Three major categories of software

Technology, Computer

Software can be divided into three major categories according to popularity: application software, system software, and web applications. Within each category there are dozens, if not hundreds, of specialized software types, but for the purpose of this study, we will concentrate on the most popular software type of each category.

Software applications refer to programs on a client machine which are written to perform specific tasks. Nowadays, there is a wide range of software applications being developed including word processing programs, database management tools, photo editing software, etc. But during the last decade the web has become the new deployment environment for software applications. Software applications that were previously built for specific operating systems and devices are now being designed specifically for the web (web-enabled). Because of this new movement, and as the web becomes increasingly a universal interface for software development, the software industry is experiencing a major evolution toward web-related software applications (Festa 2001). For example, the recent release of Google’s Chrome web browser which was specifically designed to enable the execution of web applications and services in the web browser confirms this trend.

As the web evolves, the surrounding and supporting technologies are becoming more complex. This is especially relevant in web-enabled applications such as web browsers, email/news clients, VoIP and chat clients which allow the interaction with the web from the client side. Web browsers specifically have become the doorway to the Internet and are currently the most widely used applications and the standard tool for consuming Internet

services. This evolution toward web-related applications had a direct impact on the security of such applications. For instance, vulnerabilities and attacks against web browsers became more popular as such attacks compromise the security and privacy and have serious implications for web users. Once a web-related software is infected, the user's web interaction can be fully exposed to the attacker. For instance, an infected web browser can expose the victim's web addresses, data typed into forms, user sessions and cookies. Moreover, vulnerability risks in a web browser can have a serious implication for intranets (Anupam and Mayer 1998). Most users use the same browser to access information on the intranet as well as the Internet. A user who has been attacked through vulnerable web browser has compromised his or her firewall for the duration of the browsing session (c). Examples of such vulnerability risks against web browsers include key loggers. Key loggers are a form of spyware which can be installed through vulnerability in a web browser and then logs all pressed keys whenever a user visits a certain online banking web site.

The increase in vulnerability risks in web-related software is related to the exponential growth of the Internet. As we enter through the second decade of the 21st century, the rapid adoption of the Internet market along its ubiquitous presence will continue to make Internet technologies such as web-related applications a prime target for attackers as they constitute the largest mass of victims.

Hypothesis 1a: Vulnerability type will be highly positively related to web-related software applications

Hypothesis 1b: Frequency of vulnerability will be highly positively related to web-related software applications

Hypothesis 1c: Severity of vulnerability will be highly positively related to web-related software applications

System Software:

System software refers to the set of computer programs which are required to support the execution of application programs and maintain system hardware. Operating systems, utilities, drivers and compilers are among the major components of system software. Such components are the enablers and service providers to software applications. Among these components, the operating system is the most popular and important one. The operating system market for client PCs has evolved along the lines predicted by theories of increasing returns and network externalities (Shapiro and Varian 1999). But with this increase in network externality, there has been a dramatic increase in vulnerabilities (cite xxx). For instance, between 2007 and 2009, the number of operating system vulnerabilities almost doubled from 220 to 420 vulnerabilities (CVE 2010). Such increase in vulnerabilities can be caused by several reasons. First, network externality implies larger user base which makes operating systems an attractive target for hackers. In addition to that, viruses and worms can spread more rapidly because of the large installed user base and network effect. Second, the architecture of some operating systems like Windows allows vulnerabilities to gain a direct access to the operating system files through external scripts; meaning that if malicious scripts are sophisticated enough, they can exploit system files

through software applications or through system software directly. And last, the fame factor for discovering vulnerabilities in systems with significant installed user base make them potentially significant target for hackers.

Lately, new technologies such as web-based cloud computing, virtualization and Just enough Operating System (JeOS) have been gradually diminishing the importance of the traditional operating system (Geer 2009). With cloud computing technologies, users can access web applications through their web browser; meaning that an OS like Google Chrome will only be needed to run the web-browser. Moreover, with virtualization technology a personal computer or a server is capable of running multiple operating systems or multiple sessions of a single OS at anytime without having the user rely on a single OS. Similarly, Just enough Operating System (JeOS) focuses on running applications which require minimal OS. As these technologies are gaining popularity and becoming more adopted by users, the role of an OS is starting to decrease so does its network externality. With this is mind, we hypothesize that attackers interests and vulnerability risks will gradually shift to other technologies as they become more popular.

Hypothesis 1d: Vulnerability type will be positively related to system software

Hypothesis 1e: Frequency of vulnerability will be positively related to system software

Hypothesis 1f: Severity of vulnerability will be positively related to system software

Web Applications:

The remarkable reach of web applications into all areas of the Internet makes this field among the largest and most important parts of the software industry. As of today, the Internet consists of hundreds of thousands of small and large-scale web applications ranging from e-Commerce applications to social networking sites to online gaming. This popularity has attracted large user base which made web applications lucrative targets for attackers to exploit vulnerabilities. Web applications are currently subject to a plenty of vulnerabilities and attacks, such as cross-site scripting (XSS), session riding (CSRF) and browser hijacking (Mansfield-Devine 2008). Hence, the landscape of vulnerabilities has changed significantly during the first decade of the 21st century. Previously, buffer overflow and format string vulnerabilities accounted for a large fraction of all vulnerabilities during the 1990s, but as web applications became more popular, new vulnerabilities and attacks such as SQL injections and XSS attacks exceeded earlier vulnerabilities. According to CVE surveys, security issues in web applications are the most commonly reported vulnerabilities nowadays. In response, web application vendors dedicated more resources towards securing their products as they tend to receive more attention as potential targets because of their large pool of possible victims (Mercuri 2003). The problem of web application vulnerabilities is becoming more complicated with the recent movement towards Web 2. 0 technologies. The landscape of Web 2. 0 enables new avenue of vulnerabilities by using sophisticated scripts on the client side. Moreover, Web 2. 0 websites are becoming riskier than traditional websites

because they use more scripting capabilities to allow users to upload content, share information and gain more control.

Despite the growth of web applications and Web 2. 0, these technologies are still limited by the available resources such as network bandwidth, latency, memory and processing power. More specifically, it’s argued that web applications are constrained by the capabilities of the web browser they are running in. With this drawback, web application users will ultimately have to rely on their own resources to accomplish magnitudes of tasks. Compared to system and software applications, we hypothesize that web applications will continue to experience vulnerability risks but at a lower rate than other popular software.

Hypothesis 1g: Vulnerability type will be least positively related to web applications

Hypothesis 1h: Frequency of vulnerability will be least positively related to web applications

Hypothesis 1i: Severity of vulnerability will be least positively related to web applications

Targeted Operating System

Software producers often create applications to run on a single or a combination of operating systems (OS). From a software viewpoint, maintaining security is the obligation of both the OS and the software program. But since computer hardware such as the CPU, memory and

input/output channels are accessible to a software programs only by making calls to the OS, therefore, the OS bears a tremendous burden in achieving system security by allocating, controlling and supervising all system resources.

For the most part, each of today's streamlines OSs has a main weakness. For instance, earlier OSs such as Windows NT, UNIX and Macintosh had a weakness in their access control policies (Krsul 1998). Such OSs didn't specify access control policies very clearly which meant that applications that ran by users inherited all the privileges that the access control mechanisms of the OS provided to those users (Wurster 2010). An access control policy requires an OS to give a program or a user the minimum set of access rights necessary to perform a task. In his work, Denning (1983) illustrated the working of an access control policy which typically consists of three entities namely, subjects, objects and access rights matrix. Subjects refer to users or domains whereas objects are files, services, or other resources and access rights matrix specifies different kinds of privileges including read, write and execute which are assigned to subjects over objects. A configuration of the access matrix describes what subjects are authorized to do. Vulnerabilities in OSs tend to rely on weaknesses in configuration of access control matrices to gain access to software applications and system software. This creates a serious problem since vulnerabilities can exploit software applications through the OS gain access and ultimately take over the system. An example of an access control policy failure is Java virtual application. The Java virtual machine was among the applications which defined, and enforced its own access control matrix. Its

sandbox was compromised of a number of OS components which ensured that a malicious application cannot gain access to system resources. But once the access control mechanism of the virtual machine fails, a malicious applet can be given access beyond the sandbox (McGraw and Felten 1997). Meaning that the OS can allow a malicious applet full access to the users files because to the OS there is no difference between the virtual machine and the applet.

Moreover, even with an access control policy in place, consideration must be given to system design. The OSs which are in use today have different architectures and are designed with different kernels without considering security and controlled accessibility as significant design criteria. For instance, a large portion of UNIX and Linux vulnerabilities result from boundary condition errors which are commonly known as buffer overflow (cite xxx). These boundary conditions result from a failure to properly check the bound sizes of buffers, arrays, strings. Attackers tend to exploit this weakness in UNIX and Linux OSs to gain access to system software and software applications. On the other hand, vulnerabilities in Windows OS tend to be evenly divided among exceptional conditions, boundary conditions and access control validations (cite xxx). With these types of vulnerabilities root break-in and execution of arbitrary code are common types of attacks.

When it comes to writing software for different platforms, programmers must acknowledge the potential vulnerabilities and threats targeting their software. Since different OSs have different vulnerabilities, the task of

designing a secure application tend to become much difficult since they have to consider vulnerability risks of each OS. Therefore we hypothesize that:

Hypothesis 5a: Vulnerability type will be positively related to software which target more operating systems

Hypothesis 5b: Frequency of vulnerability will be positively related to software which target more operating systems

Hypothesis 5c: Severity of vulnerability will be positively related to software which target more operating systems

Software Free Trial

Free trial strategy is used by many vendors to promote and sell their goods. This strategy is especially popular and found to be effective to promote and sell digital goods such as software and music. Unlike physical goods, the intangibility of digital products prevents consumers from assessing the products before the consumption and adoption (Heiman and Muller 1996). Such uncertainty of product functionality reduces consumers' motivation to adopt the product and is considered a source of market failure. Nowadays, offering software free trial at a low marginal production cost has resulted in the prevalence of free trials strategy.

For the software market, there are two strategies of free trial, namely a fully functional free version with limited trial period (time locked version) and a limited functional version (demo version). Each of these strategies has its own advantages and disadvantages. For instance a demo version has an

advantage of capturing the network effect from both trial users and the buyers. In contrast, some consumers may find it adequate to use only the limited functionalities provided in the demo version rather than purchasing the full version software. Similarly, offering time locked software version can negatively affect the software vendor as consumers with limited usage can utilize this short-term to fully take advantage of the free trial without buying the full software product.

Based on these trial strategies, there have been numerous studies regarding the effect of free trial on software learning curve (Heiman and Muller 1996), software piracy (Chellappa and Shivendu 2005) and software performance (Lee and Tan 2007). For this study, we are interested in measuring the effect of free trial strategies on software vulnerabilities. Although software vendors often release demo or time locked versions, such versions can still contain good source of information for the attackers. Attackers typically misuse the trial versions to look for, find and exploit vulnerabilities. Furthermore, attackers can reverse engineer the limited code and find vulnerabilities (Sutherland et al. 2006). This technique has become particularly important as the attacker can apply vulnerabilities found in free trial versions to exploit full version software. Moreover, there are many hacker groups on the internet who specialize in cracking free trial and full versions software and releasing them on the internet under what is known as warez. Such groups usually compete with one another to be the first to crack and release the new software. These cracked versions (warez) can also serve as potential targets for attackers looking for vulnerabilities. Hence, while providing free trial versions of software by software vendors is a marketing strategy,

vendors should also expect such free versions can become targets for vulnerabilities and early exploits.

Hypothesis 6a: Vulnerability type will be positively related to software which offer trial versions

Hypothesis 6b: Frequency of vulnerability will be positively to software which offer trial versions

Hypothesis 6c: Severity of vulnerability will be positively to software which offer trial versions

Software License

The diversity of the software business model drives the need for different types of software licenses. A software license is a legal agreement forming a binding contract (relationship) between the vendor and the user of a software product and it’s considered an essential part in the evolution of the software to a market product. Software license is regarded as one of the fundamentals of OSS as there are currently close to 73 different licenses (Perens 2009). Most OSS licenses are classified based on the restrictions they impose on any derivative work (Lerner and Tirole, 2005). Examples of OSS licenses include GPL, LGPL and BSD. General Public License (GPL) is currently the most popular OSS license which states that any derived work from other GPL software has to be distributed under the same licensing terms. The Lesser GPL (LGPL) and the Berkeley Software Distribution (BSD) are other popular alternatives to GPL with similar characteristics.

OSS projects rely heavily on code reuse as shown by DrDobbs (2009). In their work, 1311 OSS projects were analyzed and 365000 instances were found of code reuse among those projects. In principle, most of the OSS licenses allow programmers to modify and reuse existing code. This degree of code inheritance can have positive and negative effects on the security of the software. In their work, Brown and Booch (2002) discussed how reuse of OSS code can inherent insecurities and talked about the concerns which companies have regarding OSS code and how it was developed and in particular the origins and the reuse of its code. Indeed an analysis study by Pham et al. (2010) suggested that one of the key causes of vulnerabilities is due to software reuse in code, algorithms/standards, or shared libraries/APIs. They proposed the use of new model which uses algorithm to map similar vulnerable code across different systems, and use the model to trace and report vulnerabilities to software vendors. Reuse of OSS software has caused concerns as developers might inherent vulnerabilities from existing code but regardless of the open source community or software vendors' positions on this debate, the possibility of security issues by reusing OSS code has been sufficient to the point where some vendors stopped reusing OSS code in their software. From a security perspective and when it comes to reusing OSS, vendors tend to follow one of the following approaches. Abandon OSS software; only reuse code which has been extensively reviewed; or maintain a relationship with the OSS community and get involved with the development process (Brown and Booch 2002).

It's our belief that licenses which allow developers to reuse source code will be more susceptible to vulnerabilities than closed source proprietary

licenses. Meaning that software licenses which allow code reuse are more likely to inherit or contaminate derivate work. In contrast, commercial licenses which don’t share or allow code reuse are less susceptible to inherit or contaminate vulnerabilities.

Hypothesis 4a: Vulnerability type will be positively related to open source software licenses

Hypothesis 4b: Frequency of vulnerability will be positively related to open source licenses

Hypothesis 4c: Severity of vulnerability will be positively related to open source licenses

Source Code Availability

Security of open source software (OSS) and closed source software has been a hot topic with many arguments repeatedly presented. Advocates of OSS argue that more reviewers strengthen the security of the software as it eases the process of finding bugs and speeds it up “ given enough eyeballs, all bugs are shallow” (Raymond and Young 2000). Opponents of this idea disagree and claim that not all code reviewers and testers have enough skills and experience compared to code reviewers at companies who are more skilled at finding flaws. The argument is that oftentimes code reviewers and testers need to have further skills other than programming such as cryptography, stenography and networking. Moreover, proponents of closed source software claim that security by obscurity is the main strength of closed source software since it’s harder to find vulnerabilities when the code

is not accessible. However, proponents of OSS argue that it’s possible to gain access to closed source code through publicly available patches and disassembling software (Tevis 2005).

It’s important to note that the impact of the availability of source code on security depends on the open source development model. For instance, the open source cathedral model allows everyone to view the source code, detect flaws/bugs/vulnerabilities and open reports; but they are not permitted to release patches unless they are approved by project owners. OSS projects are typically regulated by project administrators who require some time to review and approve patches. Attackers can take advantage of the availability of source code and published vulnerability reports to exploit them (Payne 2002). However, proponents of OSS argue that vulnerabilities in OSS projects can be fixed faster than those in closed source software because the OSS community is not dependent on a company’s schedule to release a patch.

Despite the continuous debate on OSS security, advocates from both sides agree that having access to the source code makes it easier to find vulnerabilities but they differ about the impact of vulnerabilities on software security. First of all, keeping the source code open provides attackers with easy access to information that may be helpful to successfully launch an attack. Publically available source code gives attackers the ability to search for vulnerabilities and flaws and thus increase the exposure of the system. Second, making the source code publicly available doesn’t guarantee that a qualified person will look at the source and evaluate it. In the bazaar style

environment, malicious code such as backdoors may be sneaked into the source by attackers posing as trustful contributors. For instance, in 2003 Linux kernel developers discovered an attempt to include a backdoor in the kernel code (Poulsen 2003). Finally, for many OSS projects there is no a priori selection of programmers based on their skills; project owners tend to accept any help without checking for qualifications or coding skills.

Given the issues surrounding source code availability in OSS, we hypothesize that making source code publically available will induce attackers and increase vulnerability risks.

Hypothesis 1a: Vulnerability type will be positively related to source code availability

Hypothesis 1b: Frequency of vulnerability will be positively related to source code availability

Hypothesis 1c: Severity of vulnerability will be positively related to source code availability

Software Programming Language

Selecting a suitable programming language is one of the most important decisions which have to be made during software planning and design. A chosen programming language has direct effect on how software ought to be created and what means must be used to guarantee that the software functions properly and securely. Software programs which are written using an insecure language may cause system dependent errors which are known

to be difficult to find and fix (Hoare 1973). For example, buffer overflows vulnerabilities and other low-level errors are well known issues in C and C++ languages (Cowan 1999).

As of today, there exist numerous programming languages but the topic of security in programming languages has been widely disregarded as it's believed that programming errors and flaws should be eliminated by the programmers themselves. Current approaches to this issue are essentially ad hoc where best programming practices and secure programming techniques are implemented during or after the design stage. Although this approach helps in preventing coding errors and flaws by relying on programmers' skills and experience, it is difficult to say with any certainty what vulnerabilities are prevented and to what extent. More importantly, the ad hoc approach doesn't protect against new and evolving vulnerabilities as it only handles known vulnerabilities and specific coding flaws.

In his paper, Hoare (1974) stated that a programming language is secure only if the compiler and run time support are capable of detecting flaws and violations of the language rules. The main issue with this statement is that current compilers and debugging tools are not reliable since they parse code differently; therefore, it's impossible to guarantee the same results for programs. Additionally, such tools don't help the programmer in finding vulnerabilities or flaws as they only report syntax errors. Typically, compilers and debugging tools don't allow for security checks on debugging runs, therefore no trust can be put in the results.

An evolving trend in secure programming has been the use of formal language semantics. Formal language semantics try to reason with and prove security properties of the code. For example, in their paper, Leroy and Rouaix (1998) developed a formal technique to validate a typed functional language to ensure that memory locations always contain appropriate values to avoid buffer overflow vulnerabilities. Although the use of formal language semantics has been advocated (Dean et al. 1996, Meseguer and Talcott 1997, Volpano 1997), it wasn't widely adopted among programmers.

When it comes to software languages, security is essentially dependent on numerous factors such as language developers, programmers and debugging tools. With so many factors, we believe that correlating software language with vulnerability risks will be insignificant.

Hypothesis 2a: Vulnerability type will be insignificantly correlated with software language

Hypothesis 2b: Frequency of vulnerability will be insignificantly correlated with software language

Hypothesis 2c: Severity of vulnerability will be insignificantly correlated with software language

Targeted Software Users

There are many different types of computer users with a wide range of background, skills, and learning habits. Computer users are typically classified into two distinct groups, namely sophisticated and novice

(unsophisticated) users. Sophisticated users have an advanced understanding of computer and Internet technologies; they tend to be more security-aware. Novice users refer to non-technical personnel who are not experienced with computers and the Internet; they rely on computers for simple tasks such as word-processing, spreadsheets, and occasional web surfing. Such users are more prone to security issues due to their inexperience. For instance ignoring software updates and security patches, failing to run essential protection utilities such as an anti-virus or firewall applications are typical security issues with novice users. Because of differences in experience level between both groups, some argue that vulnerabilities affect novice users more than sophisticated ones. Although this might be true for viruses and worms and old vulnerabilities, but when it comes to dealing with zero-day vulnerabilities everyone becomes a victim regardless of their sophistication level. Zero day vulnerabilities refer to unreported exploitable vulnerabilities for which a patch is not available from software vendors (cite xxx). Moreover, when it comes to attackers and potential targets, eventually everyone is a target. Despite the type of computer users, the objective of vulnerability attacks is to hack as many computers as possible with the least amount of effort (Spitzer 2002). Attackers tend to focus on a single vulnerability and use automated scanning tools to search for as many systems as possible for that vulnerability. Such automated tools are often called autorooters and can be designed to scan a specific network for vulnerable machines or scan a range of IP addresses until a victim is found. It’s important to note that these tools don’t

distinguish between software users as they look for any vulnerable target in sight.

Hypothesis 8a: Vulnerability type will be insignificantly correlated with to targeted software users

Hypothesis 8b: Frequency of vulnerability will be insignificantly correlated with to targeted software users

Hypothesis 8c: Severity of vulnerability will be insignificantly correlated with to targeted software users

Software Price

Software price plays an important role in modifying the individual's attitude toward the software in many ways. For example, research studies which looked at software piracy found that software price to be a significant factor (incentive) which influenced the intention to pirate (Gopal and Sanders 2000). In their work, Peace et al. (2003) conducted a survey of 201 respondents and found that software price was among the major reasons for illegally copying software. Following the same analogy, studies have shown that attackers' attitudes and hackers' motivations for finding vulnerabilities are associated with several factors such as: peer approval, self esteem, politics, publicity, financial gains, curiosity and sabotage (Shaw et al. 1999).

Within the hackers' community, hacking achievements typically help individuals gain higher and more respectable status as it refers to the persons' skills and mastery level. Reaching a higher status is oftentimes

associated with noteworthy achievements such as hacking popular software. For those hackers who seek publicity or peer approval, they tend to target software with large user base due to their significant reach. So despite software price, hackers look for vulnerabilities in open source and proprietary software as long as there is a significant user base. Similarly, infamous social networking sites such as Facebook and Myspace are constant vulnerability targets regardless of their service cost. Outside the hacker's community, hackers' incentives tend to vary among political reasons (example: Google-China Hacking 2010), financial gains (example: ransom money attacks), self esteem and sabotage. Again, by analyzing each incentive, we find that software price doesn't play any role in vulnerability risks. We therefore hypothesize that:

Hypothesis 7a: Vulnerability type will be insignificantly correlated with to software price

Hypothesis 7b: Frequency of vulnerability will be insignificantly correlated with to software price

Hypothesis 7c: Severity of vulnerability will be insignificantly correlated with to software price