On the Effectiveness of Software Diversity: A Systematic Study on Real-World Vulnerabilities

Jin Han, Debin Gao, and Robert H. Deng

School of Information Systems, Singapore Management University
{jin.han.2007,dbgao,robertdeng}@smu.edu.sg

Abstract. Many systems have been introduced to detect software intrusions by comparing the outputs and behavior of diverse replicas when they are processing the same, potentially malicious, input. When these replicas are constructed using off-the-shelf software products, it is assumed that they are diverse and not compromised simultaneously under the same attack. In this paper, we analyze vulnerabilities published in 2007 to evaluate the extent to which this assumption is valid. We focus on vulnerabilities in application software, and show that the majority of these software products – including those providing the same service (and therefore multiple software substitutes can be used in a replicated system to detect intrusions) and those that run on multiple operating systems (and therefore the same software can be used in a replicated system with different operating systems to detect intrusions) – either do not have the same vulnerability or cannot be compromised with the same exploit. We also find evidence that indicates the use of diversity in increasing attack tolerance for other software. These results show that systems utilizing off-the-shelf software products to introduce diversity are effective in detecting intrusions.

1 Introduction

Software diversity has many advantages over mono-culture in improving system security [12, 21]. Linger [16] proposed methods that systematically generate stochastic diversification in program source to increase system resistance and survivability. Obfuscation techniques (e.g., instruction-set randomization [2, 15] and address space randomization [3]) were proposed to safeguard systems against code-injection attacks and other memory error exploits. N-variant systems [4] execute a set of automatically diversified variants on the same inputs, and monitor their behavior to detect divergence that signals anticipated types of exploits, against which the variants are diversified.

Instead of artificially introducing diversity, some recent work focused on utilizing existing diverse software for network protection [17] and intrusion detection [8]. Some of these systems (e.g., the HACQIT system [14, 18] and its successor [22]) employed output voting to monitor outputs from diverse replicas, while others (e.g., Behavioral Distance [8–10]) monitor the low-level behavior of the diverse replicas.

An interesting and important assumption made by many of these systems utilizing off-the-shelf diverse software is that the diverse software is vulnerable only to different exploits. With this assumption, replicas constructed using diverse off-the-shelf software will not be compromised by the same attack. This is a reasonable assumption because most of the off-the-shelf diverse software is developed independently by different groups of developers, and so the same mistake/vulnerability is unlikely to be introduced. However, to the best of our knowledge, there has not been a systematic analysis to evaluate the extent to which this assumption is correct. Such analysis also guides users in choosing between artificially introducing diversity (e.g., instruction-set randomization, address space randomization, and N-variant systems) and utilizing off-the-shelf software products to introduce diversity.

In this paper, we present a systematic analysis on the effectiveness of utilizing off-the-shelf diverse software for improving system security. In particular, we evaluate the extent to which different off-the-shelf software suffers from the same vulnerability and exploit. This is achieved by carefully analyzing over 6,000 vulnerabilities published in the year of 2007.

To get a better idea of what is to be analyzed and how this analysis benefits systems that utilize off-the-shelf diverse software, consider an example in which a system uses behavioral distance [8–10] for intrusion detection (see Fig 1). In this example, a web service is provided by two diverse web servers running on two diverse operating systems. The same input, which may potentially be an attack input, is processed by both servers. Similar architectures, e.g., diverse servers on the same operating system, have also been introduced [14, 18, 22].
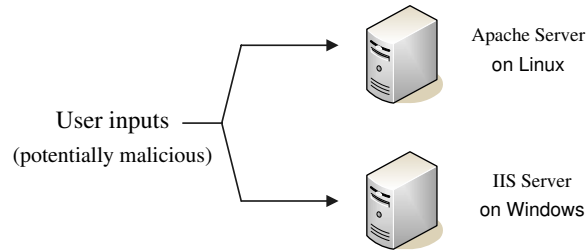
Fig. 1. An example (Behavioral Distance) of utilizing off-the-shelf diverse software

This system detects an intrusion when deviations are found in the two replicas when they are processing the same input. Such deviations may be detected in server outputs [14, 18, 22] or in the low-level behavior, e.g., system calls [8–10]. A very important observation is that such deviations occur only if the two replicas behave differently when processing the same malicious input. The system assumes that either the two replicas do not have the same vulnerability, or they cannot be exploited simultaneously with a single attack.

In order to evaluate the extent to which this assumption is valid, several questions need to be answered:

– Among the large number of vulnerable software products, how many of them have potential substitutes that provide similar functionality? For those that are software substitutes of one another, do they have the same vulnerability? If they do have the same vulnerability, can they be exploited with the same attack?
– Among the large number of vulnerable software products, how many of them can run on multiple operating systems? For those that run on multiple operating systems, do vulnerabilities of the software on one operating system propagate to the same software on a different operating system? If so, can they be exploited by the same attack when running on different operating systems?

To the best of our knowledge, there is no closely related work which could answer these questions. We systematically analyzed more than 6,000 vulnerabilities published in the year of 2007. In summary, our results show that more than 98.5% of the vulnerable application software products have software substitutes (and therefore can be used in a replicated system to detect intrusion), and the majority of them either do not have the same vulnerability, or cannot be compromised with the same exploit code. In addition, among the application software products, nearly half are officially supported to run on multiple operating systems. Although the different operating system distributions of the same product are likely (more than 80%) to suffer from the same vulnerability, the attack code is different in most cases. We also found evidence that indicates the use of diversity in increasing attack tolerance in other categories of vulnerable software.

It is not the objective of this paper to build systems utilizing software diversity or to evaluate how difficult it is to manage such systems. Instead, we measure the extent to which software diversity could be utilized to increase system security in using off-the-shelf software products.

In the rest of this paper, we first present the data source we utilized and some preliminary analysis (see Section 2). We then focus our analysis on the application software vulnerabilities in which we analyzed whether diverse software products providing the same services could suffer from the same vulnerability (see Section 3), and whether the same software product running on different operating systems will suffer from the same vulnerability and exploit (see Section 4). In Section 5, we present analysis on other vulnerable software products. Finally, we conclude in Section 6.

2   Source of Information and Preliminary Analysis

The main source of information we used for our analysis was the NVD/CVE (National Vulnerability Database/Common Vulnerabilities and Exposures) vulnerability database. We analyzed all the vulnerabilities recorded in CVE in the

year of 2007, which consist of 6,427 vulnerability entries[1]. To obtain detailed information on the vulnerabilities and the corresponding software products, we also consulted other sources including SecurityFocus, FrSIRT, CERT, Milw0rm, Secunia, OSVDB, IBM X-Force, as well as vulnerability advisories, security announcements, and bug lists from software vendors. After removing 87 entries that were rejected by CVE, the total number of vulnerabilities that we focused on was 6,340.

Note that the limited information introduced errors in our analysis. First, not all vulnerabilities are published. We only analyzed vulnerabilities found and published in 2007. Second, we may not have found all information on some published vulnerabilities. This is due to the limited resources we have, although we did our best in searching various public resources; it might also be the fact that some information about the vulnerabilities is not publicly available.

Our first step in the analysis was to find whether the vulnerable software has any substitutes (software products that offer similar functionality). We also categorized the vulnerabilities into five different types for further analysis.

2.1 Software without substitutes

To implement a replicated system with diverse replicas (e.g., the one shown in Fig 1), we need to find (at least) two software products that provide the same service (*software substitutes*) and/or software products that run on multiple operating systems. If the software product does not have any substitutes and runs only on a single operating system, then diversity using off-the-shelf software cannot work and one has to introduce diversity via other artificial means (e.g., address space randomization). Therefore, we first analyze all the vulnerable software products in the CVE database to see if they have any substitutes.

We find that most software products do have substitutes and those that do not have mostly fall into one of the following three categories:

- **Hardware specific software**: This includes hardware drivers and firmware only provided by corresponding hardware vendors.
- **OS specific software**: This includes utilities that are specific to an operating system, e.g., Mac Installer, Windows Login window. They are only provided by the OS vendor.
- **Domain specific and customized software**: This includes that used in medical, biological, nuclear and other specific domains. The customized software refers to that developed for a specific company, e.g., management software that is used in a specific company, ActiveX controls developed and used for online transactions on a specific web site.

Table 1 shows some examples of software products that do not have substitutes. An interesting observation is that we did not find many vulnerable software products from the CVE database that are domain specific or customized.

---

[1] The CVE 2007 database published on April 25, 2008 was used (`http://nvd.nist.gov/download/nvdcve-2007.xml`).

Table 1. Examples of software products without substitutes

Vendor	Product	CVE entry
ATI	Display driver	CVE-2007-4315
NVIDIA	Video driver	CVE-2007-3532
Intel	2200BG Wireless driver	CVE-2007-0686
HP	Help and Support Center	CVE-2007-3180
HP	Quick Launch Button	CVE-2007-6331
Alibaba	Alipay ActiveX control	CVE-2007-0827
Microgaming	Download Helper ActiveX	CVE-2007-2177

This does not necessarily mean that these software products do not have vulnerabilities. Domain specific and customized software products are used in a more controlled environment and it is less likely that they are reported in public vulnerability resources.
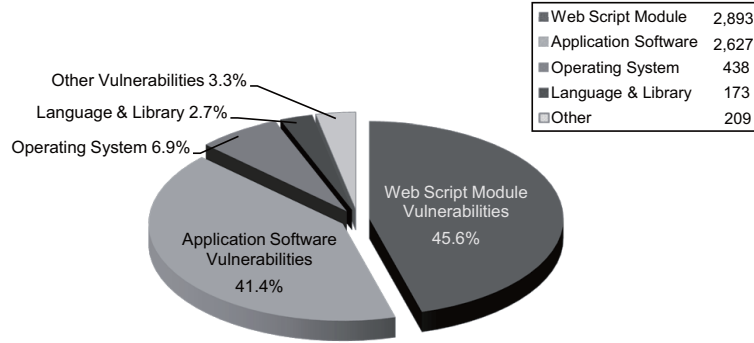
2.2 Vulnerable software categorization

Some vulnerabilities exist in application software that runs as user-space programs on an operating system. Others may exist in scripts that run on top of another software program. The analysis we performed varies according to the type of vulnerable software products. Therefore, we first put the vulnerable software into different categories.

- **Application software**: Application software is the most interesting because it is relatively easy to find the software substitutes. It is usually compiled into binary format and run as a process of its own in the user space. Word processors, web browsers, web servers and computer games are some examples of application software. It also includes plug-ins, extensions, and add-on's to application software, except those for a web server (see the next category).
- **Web script modules**[2]: These are light-weighted software modules which only run on web servers. We put them into a separate category instead of a sub-category of application software because of the large number of vulnerabilities in them. Examples include Content Management Systems (CMS), forums, bulletin boards, and other script modules.
- **Operating systems**: This category includes the operating system kernel and utilities that are closely related to the operating system, e.g., Apple Installer and the login window of Microsoft Windows.
- **Languages and libraries**: These include programming languages and libraries for general programming use, e.g., PNGlib (for decoding the PNG image) and SMTPlib (for implementing the SMTP protocol).

---

[2] They may be called web applications (e.g., in SANS [5]). We call this category web script modules, instead, to avoid the misunderstanding that it also contains web servers and browsers.

– **Others**: For example, firmware (including Routers, IP phones, hardware firewalls, etc.), software that runs on mobile phone, video game consoles (e.g., XBox) and so on.

Fig 2 shows the number of vulnerabilities in each software category and the corresponding percentage.

■ Web Script Module	2,893
■ Application Software	2,627
■ Operating System	438
■ Language & Library	173
□ Other	209

Fig. 2. Vulnerabilities in different software categories

2.3 Vulnerabilities in application software

As shown in Fig 2, 41.4% of the vulnerabilities found in 2007 are in application software. We focus our analysis on this category because it contains most of the commonly used and critical software, and it is usually what an intrusion detection system tries to protect. Not only that, it is also easy to find substitutes for an application software product, which makes it a natural candidate for introducing diversity. This is also the category for which information is best available and therefore the results of our analysis are most accurate.

The first analysis we did was to find the number of vulnerable application software products that do not have substitutes. As discussed in Section 2.1, this is important because one of the two ways of utilizing off-the-shelf software products to introduce diversity is to use software substitutes (the other is to run the same software on multiple operating systems). If many vulnerable application software products do not have any substitutes, then we will have to rely on the other way of introducing diversity.

We found 1,825 distinct application software products in all the 2,627 application software vulnerabilities[3], out of which only 25 (1.4%) do not have software

---

[3] A total number of 4,120 different names of software products were found in the descriptions of these vulnerabilities. Many of them were duplicates with different naming conventions or different product versions. After eliminating these duplicates, we found 1,825 distinct software products.

substitutes. Some of the examples were shown in Table 1. This result coincides with our expectation in view of the highly competitive software industry market.

We have found that most software products in this category have software substitutes. The next question is whether these software products and their corresponding substitutes have the same vulnerability or not. In order to do this analysis, we further classify the application software vulnerabilities (Box 1 in Fig 3) into two sub-categories: vulnerabilities that exist in multiple software products (Box 2) and vulnerabilities that exist in a single software product (Box 3).

The results of the classification are obtained by examining the vulnerable product information and the description of each vulnerability in the CVE database. Fig 3 shows that majority of the vulnerabilities (2037 out of 2627) exist in only a single software product, which is an evidence in favor of introducing diversity since the replicas constructed in a replicated system are unlikely to suffer from the same vulnerability. We look into each of the two categories for further analysis.
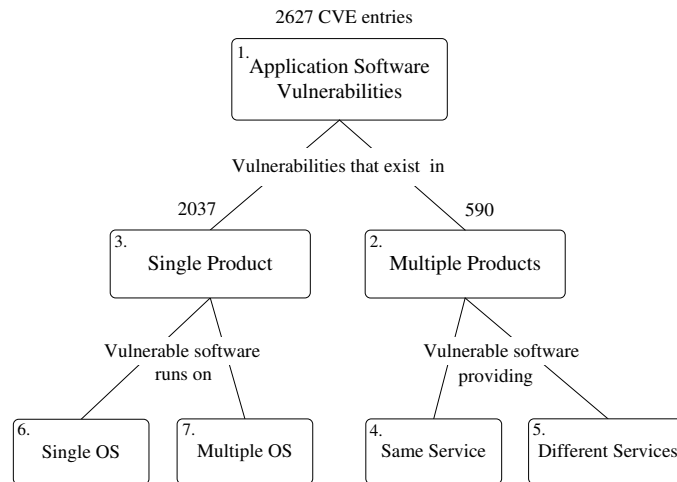
Fig. 3. Analysis on application software vulnerabilities

Among the vulnerabilities that exist in multiple software products (Box 2 in Fig 3), we want to find out whether software products suffering from the same vulnerability are substitutes of one another (i.e. whether they provide the same service). This analysis is important because only software products providing the same service can be used in an intrusion detection system using software diversity (such as the behavioral distance system shown in Fig 1). If software programs and their substitutes suffer from the same vulnerability (Box 4), then such intrusion detection systems will not be effective in detecting intrusions. We present our detailed analysis for this in Section 3. If multiple software products

– which suffer from the same vulnerability – are not providing the same service (Box 5), then they are not used simultaneously for constructing the intrusion detection system and therefore will not affect the effectiveness of diversity using off-the-shelf software products.

Among those vulnerabilities that exist in a single product (Box 3 in Fig 3), we want to find out how many of these software products can execute on multiple operating systems. For those that run on multiple operating systems (Box 7), it is also important to find out whether their vulnerabilities can be exploited in the same way when they are running on multiple operating systems. We present our analysis of these problems in Section 4. If a software product can only run on a single operating system (Box 6), then it cannot be used in a replicated system in which replicas are constructed using the different distributions of a single software product on multiple operating systems.

3   Vulnerabilities in Software Substitutes

As shown in Fig 3, there are 590 entries of vulnerabilities in multiple software products. Each of these vulnerabilities exists in more than one software product, which may or may not provide the same service. In this section, we first briefly show our method for finding vulnerabilities in software substitutes and our findings using this method (Section 3.1), and then discuss the attack code for exploiting the same vulnerability in these software substitutes (Section 3.2).

3.1   Finding vulnerabilities in software substitutes

An interesting observation is that the same vulnerability may be represented in multiple entries in the CVE database. For example, entries CVE-2007-2761 and CVE-2007-2888 correspond to the same vulnerability (see Table 2). For this reason, we cannot simply rely on different CVE entries to distinguish different vulnerabilities.

Different CVE entries that refer to the same vulnerability usually have similar descriptions. We use Vector Space Model [19], one of the classical models in information retrieval, to compare the descriptions for all CVE entries. The similarity between two vulnerability descriptions is calculated using

$$\mathsf{sim}(d_1, d_2) = \frac{\overrightarrow{d_1} \cdot \overrightarrow{d_2}}{|\overrightarrow{d_1}| \times |\overrightarrow{d_2}|} = \frac{\sum_{i=1}^{t} w_{i,1} \times w_{i,2}}{\sqrt{\sum_{i=1}^{t} w_{i,1}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,2}^2}}$$

where $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ are the descriptions of two vulnerability entries, $w_{i,j}$ is the weighting for the ith term in description d_j which is assigned with the frequency of the term. The threshold for the similarity score is set to 0.65 by manual tuning to obtain a good trade-off between the number of false positives and false negatives.

After the automatic comparison process using Vector Space Model and additional manual verification and correction, 410 distinct vulnerabilities are obtained from the 590 vulnerability entries that exist in multiple software products. We then performed a detailed analysis for each vulnerability and found that 29 of them (which involve 69 CVE entries) fall into the category in which the same vulnerability exists in multiple software products providing the same services (software substitutes). Some examples are shown in Table 2.

Table 2. Two examples of the same vulnerability in software substitutes

CVE Entry	Description
CVE-2007-2761	Stack-based buffer overflow in MagicISO 5.4 build 239 and earlier allows remote attackers to execute arbitrary code via a long filename in a .cue file.
CVE-2007-2888	Stack-based buffer overflow in UltraISO 8.6.2.2011 and earlier allows user-assisted remote attackers to execute arbitrary code via a long FILE string (filename) in a .cue file.
CVE-2007-0548	KarjaSoft Sami HTTP Server 2.0.1 allows remote attackers to cause a denial of service (daemon hang) via a large number of requests for nonexistent objects.
CVE-2007-3340	BugHunter HTTP SERVER (httpsv.exe) 1.6.2 allows remote attackers to cause a denial of service (application crash) via a large number of requests for nonexistent pages.
CVE-2007-3398	LiteWEB 2.7 allows remote attackers to cause a denial of service (hang) via a large number of requests for nonexistent pages.

The result shows that although many vulnerabilities (410) exist in multiple software products, only a small portion of them (29) exist in multiple software products that provide the same service. Note that although the Vector Space Model helped a lot in finding similar descriptions in different vulnerability entries, some manual analysis was needed to obtain the results shown above.

3.2 Exploit Code

In this step of the analysis, we further examine the 29 vulnerabilities that exist in software products providing the same services. If it happens that these software products are used to construct replicas in a replicated system (e.g., a behavioral distance system in Fig 1), then both replicas suffer from the same vulnerability. We want to find out whether the exploit codes on them are the same. If they are the same, then both replicas will be compromised by a single attack, and the intrusion detection system will fail to detect the intrusion.

We manage to find all the exploit codes (on multiple products) for 20 out of the 29 vulnerabilities. Exploit codes for the rest do not seem to be readily available to the public. By comparing the exploit codes for each of the 20 vulnerabilities for all the corresponding software substitutes, we found that the exploit code is the same across multiple software products for 14 of the 20 vulnerabilities.

It is not surprising that the same vulnerability will be exploited in the same way, even on different software products. A couple of notes are worth mentioning though. First, some of these vulnerabilities are about denial of service (DoS) attacks, which are usually not the type of intrusions a replicated system utilizing software diversity tries to detect [8, 9]. For example, the same exploit code for sending a large number of requests for non-existent pages will cause a denial of service in the three software products in the second group in Table 2. Therefore, this result is not necessarily a strong evidence against the effectiveness of using off-the-shelf software to introduce diversity. Second, we have not studied the effect of using multiple operating systems at this point. In some cases, the exploit codes may be dependent on the operating system, especially in code injection attacks (see the next section).

3.3 Summary

To summarize, our analysis of the application software products shows that 22.5% (590 out of 2627) of the vulnerability entries are vulnerabilities in multiple software products, among which 7.1% (29 out of 410) are vulnerabilities in multiple software products that provide the same service. For those vulnerabilities in multiple software products providing the same service, there are roughly 70% (14 out of 20) chances that the same exploit code can be used to compromise these software products. Although strictly speaking these three numbers cannot be multiplied together directly[4], they are very good indications that diverse off-the-shelf application software products can be utilized effectively in replicated systems to detect intrusion and increase system resilience against software attacks.

4 Software Products running on Multiple Operating Systems

Having analyzed the branch of vulnerabilities that exist in multiple software products in Fig 3 in Section 3, we now focus on the branch of vulnerabilities that exist in a single software product. As shown in Fig 3, this category consists of the majority of vulnerabilities in application software. Therefore, understanding how software products in this category can be utilized to introduce diversity is important. Here we focus on diversity via running software on multiple operating systems, since the vulnerability exists only on a single product and diversity via running software substitutes will definitely work. Running the same software on multiple operating systems is also a cheaper way of introducing diversity due to its lower cost in managing the replicated system.

In this section, we first briefly show the different operating systems we considered (Section 4.1), and then examine whether the software products in this

---

[4] Due to the lack of knowledge about the number of vulnerabilities each software has, the commonality of each software product in terms of the number of requests per unit time, the consequence of a compromise, and etc.

category run on multiple operating systems (Section 4.2). Finally, similar to our analysis in Section 3.2, we analyzed the corresponding exploit code in Section 4.3.

4.1 Different operating systems
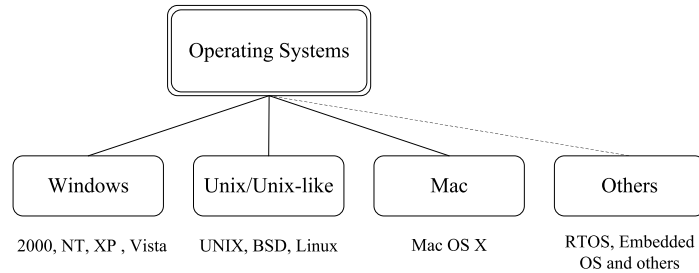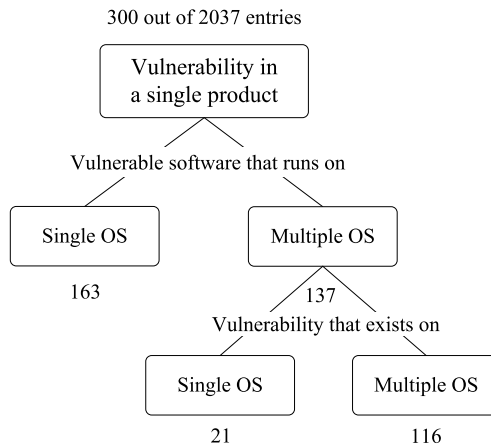

Fig. 4. Different operating systems

Fig 4 shows the different operating systems that we consider in our analysis. We classify operating systems into four families: Microsoft Windows, Unix/Unix-like, Mac and others (see Fig 4). This is mainly due to their different kernels and binary executable formats (Portable Executable for Windows systems, ELF for Unix and Unix-like systems, and Mach-O for Mac). Note that it is an important requirement that these operating systems are diverse so that the same exploit is unlikely to compromise the same program running on different operating systems. Although Mac OS X shares part of the kernel code with BSD operating systems, we show in Section 5.2 that they rarely share common vulnerabilities.

4.2 Software products running on multiple operating systems

Next, we want to find out whether software products in this category (in which vulnerabilities exist only in one software product) can run on multiple operating systems. Since a lot of manual work is required in this analysis, we randomly picked 300 out of the $2,037$ vulnerability entries for analysis. Results are shown in Fig 5.

Fig 5 shows that more than 54% (163 out of 300) of the software products we analyzed officially supports only one operating system. However, note that it is still possible to construct diverse replicas using software substitutes that provide the same service for them.

Among the rest of the 45.7% software products that are supported to run on multiple operating systems, 15.3% (21 out of 137) do not share the same vulnerability among different operating system versions (e.g., the first entry in Fig 6, in which the vulnerability exists only on the Windows version of Mozilla Firefox, but not on the Unix and Mac versions). From our analysis, this is mainly

300 out of 2037 entries

```
        ┌─────────────────┐
        │  Vulnerability in │
        │  a single product │
        └─────────────────┘
      Vulnerable software that runs on
     ┌───────────┐      ┌───────────┐
     │ Single OS │      │ Multiple OS │
     └───────────┘      └───────────┘
         163               137
                  Vulnerability that exists on
              ┌───────────┐    ┌───────────┐
              │ Single OS │    │ Multiple OS │
              └───────────┘    └───────────┘
                   21               116
```

Fig. 5. Vulnerable software on multiple operating systems

due to the fact that many of these vulnerabilities are design errors, which easily propagate across versions that run on multiple operating systems. One typical example is the vulnerability entry CVE-2007-5264, in which the client's information is sent unencrypted to the game server (second entry in Fig 6).

CVE Entry	Description
CVE-2007-3285	Mozilla Firefox before 2.0.0.5, when run on Windows, allows remote attackers to bypass file type checks and possibly execute programs via a (1) `file:///` or (2) `resource:URI` with a dangerous extension, followed by a NULL byte (%00) and a safer extension. (Vulnerability in only one of the OS versions of the software product)
CVE-2007-5264	Battlefront Dropteam 1.3.3 and earlier sends the client's online account name and password unencrypted to the game server. A remote attacker with administrative privileges could exploit this vulnerability to obtain user account, product key and other sensitive information. (Vulnerability in multiple OS versions of the software product)

Fig. 6. Vulnerabilities in software products that run on multiple OSes

4.3 Exploit Code

Similar to Section 3.2, in this subsection we look into the 116 vulnerabilities (each of which exists on multiple OS versions of the single software product), to see whether the same exploit code can be used to compromise the corresponding software program that executes on multiple operating systems.

We first consider a naive attacker, who is not aware that a replicated system where the vulnerable software is being executed on multiple operating systems. We assume that the attacker is trying to exploit a known vulnerability to execute some attack code, e.g., to overflow a buffer and overwrite a return address in order to execute a shellcode. There are at least two reasons why such an exploit is unlikely to succeed.

First, the source of the same software product on different OSes may be different. This could cause many differences in, e.g., memory layout which is critical for a successful buffer overflow. For example, calculating time intervals on Windows usually requires two variables (`SYSTEMTIME` and `FILETIME`) and a conversion between the two, whereas it usually takes only one variable (`timeval`) on Linux.

Second, even when the source is exactly the same for different OS distributions of the same product, the attack code to be executed may be different due to the different APIs and system calls across different operating systems. It is highly unlikely that the same machine code can be used on different operating systems, e.g., to open a shell. The system interface could be different even across OSes in the same family, e.g., different versions of Microsoft Windows. Table 3 shows some of the typical system calls and their corresponding system call numbers on different versions of the Windows operating system.

Table 3. System calls on Windows

System Call	NT	2000	XP	2003 Server	Vista
`NtClose()`	0x000f	0x0018	0x0019	0x001b	0x002f
`NtOpenFile()`	0x004f	0x0064	0x0074	0x007a	0x00b8
`NtReadVirtualMemory()`	0x0089	0x00a4	0x00ba	0x00c2	0x0102
`NtTerminateProcess()`	0x00bb	0x00e0	0x0101	0x010a	0x014f

Next, we consider a more sophisticated attack in which the attacker is aware that a replicated system running the vulnerable software on multiple operating systems is in use. If the attacker wants to evade the intrusion detection system, he/she will most likely have to design and implement an exploit code that first figures out which operating system is running and subsequently execute the corresponding exploit code (see Algorithm 1).

Note that Algorithm 1 is very different from one in which the attacker knows the operating system (and its version) to be exploited before sending the attack code. Many attack tools first interact with the vulnerable server to find out which operating system is running by using operating system fingerprinting techniques [7, 23]. After that, the attack packets specifically designed for the corresponding operating system are sent to the vulnerable server. This type of attacks will not work here because 1) the replicated system (e.g., Fig 1) usually removes any non-determinism in the system, which makes operating system fingerprinting impossible or inaccurate; 2) the same operating-system-specific

Algorithm 1 Exploiting the same software running on multiple OSes

```
os_ret ← os_test();
if is_win(os_ret) then
   win_attack_code();
else if is_unix(os_ret) then
   unix_attack_code();
else if is_mac(os_ret) then
   mac_attack_code();
end if
```

attack will be duplicated and sent to all replicas, and the attack only compromises the vulnerable replica (the difference of the behaviors of the compromised and uncompromised replicas makes such operating-system-specific attacks easily detectable).

There are at least two difficulties in implementing Algorithm 1. One is to implement `os_test()` which not only executes on all different operating systems but returns different outputs when executing on different operating systems. The other is that such an exploit code, which is at least several times that of the exploit code for any specific operating system, is usually too long to fit in the limited buffer available in the vulnerable program. We have not found a real attack that employs the technique shown in Algorithm 1.

Another observation is that only three cross-OS viruses have been reported in Kaspersky Lab's viruslist according to the statement issued by Kaspersky Lab[5]. According to Kaspersky Lab, all the three viruses are proof-of-concept malicious programs written purely with the intention of demonstrating that such viruses are possible. None of these viruses actually had any practical applications so far.

4.4 Summary

In this section, we analyze the vulnerabilities that exist in a single application software product. Our analysis shows that:

- 45.7% (137 out of 300) of the vulnerable software products involved in this category are officially supported on multiple operating systems;
- Among those that are officially supported on multiple operating systems, 84.7% have the vulnerability propagated across multiple OS versions;
- At least two factors (different memory layout and different machine instructions) make it difficult to construct an exploit that can compromise software running on multiple operating systems simultaneously. No such practical attacks have been reported.

These findings show that roughly 50% of the software products are candidates for a replicated system running the same software on multiple operating systems. Even if the same vulnerability exists on multiple replicas, compromising them simultaneously remains difficult. However, due to the fact that most of

---

[5] http://www.kaspersky.com/news?id=184875287

these vulnerabilities are shared among the different OS versions of the same software, utilizing diverse operating systems is not as effective as utilizing software substitutes.

5   Vulnerabilities in Other Software Products

In this section, we present our analysis on the other three categories, namely web script modules, operating systems, language and libraries.

5.1   Web script modules

Software in this category consists of light-weighted products that run on web servers to provide web-based applications. Examples include forums, bulletin boards, shopping carts and other script modules. We analyzed the CVE vulnerability database and found close to $3,000$ entries that fall into this category. Some common and well-known types are shown in Table 4.

Table 4. Vulnerabilities in web script modules

Vulnerability Types	Number of entries	Percentage
Cross-site scripting	714	24.7%
SQL injection	669	23.1%
PHP remote file inclusion	634	21.9%
Directory/Path traversal	267	9.2%
Cross-site request forgery	50	1.7%
Others	559	19.3%
Total	2893	100%

An interesting finding is that most of the vulnerable software in this category is operating system independent. For example, most PHP modules are deployed on Apache web servers, which can run on all common operating systems. This means that we could use diverse operating systems to introduce software diversity. However, it is different from the application software we analyzed in Section 4, since many of the web script modules operate on top of a web server, and seldom interact with the operating system. If the vulnerable software does not interact with the operating system, then constructing replicas using diverse operating systems is not an effective way of introducing diversity because the exploit code is likely to be the same on different replicas. Therefore, we shift our focus of analysis to using software substitutes for introducing diversity.

Cross-site scripting (XSS) vulnerabilities Cross-site scripting (XSS) is one of the most common web script module vulnerabilities in the CVE database. Attackers exploit this vulnerability by injecting malicious scripts into the output of an application (usually a web page) which is sent to the client's web browser.

This script is then executed on the client's web browser and used to transfer sensitive data to a third party (i.e., the attacker) [24]. Unlike other types of web vulnerabilities, XSS vulnerabilities exist and are exploited on the server side but take effects on the client side. Thus, the protection and prevention mechanisms are carried out both on the server side [25] and the client side [24].

In most cases, the server-side scripts are vulnerable no matter what operating systems or web servers on which the scripts run (see an example in Fig 7, the attack payload is usually some malicious `HTML/JavaScript`, which is first posted to the server and then downloaded and run at the client side), thus introducing diversity on the server side is not effective. However, introducing diversity on the client side by utilizing diverse browsers is possible. Fig 8 shows two examples of XSS attack payload in the exploit code as shown in Fig 7.

Description	Cross-site scripting vulnerability in `picture.php` in Advanced Guestbook 2.4.2 allows remote attackers to inject arbitrary web script or HTML via the picture parameter.
Exploit code	`http://www.site.com/picture.php?picture=[attack payload]`

Fig. 7. CVE entry CVE-2007-0605 and the corresponding exploit code

[Payload 1] Works for Internet Explorer 6.0 but not Opera 9.0 or Firefox 2.0

```
<IMG SRC=javascript:location.replace('http://
evil.com/steal/index.asp?cookies='+encodeURI(document.cookie))>
```

[Payload 2] Works for Opera 9.0 but not Internet Explorer 6.0 or Firefox 2.0

```
<IMG SRC=javascript:document.createElement('IMG').setAttribute('src',
'http://evil.com/steal/index.asp?cookies='+encodeURI(document.cookie))>
```

Fig. 8. XSS attacks that have different impact on browsers

Both XSS attack payloads shown in Fig 8 utilize the `HTML` tag `<IMG>` and are used for stealing cookies from client machines that access the vulnerable web site. The exploit codes do not have the same effect on the contemporary browsers because of the implementation difference. The evidences that XSS attack codes have different effects on different browsers can also be found from other resources. For example, 68 out of the 110 XSS attack vectors on the XSS Cheat Sheet (`http://ha.ckers.org/xss.html`) have different impacts on diverse web browsers. Note that the application scenario here is slightly different from the example shown in Fig 1: utilizing diverse browsers to construct the replicated system is a client-side solution instead of the server-side example shown in Fig 1. Our results show that by comparing the different impacts on different browsers when given the same input, many XSS attacks could be detected. Analyzing the detection rate of such a system is out of the scope of this paper.

SQL injection SQL injection arises when a user input is not correctly or sufficiently filtered. SQL injection attacks are usually launched through specially crafted user inputs on web applications that use strings to construct SQL queries [1]. Although simple SQL statements are constructed exactly the same for different databases, they are different in constructing sophisticated SQL Injection exploits. Consider Blind SQL Injection in CVE-2007-1166, CVE-2007-3051, and many other vulnerable products. The exploit code utilizes the following SQL statements (simplified version).

$$IF\ ((SELECT\ user)\ =\ 'Alice')\ SELECT\ 1\ ELSE\ SELECT\ 1/0$$

After receiving this request, the SQL Server will throw a divide-by-zero error if the current user is not Alice, while the MySQL server will report a *parsing error*.[6] There has also been research on utilizing diverse off-the-shelf databases to obtain fault tolerance [11].

Directory traversal Directory traversal (or path traversal) vulnerabilities appear when web applications do not sufficiently validate or sanitize the user-supplied file names. It may allow attackers to gain access to directories and files that reside outside of the directory of web documents.

A notable difference in traversing directories on diverse operating systems is that Unix and Unix-like systems use "`../`", while Windows systems use "`..\`". Not only that, the root directory on Windows uses the "`<drive letter>:\`" format, which limits directory traversal to a single partition (e.g., `C:\`). There are other differences, e.g., the file organization also varies a lot on different operating systems.

Remote File Inclusion (RFI) RFI vulnerabilities allow an attacker to include his own malicious PHP code on a vulnerable web application. RFI attacks are possible because of several PHP configuration flags that are not carefully set. This vulnerability could be avoided easily by disabling two global flags in PHP [6]. Thus, RFI vulnerabilities are not the focus of our study in this paper.

Cross-site Request Forgery (CSRF) By launching a successful CSRF attack to a user, an adversary is able to initiate arbitrary `HTTP` requests from that user to the vulnerable web application [13]. CSRF attacks are usually executed by causing the victim's web browsers to create hidden `HTTP` requests to restricted resources. Therefore, similar to XSS vulnerabilities, using diverse browsers is a possible way of detecting CSRF vulnerabilities.

---

[6] Example statement here was tested on SQL Server 2005 and MySQL 5.0. More resources on different syntax for constructing SQL Injection attacks to different databases can be found on SQL Injection Cheat Sheet at `http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/`

5.2 Operating systems, languages and libraries

For operating system vulnerabilities, we try to find out if diverse operating systems have the same vulnerability. We find that Mac OS X has some common vulnerabilities with BSD (e.g., CVE-2007-0229), mainly because the implementation of Mac OS X kernel shares part of the code of BSD kernel [20]. However, these common vulnerabilities only constitute 2% (2 out of 98) of all the vulnerabilities on Mac OS, which indicates that utilizing Unix/Unix-like OS and Mac OS to construct replicas is effective.

Another observation we have is that different Linux operating systems have many common vulnerabilities, since they share the same kernel (e.g. CVE-2007-3104, CVE-2007-6206 and others). These vulnerabilities contribute 64% (71 out of 111) of all the Linux OS vulnerabilities, which shows that different Linux OS are not diverse enough. Finally, by examining all the 438 OS vulnerabilities, no evidence has been found that the same OS vulnerability exists in both Windows and Unix/Unix-like or in both Windows and Mac operating systems.

Many programming languages and libraries (e.g., Java, PHP, Perl, and etc.) support multiple operating systems. However, our analysis in the CVE vulnerability database shows that many of the vulnerabilities in these products are platform dependent. For example, CVE-2007-5862 (a Java vulnerability that exists only in Mac OS X) and CVE-2007-1411 (a PHP buffer overflow vulnerability that allows local, and possibly remote, attackers to execute arbitrary code via several vulnerable PHP functions that exists only in Windows[7]).

5.3 Summary

Although in general, software diversity is not very effective in web applications, it is successful in detecting exploits of some web script module vulnerabilities by, for example, utilizing diverse browsers to defend against XSS and CSRF attacks and utilizing diverse databases to detect SQL Injection attacks.

Most OS vulnerabilities only exist in one OS family, which indicates that diversity is useful when utilizing diverse operating systems of different OS families. Although most language and library vulnerabilities are platform independent, there are cases in which they exist in only one particular OS version.

6 Conclusion

In this paper, we analyzed the vulnerabilities published in 2007 to evaluate the effectiveness of two ways of introducing software diversity utilizing off-the-shelf

---

[7] This result is obtained by analyzing NVD/CVE, SecurityFocus and the PHP Buglists. SecurityFocus gives misleading information which indicates that this vulnerability exists on Unix/Unix-like systems (see http://www.securityfocus.com/bid/22893/info). However, the PHP Bug Info (Bug #40746) shows that it is a problem with the function dbopen() in the Microsoft ntdblib library, and does not exist when compiled with FreeTDS version of the dblib library that is used by Unix/Unix-like systems.

software: one is by utilizing different software products that provide the same service, and the other is by utilizing the same software product on different operating systems.

The results show that more than 98.5% of the vulnerable application software products have substitutes and the chance that these software substitutes be compromised by the same attack is very low. Nearly half of the application software products are officially supported to run on multiple operating systems. Although the different OS distributions of the same product have more than 80% of a chance to suffer from the same vulnerability, their attack code is quite different. For the web script modules and other types of software, although software diversity is less effective than that in the application software, some evidence has been found that there are possible ways to benefit from software diversity in these categories.

The limitation of our work mainly includes two parts. The first is that a large amount of manual work has been spent in order to get the accurate statistical results, which is too costly and time consuming. Other information retrieval and artificial intelligence techniques could be applied in our future work to speed up the analysis process. The other limitation is that we have not yet obtained the statistics for some categories due to the large information search space and the lack of closely related resources, which is a challenging task that remains to be done in the future.

References

1. Sruthi Bandhakavi, Prithvi Bisht, P. Madhusudan, and V. N. Venkatakrishnan. Candid: preventing sql injection attacks using dynamic candidate evaluations. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 12–24, New York, NY, USA, 2007. ACM.

2. Elena Gabriela Barrantes, David H. Ackley, Trek S. Palmer, Darko Stefanovic, and Dino Dai Zovi. Randomized instruction set emulation to disrupt binary code injection attacks. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 281–289, New York, NY, USA, 2003. ACM.

3. Sandeep Bhatkar, Daniel C. DuVarney, and R. Sekar. Address obfuscation: an efficient approach to combat a board range of memory error exploits. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, page 8, Berkeley, CA, USA, 2003. USENIX Association.

4. B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser. N-variant systems – A secretless framework for security through diversity. In *Proceedings of the 15th USENIX Security Symposium*, August 2006.

5. Rohit Dhamankar. SANS Top-20 Security Risks, 2007. `http://www.sans.org/top20/2007/`.

6. Jake Edge. Remote file inclusion vulnerabilities. Octobor 2006. `http://lwn.net/Articles/203904/`.

7. Gordon Lyon Fyodor. Remote os detection via tcp/ip stack fingerprinting. Technical report, INSECURE.ORG, October 1998.

8. Debin Gao, Michael K. Reiter, and Dawn Song. Behavioral distance for intrusion detection. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, pages 63–81, 2005.

9. Debin Gao, Michael K. Reiter, and Dawn Song. Behavioral distance measurement using hidden markov models. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID 2006)*, pages 19–40, 2006.

10. Debin Gao, Michael K. Reiter, and Dawn Song. Beyond output voting: Detecting compromised replicas using HMM-based behavioral distance. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, July 2008.

11. Ilir Gashi and Peter Popov. Fault tolerance via diversity for off-the-shelf products: A study with sql database servers. *IEEE Transactions on Dependable Secure Computing*, 4(4):280–294, 2007. Member-Lorenzo Strigini.

12. D. Geer, R. Bace, P. Gutmann, P. Metzger, C. P. Pfleeger, J. S. Quarterman, and B. Schneier. Cyberinsecurity: The cost of monopoly. Technical report, CCIA, 2003.

13. Nenad Jovanovic, Engin Kirda, and Christopher Kruegel. Preventing Cross Site Request Forgery Attacks. In *IEEE International Conference on Security and Privacy for Emerging Areas in Communication Networks (Securecomm)*, 2006.

14. J. Just, J. Reynolds, L. Clough, M. Danforth, K. Levitt, R. Maglich, and J. Rowe. Learning unknown attacks - A start. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, 2002.

15. Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 272–280, New York, NY, USA, 2003. ACM.

16. Richard C. Linger. Systematic generation of stochastic diversity as an intrusion barrier in survivable systems software. In *HICSS '99: Proceedings of the Thirty-Second Annual Hawaii International Conference on System Sciences-Volume 3*, page 3062, Washington, DC, USA, 1999. IEEE Computer Society.

17. Adam J. O'Donnell and Harish Sethu. On achieving software diversity for improved network security using distributed coloring algorithms. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 121–131, New York, NY, USA, 2004. ACM.

18. J. Reynolds, J. Just, E. Lawson, L. Clough, and R. Maglich. The design and implementation of an intrusion tolerant system. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN02)*, 2002.

19. G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

20. Amit Singh. *Mac OS X Internals: A Systems Approach*. Addison-Wesley, 2006.

21. Mark Stamp. Risks of monoculture. *Communications of the ACM*, 47(3):120, 2004.

22. E. Totel, F. Majorczyk, and L. Me. COTS diversity based intrusion detection and application to web servers. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, 2005.

23. Chris Trowbridge. An overview of remote operating system fingerprinting. Technical report, The SANS Institute, July 2003.

24. Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross-site scripting prevention with dynamic data tainting and static analysis. In *Proceeding of the Network and Distributed System Security Symposium (NDSS)*, February 2007.

25. Gary Wassermann and Zhendong Su. Static detection of cross-site scripting vulnerabilities. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 171–180, New York, NY, USA, 2008. ACM.