing unexpected format can be uploaded to Horticulture Club Sales Assistant. This allows a potential attacker to upload files containing executable code to the Horticulture Club Sales Assistant. A flaw found within the Horticulture Club Sales Assistant can lead the uploaded file being executed by users whom do not have

executable permission. This scenario is a combination of the violation of two constraints/assumptions and can lead to such attacks as denial of service.

- Data of unexpected format can be received and used by the Horticulture Club Sales Assistant. Not thoroughly checking data received through the network interface before using it leads to vulnerabilities in the Horticulture Club Sales Assistant. Attacks such as cross-site scripting, SQL injection and denial of service can be performed using unchecked input values.
- Database connections used by the Horticulture Club Sales Assistant are not properly managed. There are a limited number of available connections and connections are often left open after the process has finished executing. Abandoned connections are not recycled, leading to an exhaustion of database resources. For a database centralized web application, such as the Horticulture Club Sales Assistant, an exhaustion of database resources can lead to denial of service.
- Passwords are stored in clear text by the system executable. If an attacker knows what they are looking for and has access to the computer system on which the Horticulture Club Sales Assistant is running, passwords can be compromised. Along with storing passwords, the username associated with the password is stored as well. This can give an attacker a legitimate username/password pair that can be used to gain access to the system.

The application of End-Game VV&T to the Horticulture Sales Club executable is successful in identifying potential vulnerabilities. Fixing violations of constraints/assumptions found during End-Game VV&T can help protect the Horticulture Club Sales Assistant from the possible attacks mentioned. Results show that the constraints/assumptions can successfully be applied to a software system using End-Game VV&T. Furthermore, the refined test strategies can identify violations of the constraints/assumptions in a way that produces usable results that can be used to better secure a software system.

5.5 Potential Improvements

Potential improvements to applying End-Game VV&T include using software specific constraints/assumptions, combining the constraint/assumption test strategies with existing test strategies, and submitting the system executable to an automated testing tool in addition to applying the test strategies. Software specific test strategies can be added to End-Game VV&T to enhance its application. Proposed software specific test strategies are not the same as refined base test strategies. Instead, software specific test strategies target individual components of the system executable. For example, if the software system utilizes a database, additional test strategies can be added to End-Game VV&T that target the functionality of the database.

Existing test strategies can be added to the constraint/assumption based test strategies to enhance the application of End-Game VV&T. Test strategies are already in place to test software components such as network connections and filesystem usage. These test strategies can be combined with the strategies of End-Game VV&T to better test the system resources targeted by the constraints/assumptions. An advantage of adding existing test strategies over adding software specific test strategies is that existing test strategies have already been developed.

Submitting the system executable to an automatic security testing tool can help improve the results of End-Game VV&T. Results from the testing tool can be combined with the results of End-Game VV&T to produce a broader image of a software system's security status. An automatic tool does not require much effort to use and will yield results much faster than test strategies used in End-Game VV&T. Thus, using an automated tool in conjunction with End-Game VV&T would not hinder the current End-Game VV&T process.

6. Conclusions

The purpose of this research is to determine if the constraints/assumptions technology can be applied to a software system using Partial and End-Game VV&T, and produce viable results that will enhance the security of the software system. Software security has become a major concern as the use of the Internet and web based applications have increased. Sophistication of attacks on software systems has increased, prompting the need for greater security assessment and security requirements. Section 6.1 summarizes work performed in this research and addresses the answer to the problem statement. Section 6.2 identifies major contributions of this work to the software security field and the constraints/assumptions technology. Section 6.3 discusses future work stemming from this research and the constraints/assumptions technology.

6.1 Summary

Partial and End-Game VV&T are applied to the Horticulture Club Sales Assistant to determine if they produced viable results and identify vulnerabilities of the system and enhance its overall security. Development artifacts of the Horticulture Club Sales Assistant relevant and beneficial to Partial VV&T are identified and reviewed under Partial VV&T. Constraints/assumptions are translated on an artifact to artifact basis to target instances of the constraints/assumptions within an artifact. Instances of constraints/assumptions within an artifact are placed in the artifact's constraint/assumption review table. Each instance in the table has an associated constraint/assumption status that is taken from the developed status nomenclature.

The constraint/assumption review tables are used to compile the results of Partial VV&T. Results of Partial VV&T reveal several vulnerabilities that exist in the Horticulture Club Sales Assistant. Tracing vulnerabilities through the constraint/assumption review tables yields the origin of the vulnerabilities in the development artifacts. Knowing the origin of vulnerabilities can allow portions of the software system to be reengineered rather than patched. Application of Partial VV&T produces results that are able to identify vulnerabilities of the Horticulture Club Sales Assistant that if addressed, can enhance the security of the Horticulture Club Sales Assistant.

Base test strategies used by End-Game VV&T are refined before being applied to the Horticulture Club Sales Assistant. Refined test strategies are able to identify violations of constraints/assumptions within the Horticulture Club Sales Assistant. Unlike the results of Partial VV&T, the results of End-Game VV&T can only indicate that vulnerabilities exist within the Horticulture Club Sales Assistant and cannot identify the origins of the vulnerability. End-Game VV&T is successful in identifying vulnerabilities of the Horticulture Club Sales Assistant that if fixed, can enhance the security of the Horticulture Club Sales Assistant. Both Partial and End-Game VV&T prove to be viable security assessment tools that can identify vulnerabilities within a software system and be used to enhance the security of a software system.

6.2 Contributions

Complexity of software systems is ever expanding and as such, tools used to enhance the security of software systems must also expand. The constraints/assumptions technology is expandable and can accommodate the changing landscape of newly developed software systems. Contributions made by this research transition the constraints/assumptions technology from theoretical to applicable. Contributions of this research to the body of knowledge are:

- ***Application of constraints/assumptions technology to a software system using Partial and End-Game VV&T.*** Prior to this research, the constraints/assumptions technology had not been applied in full to an existing software system. This research explains how to use constraints/assumptions to target software specific issues and identify underlying software vulnerabilities. Application of Partial and End-Game VV&T are documented in this work and can be used as reference when applying Partial or End-Game VV&T to another software system.

- ***Identifying relevant development artifacts of a software system to use in the application of Partial VV&T.*** The application of Partial VV&T involves using artifacts produced during the development of a software system and examining the artifacts for the presence of constraints/assumptions. Not all development artifacts are created equal; some artifacts produced during the development of a software system may not be beneficial to Partial VV&T. This research examines the development artifacts of the Horticulture Club Sales Assistant and identifies which artifacts to use in the application of Partial VV&T. The selection of each artifact is explained and the potential benefit of each artifact to Partial VV&T is discussed. Identifying artifacts to use in Partial VV&T is as important as the application of Partial VV&T itself. The selection process documented in this research can be used to guide the selection of development artifacts for future applications of Partial VV&T.
- ***Constraints/assumptions are translated to be specific to both the software system and development artifact in question when applying Partial VV&T.*** The constraints/assumptions, in their current form, are unlikely to exist in any software development artifact. Constraints/assumptions must be translated to language expected of the development artifact. A constraint/assumption may be represented in a development artifact by a single word or an entire paragraph. Connections must be made from the instances within the development artifact to the constraints/assumptions. This research documents the translation of constraints/assumptions to instances found within development artifacts. Translating the constraints/assumptions is a difficult aspect of applying Partial VV&T and the translations documented in this research can give guidance to future applications of Partial VV&T.
- ***Development of constraint/assumption review table and status nomenclature.*** In applying Partial VV&T, instances of constraints/assumptions within the development artifacts, as well as their status, are recorded. To record this information, a

constraint/assumption review table is developed that can be used for all development artifacts. The review table denotes related instances of constraints/assumptions found within a development artifact along with the status of the development artifact to the given constraint/assumption. This status can be denoted using the status nomenclature that is also developed in this work. The status nomenclature describes all possible status's a development artifact can have with relation to a constraint/assumption. The constraint/assumption review table and status nomenclature are developed to be generic such that they can be applied to any software system and development artifact.

- ***Documented application of Partial VV&T to a software system.*** This research uses the developed constraint/assumption review table and status nomenclature to apply Partial VV&T to a software system. Development artifacts identified to be relevant and beneficial to the application of Partial VV&T are used. In applying Partial VV&T to the various development artifacts, a constraint/assumption review table is created for each artifact. Instances within the constraint/assumption review table represent instances of translated constraints/assumptions found within the development artifact. Constraint/assumption review tables of all examined development artifacts are used to gather results of Partial VV&T. Using these methods, the application of Partial VV&T is successful in identifying security flaws within a system. This application of Partial VV&T is documented so that it may be followed for future applications of Partial VV&T.
- ***Base test strategies used for End-Game VV&T are refined to be more specific to the software system.*** The application of End-Game VV&T is centralized around base strategies developed in previous work. To maximize the effectiveness, these base test strategies can be refined to be more software system specific. This research documents refined base test strategies that are used in the application of End-Game VV&T. The refined test strategies can be used as a reference in refining the base test strategies for any software system.
- ***Application of End-Game VV&T to a software system using refined test strategies.*** The refined test strategies are used in the application of End-Game VV&T to a software system. End-Game VV&T is able to find security flaws within the software system by

testing for constraints/assumptions using the refined test strategies. Methods in applying the refined test strategies are interpreted beyond those described in the base test strategies. Interpretations of applying the refined test strategies to the software system are document in the application of End-Game VV&T. The documented application of End-Game VV&T can be used as a reference for future applications of End-Game VV&T.

6.3 Future Work

Many directions exist in which future work with constraints/assumptions technology can be explored. An extension of this research is to apply Full VV&T to the development of a software system. Application of Full VV&T starts at the beginning of the development lifecycle for a software system. Security requirements relating to constraints/assumptions can be put forth as requirements at the beginning of the development lifecycle of the system. Constraint/assumption based security requirements have been developed by Lee Clagget [71]. Security of a system developed under Full VV&T can be tested to determine if additional security has been added to the system by comparing it to similar software systems.

Additional constraints/assumptions can be added to existing ones to address features of software systems that are not currently covered. For example, constraints/assumptions related to power management by a software system can be beneficial for mobile applications. Also, current constraints/assumptions related to the network interface and data security may not apply to newer protocols such as Bluetooth. The constraints/assumptions are expandable by design and as new features and technologies emerge, the constraints/assumptions can be updated accordingly.

Design of an automated test tool utilizing the constraints/assumptions can be an added asset to the application of End-Game VV&T. Existing automated test tools for software systems use rules in order to identify potential security flaws and vulnerabilities within a system. An automated testing tool using the constraints/assumptions as its testing rules can quickly find violations of constraints/assumptions. This can alleviate the need for some of the manual testing currently being performed during the application of End-Game VV&T.

Bibliography

[1] A. Bazaz, "A framework for deriving verification and validation strategies to assess software security," Virginia Polytechnic Institute & State University, 2006, p. 133.

[2] J. D. Arthur, A. Bazaz, R. E. Nance, and O. Balci, "Mitigating Security Risks in Systems that Support Pervasive Services and Computing: Access-Driven Verification, Validation and Testing," in *Pervasive Services, IEEE International Conference on*, 2007, pp. 109-117.

[3] M. Paul, "Software Security: Being Secure in an Insecure World," http://www.isc2.org/csslp-whitepaper.aspx, 2009.

[4] Internet Security Systems, "Top Attacks for Virginia Tech," 2009.

[5] "Obama Creates 'Cyber Czar' Position to Fight Digital Threats," http://www.foxnews.com/politics/2009/05/29/obama-set-create-cyber-czar-position/, 2009.

[6] B. Brewin, nextgov, "Digital Files on Presidential Helicopter Found in Iran," http://www.nextgov.com/nextgov/ng_20090302_8335.php, 2009.

[7] L. Baldor, Associated Press, "White House among targets of sweeping cyber attack," http://www.usatoday.com/tech/news/computersecurity/2009-07-08-government-cyberattack_N.htm, 2009.

[8] G. McGraw, "Testing for security during development: why we should scrap penetrate-and-patch," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 13, pp. 13-15, 1998.

[9] M. Paul, "The Need For Secure Software," http://www.isc2.org/csslp-whitepaper.aspx, 2009.

[10] G. McGraw, "Software security," *Security & Privacy, IEEE,* vol. 2, pp. 80-83, 2004.

[11] F. Piessens, "A Taxonomy of Causes of Software Vulnerabilities in Internet Software," *Supplementary Proceedings of the 13th International Symposium on Software Reliability Engineering,* pp. 47-52, 2002.

[12] R. C. Seacord and A. D. Householder, "A Structured Approach to Classifying Security Vulnerabilities," *CMU/SEI-2005-TN-003,* 2005`.

[13] R. P. Abbott, J. S. Chin, J. E. Donnelley, W. L. Konigsford, S. Tokubo, and D. A. Webb, "Security Analysis and Enhancements of Computer Operating Systems," *NBSIR 76-1041, Institute for Computer Sciences and Technology, National Bureau of Standards,* April 1976.

[14] R. Bisbey and D. Hollingsworth, "Protection Analysis Project Final Report," *ISI/RR-78-13, DTIC AD A056816, USC/Information Sciences Institute,* May 1978.

[15] M. Bishop and D. Bailey, "A Critical Analysis of Vulnerability Taxonomies," *Technical Report CSE-96-11,* vol. Department of Computer Science an the University of California at Davis, September 1996.

[16] M. Bishop, "A Taxonomy of UNIX System and Network Vulnerabilities," *Technical Report CSE-95-8,* May 1995.

[17] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi, "A taxonomy of computer program security flaws," *ACM Comput. Surv.,* vol. 26, pp. 211-254, 1994.

[18] A. Bazaz, J. D. Arthur, and J. G. Tront, "Modeling Security Vulnerabilities: A Constraints and Assumptions Perspective," in *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on*, 2006, pp. 95-102.

[19] SANS, "The Top Cyber Security Risks," http://www.sans.org/top-cyber-security-risks/, 2009.

[20] R. Shimonski, "Your Quick Guide to Common Attacks," http://www.windowsecurity.com/articles/Common_Attacks.html, 2004.

[21] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner, "State of the practice of intrusion detection technologies," *Technical Report CMU/SEI-99-TR-028,* January, 2000.

[22] G. McGraw, "Automated Code Review Tools for Security," *Computer,* vol. 41, pp. 108-111, 2008.

[23] Hammurapi, http://www.hammurapi.biz/hammurapi-biz/ef/xmenu/hammurapi-group/products/hammurapi/index.html, 2009.

[24] D. P. Gilliam, J. C. Kelly, J. D. Powell, and M. Bishop, "Development of a software security assessment instrument to reduce software security risk," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2001. WET ICE 2001. Proceedings. Tenth IEEE International Workshops on*, 2001, pp. 144-149.

[25] S. Lipner, "The trustworthy computing security development lifecycle," in *Computer Security Applications Conference, 2004. 20th Annual*, 2004, pp. 2-13.

[26] R. S. Pressman, *Software Engineering: A Practitioner's Approach*: McGraw-Hill Higher Education, 2000.

[27] P. Smith, "Waterfall Model," http://en.wikipedia.org/wiki/File:Waterfall_model.svg, 2009.

[28] SoftDevTeam, "Incremental Lifecycle Model," http://www.softdevteam.com/Incremental-lifecycle.asp, 2009.

[29] B. W. Boehm, "A spiral model of software development and enhancement," *Computer,* vol. 21, pp. 61-72, 1988.

[30] C. Walls, *Embedded Software: The Works*: Newnes, 2005.

[31] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol -- HTTP/1.1*: RFC Editor, 1999.

[32] E. Rescorla, *HTTP Over TLS*: RFC Editor, 2000.

[33] T. Dierks and C. Allen, *The TLS Protocol Version 1.0*: RFC Editor, 1999.

[34] H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*: RFC Editor, 1997.

[35] W. Stallings, *Cryptography and Network Security (4th Edition)*: Prentice-Hall, Inc., 2005.

[36] C. L. Sabharwal, "Java, Java, Java," *Potentials, IEEE,* vol. 17, pp. 33-37, 1998.

[37] J. Gosling, B. Joy, G. Steele, and G. Bracha, *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley))*: Addison-Wesley Professional, 2005.

[38] D. Poo, D. Kiong, and S. Ashok, *Object-Oriented Programming and Java*: Springer-Verlag New York, Inc., 2007.

[39] J. Gosling and H. McGilton, "The Java Language Environment," *white paper,* May 1996.

[40] T. Lindholm and F. Yellin, *Java Virtual Machine Specification*: Addison-Wesley Longman Publishing Co., Inc., 1999.

[41] Sun Microsystems, "MySQL 5.4 Reference Manual," http://dev.mysql.com/doc/refman/5.4/en/index.html, 2009.

[42] C. Babcock, "Sun Locks UP MySQL, Looks to Future Web Development," *InformationWeek,* February 26, 2008.

[43] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM,* vol. 26, pp. 64-69, 1983.

[44] Free Software Foundation, "GNU General Public License," http://www.fsf.org/licensing/licenses/gpl.html, 2007.

[45] I. Zoratti, "MYSQL Security Best Practices," in *Crime and Security, 2006. The Institution of Engineering and Technology Conference on*, 2006, pp. 183-198.

[46] P. Kumar, *J2ee^{TM}; security for servlets, ejbs and web services: applying theory and standards to practice*: Prentice Hall Press, 2003.

[47] Sun Microsystems "J2EE 1.4 Glossary," http://java.sun.com/javaee/reference/glossary/index.jsp, 2009.

[48] J. Brittain and I. F. Darwin, *Tomcat: the definitive guide, 2nd edition*: O'Reilly, 2007.

[49] M. Bond and D. Law, *Tomcat kick start*: Sams, 2002.

[50] Sun Microsystems, "Java Servlet Technology," http://java.sun.com/products/servlet/, 2009.

[51] J. Hunter and W. Crawford, *Java Servlet Programming*: O'Reilly & Associates, Inc., 2001.

[52] Sun Microsystems, "Java Servlet Technology Overview," http://java.sun.com/products/servlet/overview.html, 2009.

[53] Apache Tomcat, http://tomcat.apache.org/, 2009.

[54] H. Bergsten, *JavaServer Pages, 3rd Edition*: O'Reilly \&amp; Associates, Inc., 2003.

[55] S. Holzner, *Sams Teach Yourself JavaServer Pages in 21 Days*: Sams, 2002.

[56] P. Smith, "An SVG graphic showing the life of a JSP file," http://en.wikipedia.org/wiki/File:JSPLife.svg, 2006.

[57] D. Williams, *Pro PayPal E-Commerce*: Apress, 2007.

[58] PayPal, "PayPal Security Key," https://www.paypal.com/cgi-bin/webscr?cmd=xpt/Marketing_CommandDriven/securitycenter/PayPalSecurityKey-outside, 2009.

[59] *Building secure software: how to avoid security problems the right way*: Addison-Wesley Longman Publishing Co., Inc., 2002.

[60] Virginia Polytechnic Institute and State University, "Creating Strong Passwords," http://www.computing.vt.edu/accounts_and_access/pickinggoodpasswords.html, 2006.

[61] MySQL Query Browser, http://dev.mysql.com/doc/query-browser/en/index.html, 2009.

[62] Sun Microsystems, "Using Jconsole," http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html, 2009.

[63] Sun Microsystems, "Java VisualVM," http://java.sun.com/javase/6/docs/technotes/guides/visualvm/index.html, 2009.

[64] Sun Microsystems, "Format Strings," http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html#format, 2009.

[65] Sun Microsystems, "Interface ServletRequest," http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/ServletRequest.html, 2009.

[66] NTFS Undelete, http://ntfsundelete.com/, 2009.

[67] WinQuota, http://www.winquota.com/, 2009.

[68] Wireshark, http://www.wireshark.org/, 2009.

[69] Virginia Polytechnic Institute and State University, "PID Creation Process," http://www.computing.vt.edu/accounts_and_access/pid/pidgenprocess.html, 2005.

[70] Microsoft, "Using Windows Firewall," http://www.microsoft.com/windowsxp/using/networking/security/winfirewall.mspx, 2006.

[71] L. Clagget, "Security Requirements for the Prevention of Modern Software Vulnerabilities and a Process for Incorporation into Classic Software Development Life Cycles," 2009.

Appendix A

Table A-1. Constraint/Assumption Review Table for the Horticulture Club Sales Assistant's Product Overview Document.

Product Overview		
Dynamic Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data Accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to the buffer.	Data from previous years as well as current inventory will be stored. Dynamic memory will be incorporated when inputting data, introducing the possibility of buffer overflow.	Related Instance => Constraint/Assumption
2. The process will not interpret data present on the dynamic memory as executable code.	Being a process that allows online ordering, there is great opportunity for executable code to be injected as an input parameter, placing it on the dynamic memory.	Related Instance => Constraint/Assumption
3. Environment variables being used by the process have expected format and values.	Environment variables can be used to pass input into web pages. These should conform to expected formats and value ranges.	Related Instance => Constraint/Assumption
4. The process will be provided with the dynamic memory that it requests.	Dynamic memory will be required for network resources as well as temporary data storage. Referencing non-allocated memory can lead to system failure.	Related Instance => Constraint/Assumption
5. Data present on the dynamic memory cannot be observed while the process is in execution.		N/A to Product Overview
6. Data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.		N/A to Product Overview

Table A-1 Continued…

Dynamic Memory - Constraints/Assumptions - Continued	Related Instance	Constraint/Assumption Status
7. A pointer variable being used by the process references a legal memory location.		N/A to Product Overview
8. A memory pointer returned by the underlying operating system does not point to zero bytes of memory.		N/A to Product Overview
9. A pointer variable being used by the process cannot reference itself.		N/A to Product Overview
10. Data accepted by the process must not be interpreted as a format string by the I/O routines.	Being a process that allows online ordering, the process will have input capability and will need to prevent against format strings.	Related Instance => Constraint/Assumption
11. The value of an integer variable/expression (signed and unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer variable.	The process will implement automated inventory tracking which implies that numerical data will be stored that will require boundary checks.	Related Instance => Constraint/Assumption
12. An integer variable/expression used by the process as the index to a buffer must only hold values that allow it access to the memory locations assigned to the buffer.		N/A to Product Overview
13. An integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values.	The implementation of automated inventory tracking as well as the process being an online ordering system indicate that quantities will be used and need to be protected with boundary checks.	Related Instance => Constraint/Assumption

Table A-1 Continued…

Static Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to buffer on the static memory.		N/A to Product Overview
2. Data held on the static memory cannot be observed while the process is in execution.		N/A to Product Overview
Filesystem - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Access permissions assigned to newly created files/directories are such that only the required principals have access to them.	Sales reports will be generated by the process and will need to be created with the proper access permissions.	Related Instance => Constraint/Assumption
2. Access permissions of the files/directories being used by the process are such that only the required principals have access to them.	Any Data being stored by the process that is deemed to be access sensitive should incorporate appropriate access permissions.	Related Instance => Constraint/Assumption
3. A file being created by the process does not have the same name as an already existing file.	If multiple sales reports are made and stored by the process then the filenames of the sales reports should not be the same.	Related Instance => Constraint/Assumption
4. A filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permissions.		N/A to Product Overview
5. A file created/populated by a principal other than the process and being used by the process will have expected format and data.	The process is utilizing a credit card system. If the credit card system returns a file to the process then it will need to be checked for the proper format.	Related Instance => Constraint/Assumption
6. A file being used by the process cannot be observed/modified/replaced while the process is in execution.	Being a web application, almost all modification of program files will occur while the process is in execution (online).	N/A to Process

Table A-1 Continued…

Filesystem - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
7. A file/directory being used by the process and stored on the file-system (information used by the process over multiple runs) cannot be observed/modified/replaced in-between these runs.	Being a dynamic online web application, files will be updated continually.	N/A to Process
8. Data held by files owned/used by the process must not be accessible after the process deletes them.		N/A to Product Overview
9. The process must be provided with the filesystem space that it requests.	The process should be provided with enough space to hold inventory, sales reports and other data the system needs to store.	Related Instance => Constraint/Assumption
10. A file having proprietary or obscure format cannot be understood or modified.		N/A to Product Overview
Network Interface – Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. The data received by the software process through the network interface is neither read nor modified by anyone other than the intended recipient.	The process will be networked and should be protected from possible man-in-the-middle attacks.	Related Instance => Constraint/Assumption
2. The data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.	Values required as input should be checked for authenticity, format and length.	Related Instance => Constraint/Assumption
3. The data sent by the software process via the network interface will not be read/modified before it reaches its destination.	The process will be networked and should be protected from possible man-in-the-middle attacks.	Related Instance => Constraint/Assumption
4. The software process will be able to utilize the network interface to send and receive data.	The process will need to be able to utilize the network in order to function properly.	Related Instance => Constraint/Assumption

Table A-1 Continued…

Network Interface - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
5. The byte order of numerical data accepted from the network interface is same as that of the host machine.		N/A to Product Overview

Randomness Resources - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. The series of random data being produced by the PRNG is unpredictable (assuming unpredictable seed).		N/A to Product Overview
2. The seed being used by the PRNG is unpredictable.		N/A to Product Overview
3. The process will have easy access to entropic data on a computer system.		N/A to Product Overview
4. The process will be able to accurately estimate entropy of a data set.		N/A to Product Overview
5. User selected passwords/keys will have sufficient amount of entropy.		N/A to Product Overview
6. If two different seeds are provided to the PRNG, it is computationally infeasible to produce the same series of data both times.		N/A to Product Overview
7. Given that the PRNG is continuously producing random data, it is computationally infeasible to produce the same sequence of random data after some time.		N/A to Product Overview

Table A-1 Continued…

Cryptographic Algorithms and Protocols - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Random data being used by the cryptographic algorithm/protocol is unpredictable.		N/A to Product Overview
2. The length of the key being used by the cryptographic algorithm or protocol is sufficient.		N/A to Product Overview
3. The hashing algorithm will not produce same hash for two different inputs.		N/A to Product Overview
4. The process cannot use encryption to ensure data integrity.	The process will implement credit card capability and must ensure the integrity of all monetary transactions.	Related Instance => Constraint/Assumption
5. The process cannot use a key more than once for a stream cipher.		N/A to Product Overview
6. The process cannot use one time pads to encrypt a large quantity of data.		N/A to Product Overview
7. The process cannot use keys that are self reported by a client or a server.		N/A to Product Overview
8. The process cannot use obfuscation instead of encryption to ensure confidentiality.	All monetary transactions by the process must ensure confidentiality.	Related Instance => Constraint/Assumption
9. The process cannot store keys/passwords in clear text.		N/A to Product Overview

Table A-2. Constraint/Assumption Review Table for the Horticulture Club Sales Assistant's Requirements Document.

Requirements Document		
Dynamic Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data Accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to the buffer.	Data Dictionary specifies data types for proposed variables to be used by the process. This also can be used to specify any needed buffer lengths by knowing the data types and their expected use.	Related Instance => Constraint/Assumption
2. The process will not interpret data present on the dynamic memory as executable code.	Being a process that allows online ordering, there is great opportunity for executable code to be injected as an input parameter, placing it on the dynamic memory. The Data Dictionary implies knowledge of expected data that will be stored on the dynamic memory as to guard against all other data.	Related Instance => Constraint/Assumption
3. Environment variables being used by the process have expected format and values.	Environment variables can be used to pass input into web pages. These should conform to expected formats and value ranges.	Related Instance => Constraint/Assumption
4. The process will be provided with the dynamic memory that it requests.	Dynamic memory will be required for network resources as well as temporary data storage. Referencing non-allocated memory can lead to system failure. The process should ensure access to memory before using it.	Related Instance => Constraint/Assumption
5. Data present on the dynamic memory cannot be observed while the process is in execution.	If a hostile entity has access control other than being a client of the server the dynamic memory may be able to be observed during execution.	Constraint/Assumption Not Met
6. Data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.		N/A to Requirements Specification

Table A-2 Continued…

Dynamic Memory - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
7. A pointer variable being used by the process references a legal memory location.		N/A to Requirements Specification
8. A memory pointer returned by the underlying operating system does not point to zero bytes of memory.		N/A to Requirements Specification
9. A pointer variable being used by the process cannot reference itself.		N/A to Requirements Specification
10. Data accepted by the process must not be interpreted as a format string by the I/O routines.	The Data Dictionary specifies all expected variables which will be handled by the I/O routines. Also specified, are the expected format of the variables which can be used to filter format.	Related Instance => Constraint/Assumption
11. The value of an integer variable/expression (signed and unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer variable.	The Data Dictionary specifies the integer variables to be used by the process and can allow for appropriate boundary checks.	Related Instance => Constraint/Assumption
12. An integer variable/expression used by the process as the index to a buffer must only hold values that allow it access to the memory locations assigned to the buffer.	The Data Dictionary specifies expected variables and their purpose, allowing for appropriate index boundary checks to be performed.	Related Instance => Constraint/Assumption
13. An integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values.	The Data Dictionary specifies several variables that are intended to hold either quantity or price. The process should then protect against these variables being non-negative.	Related Instance => Constraint/Assumption

Table A-2 Continued…

Static Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to buffer on the static memory.		N/A to Requirements Specification
2. Data held on the static memory cannot be observed while the process is in execution.		N/A to Requirements Specification
Filesystem - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Access permissions assigned to newly created files/directories are such that only the required principals have access to them.	Authorized users of the process will have delegated privileges that will restrict the access of files to only required principals	Constraint/Assumption Met
2. Access permissions of the files/directories being used by the process are such that only the required principals have access to them.	Authorized users of the process will have delegated privileges that will restrict the access of files to only required principals	Constraint/Assumption Met
3. A file being created by the process does not have the same name as an already existing file.	Image files will be stored for inventory purposes which implies a unique naming convention. There is no mention of retention of sales reports within the process.	Related Instance => Constraint/Assumption
4. A filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permissions.		N/A to Requirements Specification

Table A-2 Continued…

Filesystem - Constraints/Assumptions - Continued	Related Instance	Constraint/Assumption Status
5. A file created/populated by a principal other than the process and being used by the process will have expected format and data.	PayPal, being incorporated into this process can send a receipt or confirmation to the process to confirm payment. This confirmation will have a known format. Picture files should also be checked for known data types.	Related Instance => Constraint/Assumption
6. A file being used by the process cannot be observed/modified/replaced while the process is in execution.	The MySQL database will eliminate race conditions among the database. Being a web application, almost all modification of program files will occur while the process is in execution (online).	N/A to Process
7. A file/directory being used by the process and stored on the file-system (information used by the process over multiple runs) cannot be observed/modified/replaced in-between these runs.	Being a dynamic online web application, files pertaining to items such as inventory as well as the MySQL database will be updated continually.	N/A to Process
8. Data held by files owned/used by the process must not be accessible after the process deletes them.	Data deleted from the database will be handled by MySQL. Image files that are deleted from the filesystem do not introduce a vulnerability if accessed.	N/A to Requirements Specification
9. The process must be provided with the filesystem space that it requests.	Being an interactive web service, the process can be run on a single computational unit, as opposed to being distributed. This allows the process to run in an environment that can accommodate its resource demands.	Related Instance => Constraint/Assumption
10. A file having proprietary or obscure format cannot be understood or modified.		N/A to Requirements Specification

Table A-2 Continued…

Network Interface - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. The data received by the software process through the network interface is neither read nor modified by anyone other than the intended recipient.	Some functionality of the process will inherently require the use of the HTTPS protocol. This will help authenticate users and ensure data integrity. However, there is no indication of HTTPS being used process wide and as such, some of the process may be vulnerable to man in the middle attacks.	Related Instance => Constraint/Assumption
2. The data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.	Part of the process is able to be accessed by the public, meaning there is no access restriction among who can use the public portion of the process. The data dictionary can aid in ensuring data is of the correct format and length.	Related Instance => Constraint/Assumption
3. The data sent by the software process via the network interface will not be read/modified before it reaches its destination.	Some functionality of the process will inherently require the use of the HTTPS protocol. This will help authenticate users and ensure data integrity. However, there is no indication of HTTPS being used process wide and as such, some of the process may be vulnerable to man in the middle attacks.	Related Instance => Constraint/Assumption
4. The software process will be able to utilize the network interface to send and receive data.	The process will implement a network interface for sending and receiving data	Related Instance => Constraint/Assumption
5. The byte order of numerical data accepted from the network interface is same as that of the host machine.		N/A to Requirements Specification

Table A-2 Continued…

Randomness Resources - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
1. The series of random data being produced by the PRNG is unpredictable (assuming unpredictable seed).	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
2. The seed being used by the PRNG is unpredictable.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
3. The process will have easy access to entropic data on a computer system.	The process will use the HTTPS protocol in carrying out certain functionality. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
4. The process will be able to accurately estimate entropy of a data set.	The process will use the HTTPS protocol which will use random numbers. The entropic properties of the random and secure data used should be able to be estimated. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
5. User selected passwords/keys will have sufficient amount of entropy.	The process will require a login of a user name and a password.	Related Instance => Constraint/Assumption
6. If two different seeds are provided to the PRNG, it is computationally infeasible to produce the same series of data both times.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process

Table A-2 Continued…

Randomness Resources - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
7. Given that the PRNG is continuously producing random data, it is computationally infeasible to produce the same sequence of random data after some time.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
Cryptographic Algorithms/Protocols – Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Random data being used by the cryptographic algorithm/protocol is unpredictable.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
2. The length of the key being used by the cryptographic algorithm or protocol is sufficient.	The HTTPS protocol is used by the process and has multiple key lengths that can be used. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
3. The hashing algorithm will not produce same hash for two different inputs.	The HTTPS protocol is used by the process and uses a keyed hash when establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
4. The process cannot use encryption to ensure data integrity.	The HTTPS protocol is used by the process and is useful in the prevention of data tampering. This is done using a hashed message digest and not encryption.	Constraint/Assumption Met

Table A-2 Continued…

Cryptographic Algorithms/Protocols – Constraints/Assumptions - Continued	Related Instance	Constraint/Assumption Status
5. The process cannot use a key more than once for a stream cipher.	The HTTPS protocol uses the TLS protocol. The TLS protocol supports stream ciphers in its cipher suite. Keys must be checked to make sure they are not used more than once.	N/A to Process
6. The process cannot use one time pads to encrypt a large quantity of data.		N/A to Requirements Specification
7. The process cannot use keys that are self reported by a client or a server.	Keys used in the HTTPS protocol are from both the client AND the server.	N/A to Process
8. The process cannot use obfuscation instead of encryption to ensure confidentiality.	The HTTPS protocol is used by the process and provides encryption for data. Encryption provides confidentiality	Constraint/Assumption Met
9. The process cannot store keys/passwords in clear text.	Passwords will be used by the software process.	Related Instance => Constraint/Assumption

Table A-3. Constraint/Assumption Review Table for the Horticulture Club Sales Assistant's Design Document.

Design Document		
Dynamic Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data Accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to the buffer.	Automatic bounds checking by the Java programming language prevents buffer overflow.	Constraint/Assumption Met
2. The process will not interpret data present on the dynamic memory as executable code.	The security layer of the process will parse all data input to remove malicious information such as code injection.	Related Instance => Constraint/Assumption
3. Environment variables being used by the process have expected format and values.	Environment variables can be used to pass input into web pages. However, Java servlets use object to pass input parameters as opposed to CGI scripts which can use environment variables. There is no indication that the process directly accesses environment variables.	N/A to Process
4. The process will be provided with the dynamic memory that it requests.	Java implements an automatic memory management system. This system throws an error when the process is not able to be provided with memory that it requests.	Related Instance => Constraint/Assumption
5. Data present on the dynamic memory cannot be observed while the process is in execution.	If a hostile entity has access control other than being a client of the server the dynamic memory may be able to be observed during execution.	Constraint/Assumption Not Met
6. Data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.	Java implements an automatic memory management system. This includes automatic garbage collection. Garbage collection occurs when reference to a process object no longer exists. To ensure data cannot be accessed, the memory should be overwritten before being freed.	Related Instance => Constraint/Assumption

Table A-3 Continued…

Dynamic Memory - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
7. A pointer variable being used by the process references a legal memory location.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
8. A memory pointer returned by the underlying operating system does not point to zero bytes of memory.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
9. A pointer variable being used by the process cannot reference itself.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
10. Data accepted by the process must not be interpreted as a format string by the I/O routines.	The security layer of the process will parse all data input to remove malicious information such as code injection.	Related Instance => Constraint/Assumption
11. The value of an integer variable/expression (signed and unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer variable.	The Java programming language's automatic boundary checking property will prevent an integer from exceeding the minimum/maximum boundaries.	Constraint/Assumption Met
12. An integer variable/expression used by the process as the index to a buffer must only hold values that allow it access to the memory locations assigned to the buffer.	Java provides index checking for buffers.	Constraint/Assumption Met
13. An integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values.	Functional requirements and use cases indicate the use of quantity when ordering online from the process. This value should hold only non-negative numbers.	Related Instance => Constraint/Assumption

Table A-3 Continued…

Static Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to buffer on the static memory.	This process will not require static memory.	N/A to Design Document
2. Data held on the static memory cannot be observed while the process is in execution.	This process will not require static memory.	N/A to Design Document
Filesystem - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Access permissions assigned to newly created files/directories are such that only the required principals have access to them.	Authorized users of the process will have delegated privileges that will restrict the access of files to only required principals	Constraint/Assumption Met
2. Access permissions of the files/directories being used by the process are such that only the required principals have access to them.	Authorized users of the process will have delegated privileges that will restrict the access of files to only required principals	Constraint/Assumption Met
3. A file being created by the process does not have the same name as an already existing file.	Picture files will be uploaded for inventory purposes.	Related Instance => Constraint/Assumption
4. A filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permissions.	Files being used by the process are expected to be picture and database files. Picture files are uploaded to the process and can be enforced. The database should not affect this constraint/assumption.	Related Instance => Constraint/Assumption

Table A-3 Continued…

Filesystem - Constraints/Assumptions - Continued	Related Instance	Constraint/Assumption Status
5. A file created/populated by a principal other than the process and being used by the process will have expected format and data.	PayPal, being incorporated into this process can send a receipt or confirmation to the process to confirm payment. This confirmation will have a known format. Picture files should also be checked for known data types	Related Instance => Constraint/Assumption
6. A file being used by the process cannot be observed/modified/replaced while the process is in execution.	The MySQL database will eliminate race conditions among the database while inventory image files can be changed during execution as they are not vital to proper functionality of the process. Being a web application, almost all modification of program files will occur while the process is in execution (online).	N/A to Process
7. A file/directory being used by the process and stored on the file-system (information used by the process over multiple runs) cannot be observed/modified/replaced in-between these runs.	Being a dynamic online web application, files pertaining to items such as inventory as well as the MySQL database will be updated continually.	N/A to Process
8. Data held by files owned/used by the process must not be accessible after the process deletes them.	Data deleted from the database will be handled by MySQL. Image files that are deleted from the filesystem do not introduce a vulnerability if accessed.	N/A to Design Document
9. The process must be provided with the filesystem space that it requests.	Being an interactive web service, the process can be run on a single computational unit, as opposed to being distributed. This allows the process to run in an environment that can accommodate its resource demands.	Related Instance => Constraint/Assumption

Table A-3 Continued…

Filesystem - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
10. A file having proprietary or obscure format cannot be understood or modified.	All files should have expected format as described in the document. None of these indicate the use of proprietary or obscure formats by the process.	N/A to Process
Network Interface - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. The data received by the software process through the network interface is neither read nor modified by anyone other than the intended recipient.	The software process uses the HTTPS protocol to secure the private web portion of the system as well as monetary transactions. HTTPS protects data from being read and modified.	Constraint/Assumption Met
2. The data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.	Part of the process is able to be accessed by the public, meaning there is no access restriction among who can use the public portion of the process. The data dictionary can aid in ensuring data is of the correct format and length.	Related Instance => Constraint/Assumption
3. The data sent by the software process via the network interface will not be read/modified before it reaches its destination.	The software process uses the HTTPS protocol to secure the private web portion of the system as well as monetary transactions. HTTPS protects data from being read and modified.	Constraint/Assumption Met
4. The software process will be able to utilize the network interface to send and receive data.	The process will implement a network interface for sending and receiving data	Related Instance => Constraint/Assumption
5. The byte order of numerical data accepted from the network interface is same as that of the host machine.	Java uses network byte order, meaning there is no need for conversion.	Constraint/Assumption Met

Table A-3 Continued…

Randomness Resources - Constraints/Assumptions	Related Instance	Constraint/Assumption Status
1. The series of random data being produced by the PRNG is unpredictable (assuming unpredictable seed).	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
2. The seed being used by the PRNG is unpredictable.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
3. The process will have easy access to entropic data on a computer system.	The process will use the HTTPS protocol in carrying out certain functionality. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
4. The process will be able to accurately estimate entropy of a data set.	The process will use the HTTPS protocol which will use random numbers. The entropic properties of the random and secure data used should be able to be estimated. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
5. User selected passwords/keys will have sufficient amount of entropy.	The user login will utilize a Virginia Tech user ID and password. The entropy of the password must fall under Virginia Tech requirements.	Constraint/Assumption Met
6. If two different seeds are provided to the PRNG, it is computationally infeasible to produce the same series of data both times.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process

Table A-3 Continued…

Randomness Resources - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
7. Given that the PRNG is continuously producing random data, it is computationally infeasible to produce the same sequence of random data after some time.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
Cryptographic Algorithms/Protocols – Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Random data being used by the cryptographic algorithm/protocol is unpredictable.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
2. The length of the key being used by the cryptographic algorithm or protocol is sufficient.	The HTTPS protocol is used by the process and has multiple key lengths that can be used. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
3. The hashing algorithm will not produce same hash for two different inputs.	The HTTPS protocol is used by the process and uses a keyed hash when establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
4. The process cannot use encryption to ensure data integrity.	The HTTPS protocol is used by the process and is useful in the prevention of data tampering. This is done using a hashed message digest and not encryption.	Constraint/Assumption Met

Table A-3 Continued…

Cryptographic Algorithms/Protocols – Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
5. The process cannot use a key more than once for a stream cipher.	The HTTPS protocol uses the TLS protocol. The TLS protocol supports stream ciphers in its cipher suite. Keys must be checked to make sure they are not used more than once.	N/A to Process
6. The process cannot use one time pads to encrypt a large quantity of data.	There is no indication of the use of one time pads by the process	N/A to Process
7. The process cannot use keys that are self reported by a client or a server.	Keys used in the HTTPS protocol are from both the client AND the server.	N/A to Process
8. The process cannot use obfuscation instead of encryption to ensure confidentiality.	The HTTPS protocol is used by the process and provides encryption for data. Encryption provides confidentiality	Constraint/Assumption Met
9. The process cannot store keys/passwords in clear text.	Passwords are used by the software process.	Related Instance => Constraint/Assumption

Table A-4. Constraint/Assumption Review Table for the Horticulture Club Sales Assistant's Integration and Testing Document.

Integration and Testing Document		
Dynamic Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data Accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to the buffer.	Automatic bounds checking by the Java programming language prevents buffer overflow.	Constraint/Assumption Met
2. The process will not interpret data present on the dynamic memory as executable code.	The security layer of the process will parse all data input to remove malicious information such as code injection.	Related Instance => Constraint/Assumption
3. Environment variables being used by the process have expected format and values.	Environment variables can be used to pass input into web pages. However, Java servlets use object to pass input parameters as opposed to CGI scripts which can use environment variables. There is no indication that the process directly accesses environment variables.	N/A to Process
4. The process will be provided with the dynamic memory that it requests.	Java implements an automatic memory management system. This system throws an error when the process is not able to be provided with memory that it requests.	Related Instance => Constraint/Assumption
5. Data present on the dynamic memory cannot be observed while the process is in execution.	If a hostile entity has access control other than being a client of the server the dynamic memory may be able to be observed during execution.	Constraint/Assumption Not Met

Table A-4 Continued…

Dynamic Memory – Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
6. Data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.	Java implements an automatic memory management system. This includes automatic garbage collection. Garbage collection occurs when reference to a process object no longer exists. To ensure data cannot be accessed, the memory should be overwritten before being freed.	Related Instance => Constraint/Assumption
7. A pointer variable being used by the process references a legal memory location.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
8. A memory pointer returned by the underlying operating system does not point to zero bytes of memory.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
9. A pointer variable being used by the process cannot reference itself.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
10. Data accepted by the process must not be interpreted as a format string by the I/O routines.	The security layer of the process will parse all data input to remove malicious information such as code injection. However, the process is noted as lacking input validation.	Related Instance => Constraint/Assumption
11. The value of an integer variable/expression (signed and unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer variable.	The Java programming language's automatic boundary checking property will prevent an integer from exceeding the minimum/maximum boundaries.	Constraint/Assumption Met

Table A-4 Continued…

Dynamic Memory – Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
12. An integer variable/expression used by the process as the index to a buffer must only hold values that allow it access to the memory locations assigned to the buffer.	Java provides index checking for buffers.	Constraint/Assumption Met
13. An integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values.	The process will use quantities for keeping track of inventory. However, form validation is noted as lacking, leading to the possibility of negative quantities to be input to the process.	Related Instance => Constraint/Assumption
Static Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to buffer on the static memory.	The process will not require static memory.	N/A to Integration & Testing Document
2. Data held on the static memory cannot be observed while the process is in execution.	The process will not require static memory.	N/A to Integration & Testing Document
Filesystem - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Access permissions assigned to newly created files/directories are such that only the required principals have access to them.	Authorized users of the process will have delegated privileges that will restrict the access of files to only required principals	Constraint/Assumption Met
2. Access permissions of the files/directories being used by the process are such that only the required principals have access to them.	Authorized users of the process will have delegated privileges that will restrict the access of files to only required principals	Constraint/Assumption Met
3. A file being created by the process does not have the same name as an already existing file.	Files that are uploaded to the process and named by the process use a unique item ID number as their filename.	Constraint/Assumption Met

Table A-4 Continued…

Filesystem - Constraints/Assumptions - Continued	Related Instance	Constraint/Assumption Status
4. A filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permissions.	Files being used by the process are expected to be picture and database files. Picture files are uploaded to the process and can be enforced. The database should not affect this constraint/assumption.	Related Instance => Constraint/Assumption
5. A file created/populated by a principal other than the process and being used by the process will have expected format and data.	PayPal, being incorporated into this process, can send a receipt or confirmation to the process to confirm payment. This confirmation will have a known format. Picture files also have known data types.	Related Instance => Constraint/Assumption
6. A file being used by the process cannot be observed/modified/replaced while the process is in execution.	The MySQL database will eliminate race conditions among the database while inventory image files can be changed during execution as they are not vital to proper functionality of the process. Being a web application, almost all modification of program files will occur while the process is in execution (online).	N/A to Process
7. A file/directory being used by the process and stored on the file-system (information used by the process over multiple runs) cannot be observed/modified/replaced in-between these runs.	Being a dynamic online web application, files pertaining to items such as inventory as well as the MySQL database will be updated continually.	N/A to Process
8. Data held by files owned/used by the process must not be accessible after the process deletes them.	Data deleted from the database will be handled by MySQL. Image files that are deleted from the filesystem do not introduce a vulnerability if accessed.	N/A to Integration & Testing Document

Table A-4 Continued…

Filesystem - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
9. The process must be provided with the filesystem space that it requests.	Being an interactive web service, the process can be run on a single computational unit, as opposed to being distributed. This allows the process to run in an environment that can accommodate its resource demands.	Related Instance => Constraint/Assumption
10. A file having proprietary or obscure format cannot be understood or modified.	There is no indication of the process using proprietary or obscure formats for its files.	N/A to Process
Network Interface - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. The data received by the software process through the network interface is neither read nor modified by anyone other than the intended recipient.	The software process uses the HTTPS protocol to secure the private web portion of the system as well as monetary transactions. HTTPS protects data from being read and modified.	Constraint/Assumption Met
2. The data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.	Part of the process is able to be accessed by the public, meaning there is no access restriction among who can use the public portion of the process. The process is noted as lacking input validation leading to the possibility of incorrect formats or lengths of data to be accepted and used by the process.	Constraint/Assumption Not Met
3. The data sent by the software process via the network interface will not be read/modified before it reaches its destination.	The software process uses the HTTPS protocol to secure the private web portion of the system as well as monetary transactions. HTTPS protects data from being read and modified.	Constraint/Assumption Met
4. The software process will be able to utilize the network interface to send and receive data.	The process uses a network for sending and receiving data	Related Instance => Constraint/Assumption

Table A-4 Continued…

Network Interface - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
5. The byte order of numerical data accepted from the network interface is same as that of the host machine.	Java uses network byte order, meaning there is no need for conversion.	Constraint/Assumption Met
Randomness Resources - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. The series of random data being produced by the PRNG is unpredictable (assuming unpredictable seed).	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
2. The seed being used by the PRNG is unpredictable.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
3. The process will have easy access to entropic data on a computer system.	The process will use the HTTPS protocol in carrying out certain functionality. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
4. The process will be able to accurately estimate entropy of a data set.	The process will use the HTTPS protocol which will use random numbers. The entropic properties of the random and secure data used should be able to be estimated. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process

Table A-4 Continued…

Randomness Resources - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
5. User selected passwords/keys will have sufficient amount of entropy.	The user login will utilize a Virginia Tech user ID and password. The entropy of the password must fall under Virginia Tech requirements	Constraint/Assumption Met
6. If two different seeds are provided to the PRNG, it is computationally infeasible to produce the same series of data both times.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
7. Given that the PRNG is continuously producing random data, it is computationally infeasible to produce the same sequence of random data after some time.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
Cryptographic Algorithms/Protocols – Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Random data being used by the cryptographic algorithm/protocol is unpredictable.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
2. The length of the key being used by the cryptographic algorithm or protocol is sufficient.	The HTTPS protocol is used by the process and has multiple key lengths that can be used. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process

Table A-4 Continued…

Cryptographic Algorithms/Protocols – Constraints/Assumptions - Continued	Related Instance	Constraint/Assumption Status
3. The hashing algorithm will not produce same hash for two different inputs.	The HTTPS protocol is used by the process and uses a keyed hash when establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
4. The process cannot use encryption to ensure data integrity.	The HTTPS protocol is used by the process and is useful in the prevention of data tampering. This is done using a hashed message digest and not encryption.	Constraint/Assumption Met
5. The process cannot use a key more than once for a stream cipher.	The HTTPS protocol uses the TLS protocol. The TLS protocol supports stream ciphers in its cipher suite. Keys must be checked to make sure they are not used more than once.	N/A to Process
6. The process cannot use one time pads to encrypt a large quantity of data.	There is no indication of the use of one time pads by the process	N/A to Process
7. The process cannot use keys that are self reported by a client or a server.	Keys used in the HTTPS protocol are from both the client AND the server.	N/A to Process
8. The process cannot use obfuscation instead of encryption to ensure confidentiality.	The HTTPS protocol is used by the process and provides encryption for data. Encryption provides confidentiality.	Constraint/Assumption Met
9. The process cannot store keys/passwords in clear text.	Passwords are used by the software process.	Related Instance => Constraint/Assumption

Table A-5. Constraint/Assumption Review Table for the Horticulture Club Sales Assistant's Source Code.

Code Review		
Dynamic Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data Accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to the buffer.	String variables are used by the software process. String variables are essentially arrays of characters. While Java does not specify a maximum size for string variables, most arrays are usually limited by the maximum size of an integer that can be used as an index to the array. Automatic bounds checking by the Java programming language prevents buffer overflow.	Constraint/Assumption Met
2. The process will not interpret data present on the dynamic memory as executable code.	Input to the process is not thoroughly checked, allowing for the possibility of such things as code or command injection into variables being stored on dynamic memory. When these variables are used, the data present on the dynamic memory will be interpreted as executable code.	Constraint/Assumption Not Met
3. Environment variables being used by the process have expected format and values.	Environment variables are not explicably accessed in the source code. Also, Java servlets pass input parameters between web pages via objects as opposed to using environment variables as CGI scripts do.	N/A to Process
4. The process will be provided with the dynamic memory that it requests.	Java implements an automatic memory management system. This system throws an error when the process is not able to be provided with memory that it requests. However, the process does not ensure that it has been provided with requested memory before proceeding with execution.	Constraint/Assumption Not Met

Table A-5 Continued…

Dynamic Memory - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
5. Data present on the dynamic memory cannot be observed while the process is in execution.	If a hostile entity has access control other than being a client of the server, the dynamic memory may be able to be observed during execution.	Constraint/Assumption Not Met
6. Data owned by the process and stored on the dynamic memory cannot be accessed after the process frees the memory.	Java implements an automatic memory management system. This includes automatic garbage collection. Garbage collection occurs when reference to a process object no longer exists. To ensure data cannot be accessed, the memory should be overwritten before being freed. There is no evidence of overwriting memory before it is eligible for garbage collection.	Constraint/Assumption Not Met
7. A pointer variable being used by the process references a legal memory location.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
8. A memory pointer returned by the underlying operating system does not point to zero bytes of memory.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
9. A pointer variable being used by the process cannot reference itself.	The process is implemented using the Java programming language. The Java programming language does not use pointers.	N/A to Process
10. Data accepted by the process must not be interpreted as a format string by the I/O routines.	Methods used for input and output by the system do not allow the use of format strings	Constraint/Assumption Met

Table A-5 Continued…

Dynamic Memory - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
11. The value of an integer variable/expression (signed and unsigned) accepted/calculated by the process cannot be greater (less) than the maximum (minimum) value that can be stored in the integer variable.	The Java programming language's automatic boundary checking property will prevent an integer from exceeding the minimum/maximum boundaries.	Constraint/Assumption Met
12. An integer variable/expression used by the process as the index to a buffer must only hold values that allow it access to the memory locations assigned to the buffer.	Java provides index checking for buffers. All iteration of buffers in the source code is kept within the bounds of the buffers.	Constraint/Assumption Met
13. An integer variable/expression used by the process to indicate length/quantity of any object must not hold negative values.	There do not exist any non-negativity checks for integers being used to indicate quantity within the source code	Constraint/Assumption Not Met
Static Memory - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Data accepted as input by the process and assigned to a buffer must occupy and modify only specific locations allocated to buffer on the static memory.	The process source code does not utilize static memory.	N/A to Process
2. Data held on the static memory cannot be observed while the process is in execution.	The process source code does not utilize static memory.	N/A to Process
Filesystem - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Access permissions assigned to newly created files/directories are such that only the required principals have access to them.	Permissions are delegated to the Horticlub Member Users of the system. These permissions grant access to only required principals	Constraint/Assumption Met
2. Access permissions of the files/directories being used by the process are such that only the required principals have access to them.	Permissions are delegated to the Horticlub Member Users of the system. These permissions grant access to only required principals	Constraint/Assumption Met

Table A-5 Continued…

Filesystem - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
3. A file being created by the process does not have the same name as an already existing file.	Files that are uploaded to the process and named by the process use a unique item ID number as their filename.	Constraint/Assumption Met
4. A filename (including path) being used by the process is not a link that points to another file for which the user, executing the process, does not have the required access permissions.	There is no check on picture files being uploaded to the system. This can possibly lead to executable files being loaded and executed with root or administrative privileges.	Constraint/Assumption Not Met
5. A file created/populated by a principal other than the process and being used by the process will have expected format and data.	Image files being used for inventory purposes by the process are not checked for format. Files of any extension can be uploaded to the system and stored when only image files are expected. The uploading of files is protected via authorization.	Constraint/Assumption Not Met
6. A file being used by the process cannot be observed/modified/replaced while the process is in execution.	The MySQL database will eliminate race conditions among the database while inventory image files can be changed during execution as they are not vital to proper functionality of the system. Being a web application, almost all modification of program files will occur while the process is in execution (online).	N/A to Process
7. A file/directory being used by the process and stored on the file-system (information used by the process over multiple runs) cannot be observed/modified/replaced in-between these runs.	Being a dynamic online web application, files pertaining to items such as inventory as well as the MySQL database will be updated continually.	N/A to Process

Table A-5 Continued…

Filesystem - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
8. Data held by files owned/used by the process must not be accessible after the process deletes them.	Data is deleted directly by the process, bypassing any trash or recycle folders. Data memory is then eligible to be overwritten. However, data deleted is not zeroed out by the process making recovery possible.	Constraint/Assumption Not Met
9. The process must be provided with the filesystem space that it requests.	There is no check to determine if memory is available within the filesystem before trying to write to it. However, the Java programming language will throw an exception if no memory is available and prevent a full system crash.	Constraint/Assumption Not Met
10. A file having proprietary or obscure format cannot be understood or modified.	No files being used by the process have proprietary or obscure format.	N/A to Process
Network Interface - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. The data received by the software process through the network interface is neither read nor modified by anyone other than the intended recipient.	The software process uses the HTTPS protocol to secure the private web portion of the system as well as monetary transactions. HTTPS protects data from being read and modified.	Constraint/Assumption Met
2. The data received by the software process through the network interface is from a legitimate client or peer or server and has expected format and length.	Formats and lengths of data received by the process are not thoroughly checked before being accepted. Unexpected formats and lengths of data can lead to unexpected behavior of the software process.	Constraint/Assumption Not Met

Table A-5 Continued…

Network Interface - Constraints/Assumptions - Continued	**Related Instance**	**Constraint/Assumption Status**
3. The data sent by the software process via the network interface will not be read/modified before it reaches its destination.	The software process uses the HTTPS protocol to secure the private web portion of the system as well as monetary transactions. HTTPS protects data from being read and modified.	Constraint/Assumption Met
4. The software process will be able to utilize the network interface to send and receive data.	The software process is a networked web application running on a server implying that the network interface will be available.	Related Instance => Constraint/Assumption
	Database connections are not checked after they are requested in order to see if they exist.	Constraint/Assumption Not Met
	Database connections are not always closed which could lead to an exhaustion of network resources.	Constraint/Assumption Not Met
5. The byte order of numerical data accepted from the network interface is same as that of the host machine.	Java uses network byte order, meaning there is no need for conversion.	Constraint/Assumption Met
Randomness Resources - Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. The series of random data being produced by the PRNG is unpredictable (assuming unpredictable seed).	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
2. The seed being used by the PRNG is unpredictable.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process

Table A-5 Continued…

Randomness Resources - Constraints/Assumptions - Continued	Related Instance	Constraint/Assumption Status
3. The process will have easy access to entropic data on a computer system.	The process will use the HTTPS protocol in carrying out certain functionality. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
4. The process will be able to accurately estimate entropy of a data set.	The process will use the HTTPS protocol which will use random numbers. The entropic properties of the random and secure data used should be able to be estimated. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
5. User selected passwords/keys will have sufficient amount of entropy.	The user login will utilize a Virginia Tech user ID and password. The entropy of the password must fall under Virginia Tech requirements	Constraint/Assumption Met
6. If two different seeds are provided to the PRNG, it is computationally infeasible to produce the same series of data both times.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
7. Given that the PRNG is continuously producing random data, it is computationally infeasible to produce the same sequence of random data after some time.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process

Table A-5 Continued…

Cryptographic Algorithms/Protocols – Constraints/Assumptions	**Related Instance**	**Constraint/Assumption Status**
1. Random data being used by the cryptographic algorithm/protocol is unpredictable.	The HTTPS protocol is used by the process and uses random numbers while establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
2. The length of the key being used by the cryptographic algorithm or protocol is sufficient.	The HTTPS protocol is used by the process and has multiple key lengths that can be used. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
3. The hashing algorithm will not produce same hash for two different inputs.	The HTTPS protocol is used by the process and uses a keyed hash when establishing a connection. Major web browsers provide support and/or implementation of the HTTPS protocol.	N/A to Process
4. The process cannot use encryption to ensure data integrity.	The HTTPS protocol is used by the process and is useful in the prevention of data tampering. This is done using a hashed message digest and not encryption.	Constraint/Assumption Met
5. The process cannot use a key more than once for a stream cipher.	The HTTPS protocol uses the TLS protocol. The TLS protocol supports stream ciphers in its cipher suite. Keys must be checked to make sure they are not used more than once.	N/A to Process
6. The process cannot use one time pads to encrypt a large quantity of data.	The software process does not use one time pads.	N/A to Process
7. The process cannot use keys that are self reported by a client or a server.	Keys used in the HTTPS protocol are from both the client AND the server.	N/A to Process

Table A-5 Continued…

Cryptographic Algorithms/Protocols – Constraints/Assumptions	Related Instance	Constraint/Assumption Status
8. The process cannot use obfuscation instead of encryption to ensure confidentiality.	The HTTPS protocol is used by the process and provides encryption for data. Encryption provides confidentiality	Constraint/Assumption Met
9. The process cannot store keys/passwords in clear text.	Passwords are not stored by the software process.	Constraint/Assumption Met