Some Vulnerabilities Are Different Than Others

Studying Vulnerabilities and Attack Surfaces in the Wild

Kartik Nayak*, Daniel Marino†, Petros Efstathopoulos†, Tudor Dumitraș*

*University of Maryland, College Park †Symantec Research Labs

Abstract

The security of deployed and actively used systems is a moving target, influenced by factors not captured in the existing security metrics. For example, the count and severity of vulnerabilities in source code, as well as the corresponding attack surface, are commonly used as measures of a software product's security. But these measures do not provide a full picture. For instance, some vulnerabilities are never exploited in the wild, partly due to security technologies that make exploiting them difficult. As for attack surface, its effectiveness has not been validated empirically in the deployment environment. We introduce several *security metrics derived from field data* that help to complete the picture. They include the count of vulnerabilities exploited and the size of the attack surface actually exercised in real-world attacks. By evaluating these metrics on nearly 300 million reports of intrusion-protection telemetry, collected on more than six million hosts, we conduct an empirical study of security in the deployment environment. We find that none of the products in our study have more than 35% of their disclosed vulnerabilities exploited in the wild. Furthermore, the exploitation ratio and the exercised attack surface tend to decrease with newer product releases. We also find that hosts that quickly upgrade to newer product versions tend to have reduced exercised attack-surfaces. The metrics proposed enable a more complete assessment of the security posture of enterprise infrastructure. Additionally, they open up new research directions for improving security by focusing on the vulnerabilities and attacks that have the highest impact in practice.

1 Introduction

In order to improve the security of our software systems, we need to be able to measure how they are impacted by the various defensive techniques we introduce to protect them. Measuring security, however, is challenging. Many security metrics have been proposed, including the total count of vulnerabilities in source code, the severity of these vulnerabilities, the size of the attack surface and the time window between the vulnerability disclosure and the release of a patch. System administrators and security analysts often rely on these metrics to assess risk and to prioritize some patches over others, while developers use them as guidelines for improving software security. Practical experience, however, suggests that the existing security metrics exhibit a low level of correlation with

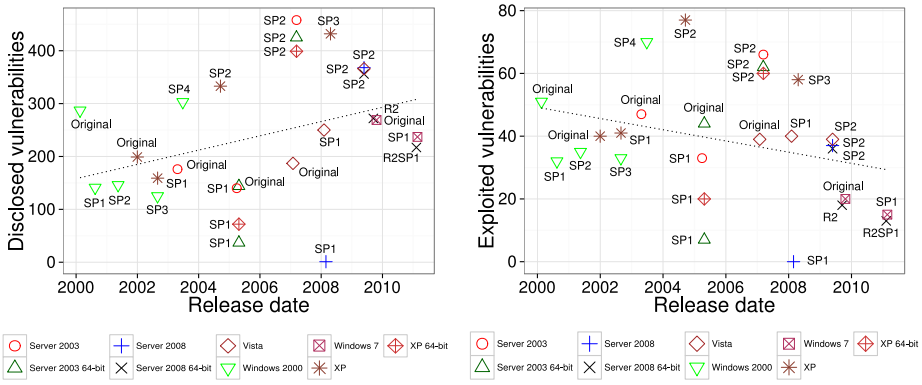
vulnerabilities and attacks, and they do not provide an adequate assessment of security [1, 2].

A vulnerability is a programming error that can be exploited by an attacker to subvert the functionality of the vulnerable software by feeding it malformed inputs (e.g., network packets or web form submissions that evade the program's error checks, allowing the attacker to execute arbitrary code on the host). For example, the vulnerability identified by CVE-2007-1748 [3] corresponds to a buffer overflow in the RPC interface for the DNS server included in several versions of Windows server. It allows remote attackers to execute arbitrary code by sending specially crafted network packets to the vulnerable host. The *total number of vulnerabilities* discovered in source code is commonly used as a measure of the system's security [1, 2]. However, this metric does not account for the fact that cyber attackers never make use of some of the discovered vulnerabilities, which may be hard to successfully exploit in the presence of security technologies such as data execution prevention (DEP) and address space layout randomization (ASLR). For example, CVE-2007-1748 was exploited in the wild, but there is no evidence of cyber attacks exploiting CVE-2007-1749.

Another popular metric is based on the observation that attacks can succeed only if the vulnerable software accepts input from potential attackers. For this reason, system administrators have long advocated turning off unnecessary system services to avoid exposure to exploits of unpatched or unknown vulnerabilities. For example, network-based attacks exploiting CVE-2007-1748 are unsuccessful—even if the vulnerability was not yet patched—if the DNS server is not running. This idea is formalized in the concept of *attack surface* [4, 5], which quantifies the number and severity of potential attack vectors that a system exposes by using a formula that takes into account the open sockets and RPC endpoints, the running services and their privilege level, the active Web handlers, the accounts enabled, etc. Reducing the attack surface, however, does not always improve security; for example, including security mechanisms in the OS may increase the attack surface, but renders the system more secure. Furthermore, the attack surface of software products changes after they are deployed in the field, as users install new applications and modify system configuration. To the best of our knowledge, the size and variability of attack surfaces has not been evaluated empirically in the field. It is, therefore, difficult to determine the effectiveness of this metric in capturing real-world conditions.

These examples illustrate that our ability to assess the security of systems that are deployed and actively utilized is currently limited by the metrics being used. In particular, the developers and the users may employ different security metrics. For example, one way of estimating the vulnerability density and the attack surface is to use existing tools that measure these properties by directly analyzing the code and the configuration of the system in question [6, 7]. However, these measurements are conducted in lab conditions, and do not reflect the real-world security of systems that are deployed and actively used in the field.

For these reasons, users are ultimately interested in metrics that help them assess the effectiveness of these techniques in the field. Figure 1 illustrates this



(a) All vulnerabilities disclosed publicly (from NVD [8]).

(b) Vulnerabilities exploited in the wild (cf. Section 4.2).

Fig. 1. Number of vulnerabilities disclosed and exploited for Microsoft Windows over 11 years of releases, with linear-regression trend lines.

problem. The number of vulnerability exploits is not proportional to the total number of vulnerabilities discovered in Windows OSes, and the two metrics follow different trends (as suggested by the trend lines in Figure 1). Additionally, there is no apparent correlation between the number of vulnerabilities discovered, and the size of the OS code.[1] This suggests the existence of deployment-specific factors, yet to be characterized systematically, that influence the security of systems in active use.

Our *goal* in this paper is to propose new metrics that better reflect security in the real world and to employ these metrics for evaluating the security of popular software. Rather than measuring security in lab conditions, we derive metrics from field-gathered data and we study the trends for vulnerabilities and attack surfaces exercised in attacks observed in the real world. While the vulnerability count and the attack surface are metrics that capture the *opportunities* available to attackers, we instead focus on attempted, though not necessarily successful, attacks in the field. This new understanding, potentially combined with existing metrics, will enable a more accurate assessment of the risk of cyber attacks, by taking into account the vulnerabilities and attacks that are known to have an impact in the real world. For instance, although it is very important that *all* vulnerabilities be addressed, system administrators might find it useful to understand if some vulnerabilities are more critical than others, as a criterion for risk assessment and for prioritizing patch deployment.

A *meta-goal* of our study is to illustrate how this analysis can be conducted using data that is available to the research community for further studies, allowing other researchers to verify and to build on our results.

[1] Approximate lines of code, in millions: Windows 2000 $\simeq$ 30, Windows XP $\simeq$ 45, Windows Server 2003 $\simeq$ 50, Windows Vista, Windows 7 $>$ 50 [9–11].

We make two contributions in this paper:

1. We propose field-measurable security metrics to identify *which* vulnerabilities are exploited (count of exploited vulnerabilities and exploitation ratio), quantify *how often* they are exploited (attack volume and exercised attack surface) and study *when* they are exploited.
2. We perform a systematic study of the exploited vulnerabilities and the exercised attack surfaces on 6,346,104 hosts. Our empirical findings include:
   - Few vulnerabilities are exploited in the real world: For Microsoft Windows, Adobe Reader, Microsoft Office and Microsoft Internet Explorer, fewer than 35% of the disclosed vulnerabilities are ever exploited. When the vulnerabilities for all these products are considered together, only 15% are exploited. Moreover, this exploitation ratio tends to decrease with newer product releases.
   - The exploitation ratio varies between products, implying that the number of vulnerabilities may not be a reliable indicator of real-world exploits and the security of a product.
   - The average exercised attack surface for Windows and IE has decreased with each new major release, and the latest versions of Reader have a far smaller exercised attack surface than earlier versions.
   - A significant portion of the average exercised attack surface of a host is due to products installed on the system and not the OS.
   - A few vulnerabilities (e.g. CVE-2008-4250 and CVE-2009-4324) account for a disproportionate number of attacks and influence our metrics.
   - Quicker upgrades to newer product versions are correlated with reduced exercised attack surfaces.
   - More than 93% of Windows and IE users are expected to remain unattacked at the end of four years. In contrast, only 50% Reader users are expected to remain unattacked for the same time period.

The rest of the paper is organized as follows. In Section 2 we review the related work on security metrics. In Section 3 we introduce our field-based metrics, and in Section 4 we describe how we measure them. Section 5 presents our empirical findings, and Section 6 discusses their implications.

2 Related work

The total number of known vulnerabilities present in source code and their severity, as represented by the Common Vulnerability Scoring System (CVSS), are commonly used security metrics. For example, Rescorla [12] analyzes the number of vulnerabilities disclosed for 4 operating systems in order to determine whether the practice of vulnerability disclosures leads to reliability growth over time. Ozment et al. [13] study the rate of vulnerability finding in the foundational code of OpenBSD and fit the data to a vulnerability growth model in order to estimate the number of vulnerabilities left undiscovered. Clark et al. [14] examine the challenge of finding vulnerabilities in new code and show that the time to discover the first vulnerability is usually longer than the time to discover the second one (a phenomenon they call the "honeymoon effect").

Several studies employ vulnerability counts as an implicit measure of security. Shin et al. [1] evaluate code complexity, code churn and the developer activity to determine the vulnerable code locations in Linux. Bozorgi et al. [15] propose a machine learning approach for predicting which vulnerabilities will be exploited based on their CVSS scores. In consequence, the National Institute of Standards and Technology (NIST) recommends CVSS scores as the reference assessment method for software security [16]. Based on an empirical analysis, Ransbotham et al. suggest that vulnerabilities in open source software have an increased risk of exploitation, diffuse sooner and have a larger volume of exploitation attempts than closed source software [17].

Because programming errors are thought to be inevitable, reducing the attack surface was proposed a decade ago as an alternative approach to securing software systems [4]. In order to exploit a vulnerability, an attacker must have an opportunity to exercise the vulnerable code, for instance by sending a message to a service listening on a network port. Such an opportunity is known as an attack vector. Attack surface reduction works by decreasing the number and severity of potential attack vectors exposed by the OS and its applications. Like the vulnerability metrics, attack surface is typically measured by analyzing the source code or the configuration of the system. Howard [4] defines an attack surface metric as a weighted combination of targets, enablers, communication channels, protocols and access rights. The Microsoft Attack Surface Analyzer tool [6] estimates the attack surface from the system configuration and monitors changes over time. Manadhata et al. [5] define attack surface as a triple, where each element represents the sum of the damage potential-effort ratio for (i) entry and exit points, (ii) channels and (iii) untrusted data items. Kurmus et al. [18] define attack surface as a function of the call graph, a set of entry functions, and a set of barrier functions.

2.1 Problems with the existing security metrics

Measurability. Some security metrics are difficult to assess. Code-based metrics have been shown to exhibit statistically significant, but small, correlations with security vulnerabilities [1,2]. Evaluating attack-surface metrics can require access to both the source code of the product and to the composition of its deployment environments [5]. In contrast, the metrics we propose can be computed from the typical telemetry collected by security products (e.g. anti-virus, intrusion-protection system) running on end-hosts around the world.

Representativeness. The existing metrics do not reflect security in the field. The CVSS "exploitability" subscore is not a good predictor for which vulnerabilities are exploited in the real world (though, most exploited vulnerabilities do have high exploitability) [19], and the CVSS-based risk evaluation does not fit the real attack data, as observed in the wild [20]. The attack surface metrics have not been validated empirically, in the deployment environment. In particular, the impact of user behavior (e.g. installing new software) on attack surfaces is unknown. In contrast, we focus on alternative metrics that better reflect the risk

Table 1. Summary of notations.

Measurement subjects	
p	A software product.
h	A host.
v	A vulnerability.
m	A calendar month.
Sets of subjects	
$V_p(t)$	The set of vulnerabilities disclosed for a product p up to time t.
$V_p^{ex}(t)$	The set of vulnerabilities known to be exploited for a product p up to time t.
$V_p^{prog,ex}(t)$	The set of progressive vulnerabilities known to be exploited for a product p up to time t.
$H_{p,m}$	The set of hosts that have product p installed during month m.
$A_{h,m}^v$	The set of attacks against host h during month m attempting to exploit vulnerability v.
Vulnerability-based metrics	
$ER^p(t)$	Exploitation ratio for vulnerabilities of product p until time t.
$ER_{Prog}^p(t)$	Exploitation ratio for progressive vulnerabilities of product p until time t.
Attack-based metrics	
EP^p	Exploitation prevalence, the proportion of hosts with p installed that ever experience an attack exploiting p.
AV_p	Attack volume, the average number of attacks per machine-month for a product p.
$EAS_h(m)$	Exercised attack surface of a host h for month m.
$EAS_h^p(m)$	Exercised attack surface of a host h for month m w.r.t. p.
$AEAS_p$	Average exercised attack surface w.r.t p over all machine-months p was installed.
$AEAS(m)$	Average exercised attack surface over all hosts for all products during month m.
$AEAS_h^p$	Average exercised attack surface over all months p was installed on host h w.r.t p's vulnerabilities.

for end-users. These metrics include the *number of vulnerabilities exploited* and the *size of the attack surface exercised* in real-world cyber attacks. We also assess how the exercised attack surface varies from one host to another.

3 Proposed metrics

In this section we introduce our proposed security metrics which, in contrast to existing metrics, are measured in the deployment environment. We consider this distinction important since security is a moving target once a system is deployed; attackers exploit new vulnerabilities (to subvert the system's functionality), vendors distribute software updates (to patch vulnerabilities and to improve security) and users reconfigure the system (to add functionality). Since our new metrics are derived from field-gathered data, they capture the state of system security as experienced by the end users. Table 1 summarizes the notations we employ.

The following metrics capture the notion of whether disclosed vulnerabilities get exploited.

1. *Count of vulnerabilities exploited in the wild.* For a product p, we consider the number of vulnerabilities known to have been exploited in the wild,

$\left|V_p^{ex}\right|$, to be an important metric. We combine information from NVD [8] and Symantec's databases of attack signatures [21, 22] to obtain the subset of a product's disclosed vulnerabilities that have been exploited. (These data sources are described in more detail in Section 4.1.) V_p^{ex} is the subset of the vulnerabilities listed in NVD that affect product p and which have at least one Symantec signature referencing the vulnerability's CVE identifier. Prior research has suggested that these signatures represent the best indicator for which vulnerabilities are exploited in real-world attacks [19].

2. *Exploitation ratio.* The exploitation ratio is the proportion of disclosed vulnerabilities for product p that have been exploited up until time t. It captures the likelihood that a vulnerability will be exploited.

$$ER^p(t) = \frac{\left|V_p^{ex}(t)\right|}{|V_p(t)|}$$

We also propose the following metrics that capture how often vulnerabilities are exploited on hosts in the wild.

1. *Attack Volume.* The attack volume is a measure that captures how frequently a product p is attacked. Intuitively, it is the average number of attacks experienced by a machine in a month due to product p being installed. It is defined as:

$$AV_p = \frac{\sum_m \sum_{h \in H_{p,m}} \sum_{v \in V_p^{ex}} \left|A_{h,m}^v\right|}{\sum_m |H_{p,m}|}$$

That is, the number of attacks that exploit a vulnerability of p against hosts with p installed, normalized by the total number of machine-months during which p was installed.

2. *Exercised Attack Surface.* We also define the *exercised attack surface*, $EAS_h(m)$, which captures the portion of the theoretical attack surface of a host that is targeted in a particular month. Intuitively, the exercised attack surface is the number of distinct vulnerabilities that are exploited on a host h in a given month m. We compute the exercised attack surface attributable to a particular product using the following formula:

$$EAS_h^p(m) = \left|\left\{v \in V_p^{ex} \;\middle|\; \left|A_{h,m}^v\right| > 0 \wedge h \in H_{p,m}\right\}\right|$$

That is, the cardinality of the set of p's vulnerabilities used in attacks against h in month m, or 0 if p is not installed on h during month m. We can now define the exercised attack surface for a host over all installed products as:

$$EAS_h(m) = \sum_p EAS_h^p(m)$$

We can then average these per host, per month metrics in various ways as listed in Table 1. In particular we can calculate an average exercised attack surface metric for a product p as follows:

$$AEAS_p = \frac{\sum_m \sum_{h \in H_{p,m}} EAS_h^p(m)}{\sum_m |H_{p,m}|}$$

Intuitively, $AEAS_p$ represents the average number of vulnerabilities that are exploited for product p during one month on one machine. So, while AV_p captures the volumes of attacks against a product, $AEAS_p$ captures the diversity of those attacks.

4 Experimental methods

4.1 Data sets

Public vulnerability databases. The National Vulnerability Database (NVD) [3] is a database of software vulnerabilities which is widely accepted for vulnerability research. For each vulnerability, NVD assigns a unique identifier called CVE-ID. Additionally, we employ the Open-Sourced Vulnerability Database (OSVDB) [23] to determine the dates when proof-of-concept exploits are published for the NVD vulnerabilities.

Symantec signatures. Symantec security products include an extensive database of signatures. Attack signatures are described on a publicly-accessible web site [24], and they are employed by Symantec's intrusion-prevention systems (IPS) for identifying attacks in network streams—including attempts to exploit known OS or application vulnerabilities. Symantec also maintains descriptions of anti-virus (AV) signatures, used by anti-virus products to scan files for known threats [21]. For threats that involve exploits, these data sets indicate the CVE-ID of the vulnerability exploited. Prior research has suggested that these signatures represent the best indicator for which vulnerabilities are exploited in real-world attacks [19].

Worldwide Intelligence Network Environment (WINE). In order to analyze the attacks happening on different hosts running different products, we use WINE [25], which contains records of which signatures are triggered in the field and when. The binary reputation dataset within WINE provides information about all the binaries detected on end-user hosts by Symantec products such as Norton Antivirus. Each binary reputation record contains the filename, its version, a file hash, machine ID, and a timestamp for the detection event.

The intrusion-prevention telemetry dataset within WINE provides information about network based attacks detected by Symantec products.We define a network-based attack against a host as a series of network packets that: 1) carry malicious code, and 2) have not been prevented by other existing defenses (e.g. network or OS firewall). Each IPS entry contains the signature ID for the threat detected, a machine ID, a platform string and a timestamp for the event. Our study involves 298,851,312 IPS entries corresponding to 6,346,104 hosts over a period of 4 years. The average duration for which a host is present in our study is approximately 13 months.

4.2 Data analysis approach

The primary requirement for our analysis is to detect an attempt to attack a host running a vulnerable product. IPS telemetry does not indicate whether the vulnerabilities had been patched at the time of the attack. Moreover, these reports indicate attacks that were blocked by the IPS product, rather than successful infections. We exclude attacks against products that are not installed on a host, as they would not result in a successful infection.

Finding signatures used to exploit products. Using NVD, we first collect the disclosure dates and the vulnerable software list for all vulnerabilities—including vendor, product and version—and manually remedy any naming inconsistencies. We join this information with Symantec's attack signatures containing CVE numbers, so as to obtain entries of the form $\langle CVE, Prod, Sign \rangle$.

Selecting the hosts used in the study. We analyze Symantec's binary reputation reports in order to determine, for each host, what products are installed and the period of time during which they remained installed. We manually map the binaries to the corresponding product versions using externally available information. For instance, `iexplore.exe` corresponds to IE, and file version 8.0.6001.18702 corresponds to IE 8 [26]. Due to users enabling different features of their Symantec product at different times, the time period for which we have binary reputation data for a host may be different from the time period for which we have IPS telemetry reports (attack detections). Our metrics require both product presence and attack information, so we include hosts in our study only during the times when they are submitting both kinds of data.

Identifying an attack against a host. By joining the binary reputation and IPS dataset for all the hosts identified, we are able to discover all the products installed on a host and all the signatures triggered on the machine. We derive the operating system installed on a host from the platform string included in IPS telemetry submissions. If a vulnerable product is present when the corresponding signature is detected, we have an attack on the host. Of the 298,851,312 telemetry records, 40,954,812 correspond to one of the vulnerabilities that we study, and of these, 20,915,168 occur on a host with the vulnerable product installed.

4.3 Products analyzed

At the time of this writing, NVD includes 61,387 vulnerabilities reported across all products. There are Symantec signatures corresponding to 1,406 of these vulnerabilities. We focus our study on Windows operating systems and on several applications that run on this platform because (i) they have been the primary target for cyber attacks over the past 10 years, and (ii) the platform has evolved in these 10 years and the versions we investigate incorporate considerable diversity, as many technical changes have been implemented between Windows

XP and Windows 7 SP1.[2] In particular, in this paper, we study the following software products:

- Microsoft Windows: XP (original, SP1, SP2, SP3), Vista (original, SP1, SP2), and 7 (original, SP1)
- Microsoft Office (referred to as Office) - Versions: 2000, 2003, 2007, 2010
- Internet Explorer (referred to as IE) - Versions: 5, 6, 7, 8
- Adobe Reader (referred to as Reader) - Versions: 5, 6, 7, 8, 9, 10, 11

There are 860 vulnerabilities in NVD for all the Windows operating systems we consider. Out of these, 132 vulnerabilities have seen exploits in the wild, according to Symantec's attack signatures. We exclude the 64-bit version of XP from our study as it belongs to the same product line as Windows Server 2003. There are 759 vulnerabilities reported for the versions of IE we consider. 108 of these vulnerabilities have seen exploits in the wild. For Office, 50 of the 163 vulnerabilities reported have been exploited. Finally, there are 337 vulnerabilities for the versions of Reader we analyze, out of which 44 vulnerabilities have seen exploits in the wild.

4.4 Threats to validity

The biggest threat to the validity of our results is selection bias. The two databases we employ to characterize vulnerabilities and exploits, NVD and Symantec's attack signatures, respectively, may be incomplete. These databases include only the vulnerabilities and exploits that are known to the security community. Moreover, as WINE does not include telemetry from hosts without Symantec's anti-virus products, our field-data based measurements may not be representative of the general population of platforms in the world. In particular, users who install anti-virus software might be more careful with the security of their computers and, therefore, might be less exposed to attacks.

Although we cannot rule out the possibility of selection bias, we note that the NVD is generally accepted as the authoritative reference on vulnerabilities, and it is widely employed in vulnerability studies, and prior work found Symantec's signatures to be the best indicator for which vulnerabilities are exploited in real-world attacks [19]. Moreover, the large size of the population in our study (six million hosts) and the diversity of platforms suggest that our results have a broad applicability. However, we caution the reader not to assume that all systems will react in the same manner to malware attacks.

[2] While we do not know the amount of code these OSes have in common, it is widely accepted that a large amount of new code was introduced in Windows Vista, including security technologies such as software data execution prevention (DEP/SafeSEH), address space layout randomization (ASLR), driver signing improvements, user account control (UAC), or the Windows filtering platform.

5 Analysis of exploited vulnerabilities and exercised attack surfaces

In this section, we evaluate the metrics introduced in Section 3 and discuss their implications. In particular, we focus on the following questions: *"How many of the disclosed vulnerabilities get exploited?"* (§5.1), *"How often do they get exploited?"* (§5.2), and *"When do they get exploited?"* (§5.3). The first question evaluates our vulnerability-based metrics under real-world conditions, while the second and third questions investigate our attack-based metrics and the deployment-specific factors that affect them.

5.1 How many vulnerabilities get exploited?

Exploitation ratio. Table 2 shows the number of exploited vulnerabilities and the exploitation ratio for all OSes and products in our study. The exploitation ratios shown include vulnerabilities disclosed and exploited as of the end of the product's support period, or as of 2014 if the product is presently supported. We account for *progressive* and *regressive* vulnerabilities [14] separately. A progressive vulnerability is a vulnerability discovered in version N that does not affect version $N-1$ or previous versions, while a regressive vulnerability is one found in version N that affects at least one of the previous versions. The progressive-regressive distinction is important for evaluating the software development process and for understanding the security of the new code added in each version—even though, from the users' point of view, it is important to study all the vulnerabilities that affect a product version. The table also includes the exploitation prevalence, EP^p, which helps to illuminate how likely a host is to experience an attack if a given product is installed. EP^p is defined as the proportion of the hosts with product p installed that experienced at least one attack targeting one of p's vulnerabilities. Note that this metric captures information not revealed by the exploitation ratio or the number of exploited vulnerabilities. For instance, Reader 9 has the same number of exploited vulnerabilities as IE 8, but its exploitation prevalence is far higher.

In aggregate, over all the software products we analyzed, about 15% of the known vulnerabilities have been exploited in real-world attacks. Note, however, that the exploitation ratio varies greatly across products and between versions of a product. This highlights the pitfall of employing the number and severity of vulnerabilities as a measure of security: a product with many high-impact vulnerabilities in NVD would be considered insecure, even if its exploitation ratio is lower than for other products. To further investigate whether the total count of vulnerabilities models the security of a software product, we compare the distributions of the disclosed and exploited vulnerabilities for each product using the Kolmogorov-Smirnov test [27]. The results suggest that we cannot reject the null hypothesis that the number of vulnerabilities and the number of exploits are drawn from the same distribution, at the $p = 0.05$ significance level, for any of the products studied. However, some differences stand out. For example, IE 5 has nearly three times as many reported vulnerabilities as Office 2000. Nevertheless,

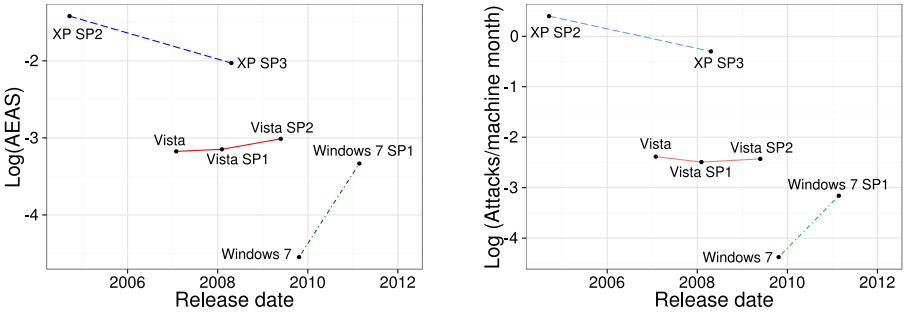
Table 2. Exploitation ratio and exploitation prevalence of products. $ER(yr)$: exploitation ratio of the product for all vulnerabilities up to the year yr. EP_P: the ratio of machines experiencing an attack over the number of machines having the product installed. NA indicates that no machines in WINE had the product installed.

yr	Product	$ER^p(yr)$	$ER^p_{Prog}(yr)$	$\left\|V_p^{ex}\right\|$	$\left\|V_p^{prog,ex}\right\|$	EP^p
2006	IE 5	0.12	0.14	27	25	NA
2010	IE 6	0.17	0.16	73	33	0.035
2013	IE 7	0.13	0.07	36	4	0.002
2013	IE 8	0.13	0.15	29	10	0.0004
2009	Office 2000	0.32	0.32	27	27	NA
2013	Office 2003	0.35	0.36	43	21	0.0002
2013	Office 2007	0.27	0.18	18	2	0
2013	Office 2010	0.25	0	5	0	0
2009	Windows XP	0.21	0.15	39	8	NA
2006	Windows XP SP1	0.28	0.31	41	11	0.026
2010	Windows XP SP2	0.23	0.27	73	16	0.011
2014	Windows XP SP3	0.13	0.07	58	12	0.047
2012	Windows Vista	0.21	0.09	39	5	0.005
2011	Windows Vista SP1	0.16	0.06	40	6	0.004
2014	Windows Vista SP2	0.11	0.06	39	2	0.011
2014	Windows 7	0.07	0.25	20	2	0
2014	Windows 7 SP1	0.07	0	15	0	0.004
2008	Adobe Reader 5	0.18	0.2	4	1	NA
2008	Adobe Reader 6	0.22	0.17	5	1	NA
2009	Adobe Reader 7	0.17	0.09	11	4	0.177
2011	Adobe Reader 8	0.16	0.15	29	18	0.180
2013	Adobe Reader 9	0.11	0.10	29	10	0.242
2014	Adobe Reader 10	0.08	0.04	13	1	0.0002
2014	Adobe Reader 11	0.06	0	5	0	0

both have a similar number of exploited vulnerabilities. This is reflected in the much higher exploitation ratio for Office. This is one example of how field-gathered data which reflects the deployment environment can complement more traditional security metrics.

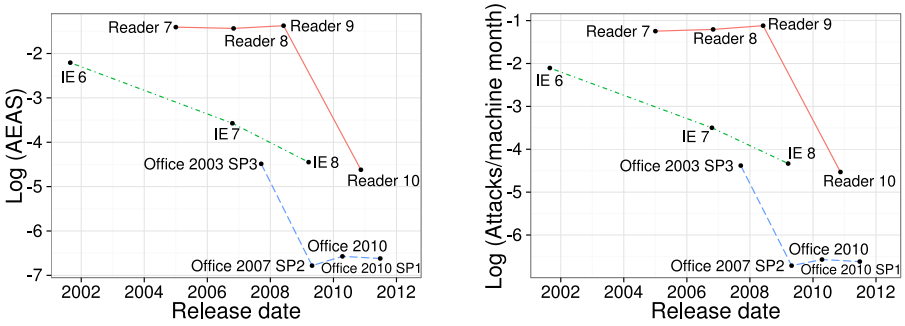
Another trend visible in Table 2 is that the latest versions of each product have a lower absolute number of exploited vulnerabilities than earlier versions (except in the case of IE). For instance, Windows 7 has fewer exploited vulnerabilities than Windows Vista, and Reader versions 10 and 11 have fewer than versions 8 and 9. One factor that has likely contributed to this decrease is the introduction of security technologies by Microsoft and Adobe that make exploits less likely to succeed, even in the presence of vulnerabilities (e.g., address space layout randomization and sandboxing). Another likely contributing factor is the commoditization of the underground malware industry, which has led to the marketing of exploit kits that bundle a small number of effective attacks for wide-spread reuse.

Time-to-exploit. The decrease in the number of exploited vulnerabilities over time could be caused by the fact that cyber attackers have had less time to find ways to exploit newer products. To investigate the influence of this confounding factor, we estimate the typical time that attackers need to exploit vulnerabilities in Windows after they are publicly disclosed. While the mention of these vulnerabilities in Symantec's AV and IPS signatures is an indication that an



(a) Average exercised attack surface. (b) Attack volume.

Fig. 2. Exercised attack surface and attack volume for Windows operating systems.



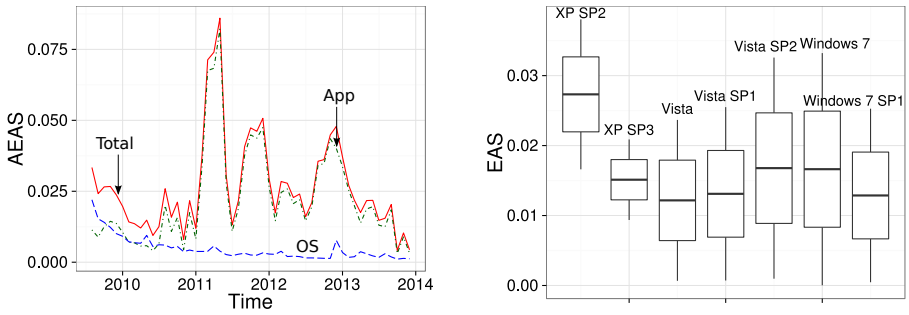
(a) Average exercised attack surface. (b) Attack volume.

Fig. 3. Exercised attack surface and attack volume for Windows applications.

exploit was, at some point, used in the real world, it is challenging to estimate the time that elapses between the vulnerability disclosure and the release of the exploit. We estimate the time-to-exploit using a combination of three methods: (i) the "exploit published" date from OSVDB; (ii) the "discovery date" from the anti-virus signature descriptions; (iii) the date when a signature is first recorded in the field data from WINE. We observe that 90% of exploits from anti-virus signatures and from attack signatures are created within 94 and 58 days after disclosure, respectively. Our observation is consistent with the prior work on this topic, which found that the exploits for 42% of vulnerabilities that are eventually exploited appear in the wild within 30 days after the disclosure date [28,29]. This shows that if a vulnerability is to be exploited, an attack will likely be observed soon after its disclosure.

5.2 How often do vulnerabilities get exploited?

Average exercised attack surface. Figure 2 shows that the average exercised attack surface ($AEAS_p$) and the attack volume (AV_p) for OS vulnerabilities tend



(a) Combined AEAS over time.

(b) Variation of combined AEAS for machines running different OSes.

Fig. 4. Average exercised attack surface (AEAS) due to OS and installed applications combined.

to increase across minor releases of Windows (except XP), but they decrease considerably with each new major release. This could be explained by the fact that, over time, attackers become more familiar with the code and the mechanisms of an OS and they are more effective in finding and exploiting vulnerabilities [14]. However, major releases refactor the code and introduce new security technologies that make it more difficult to create exploits. For example, Windows Vista introduced address space layout randomization (ASLR) and data execution prevention (DEP), which render exploits less likely to succeed even if a vulnerability is present.

Figure 3 shows the average exercised attack surface ($AEAS_p$) and attack volume metrics (AV_p) for IE, Reader and Office. For IE, these values decrease with newer releases. Note the precipitous drop in exercised attack surface between Reader 9 and Reader 10 (three orders of magnitude). This can likely be attributed to *protected mode*, an enhancement that Adobe introduced in Reader 10 specifically to mitigate and prevent security vulnerabilities [30]. We also observe that the exercised attack surface values of Reader (except version 10) are about an order of magnitude higher than those of IE. This is somewhat surprising, since Table 2 shows that the various versions of IE have nearly as many or, in the case of IE 6, far more exploited vulnerabilities than any of the versions of Reader. Taken together, these observations suggest that vulnerabilities in Reader (prior to version 10) have proven easier for cyber criminals to successfully exploit.

We also note that some vulnerabilities affect the volume of attacks disproportionately. For OS vulnerabilities, the number of attacks due to CVE-2008-4250 (the vulnerability exploited by the Conficker worm) is three orders of magnitude higher than the number of attacks due to the next most targeted vulnerability. For product vulnerabilities, the number of attacks due to CVE-2009-4324 (a vulnerability in Adobe Reader) is almost 20× as high the number of attacks due to the next most targeted vulnerability.

Reader	no	no	no	no	yes	yes	yes	yes
IE	no	no	yes	yes	no	no	yes	yes
Office	no	yes	no	yes	no	yes	no	yes
Average attack surface	0.00000	0.00002	0.00125	0.00068	0.02823	0.03130	0.03009	0.03195

Table 3. Average exercised attack surface in the presence of products.

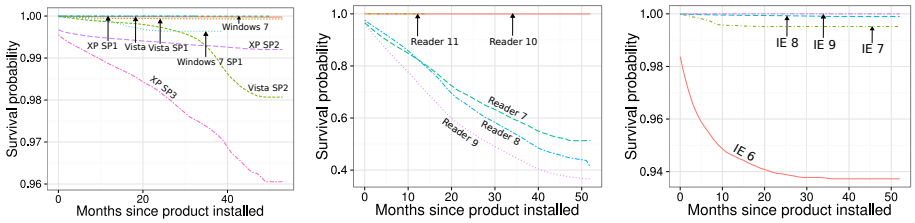
Variation of exercised attack surface. Figure 4a shows the variation of the average exercised attack surface ($AEAS(m)$) over time. Notice that application vulnerabilities contribute to most of the attack surface, which suggests that the OS vulnerabilities are more difficult to exploit. Also, we see two spikes in the 2011-2012 time frame and another spike towards the end of 2012. These spikes are correlated with the attacks exploiting CVE-2009-4324, which account for the higher exercised attack surface measurement at those times. This illustrates the fact that, even against a background of diverse attacks, a single vulnerability that is attacked heavily can increase the average attack surface by reaching more hosts. Figure 4b shows the variation of exercised attack surface across hosts running the same operating system. We note that, in this case, both OS and application vulnerabilities contribute to the exercised attack surface. Thus, even if some hosts are running the same operating system, their exercised attack surface varies considerably based on the products installed on the system.

Impact of vulnerabilities on attack surface. Each host may have one or more products installed. Moreover this may change over time. Although it is hard to quantify the impact of specific vulnerabilities on a particular host, we can calculate the average exercised attacked surface for hosts with different combinations of the products in question. Table 3 shows the average exercised attack surface for hosts in the presence of IE, Reader, and Office. When none of the products in question is present, the average exercised attack surface is 0, since we have eliminated the effect of OS vulnerabilities, thus measuring only the impact of the products on the calculated attack surface. Furthermore, the calculation of each average attack surface value only considers hosts that have the exact product combination installed, and these hosts are not reconsidered for the calculation of any of the remaining values. We observe that the vulnerabilities present in each of the products have different impact on the average exercised attack surface. For instance, the telemetry data at our disposal suggest that the presence of Reader has a higher impact on the average attack surface of hosts than the presence of IE or Office.

How much can we reduce the attack surface? The overwhelming prevalence of attacks using CVE-2008-4250 and CVE-2009-4324, and the variability in the size of the exercised attack surface among the hosts in our study, prompt the question of whether attack surface reduction methods could be used to reduce the overall risk. In the case of application vulnerabilities, a user may reduce the attack surface by uninstalling the vulnerable product. For OS vulnerabilities, the user would have to disable the vulnerable component. We manually inspect the NVD entries for the top-10 OS vulnerabilities so as to determine if

CVE	# of Attacks	# of Hosts	Affected Operating Systems	Can be turned off?	NIST severity
2008-4250	17915163	66408	XP SP2/SP3/SP2 64 bit, Vista Original/SP/SP1, Server 2003 SP1/SP2	no	10
2006-3439	55998	2577	XP SP1/SP2/SP2 64 bit	yes	10
2011-3402	40598	21717	XP SP3, Vista SP2, Windows 7 SP1, Server 2003 SP2	no	9.3
2010-1885	36984	22337	XP SP3/SP2 64 bit	yes	9.3
2010-0806	25280	15539	XP SP3, Vista Original SP/SP1/SP2	no	9.3
2009-2532	20343	756	Vista Original/SP1/SP2, Server 2008 SP2	yes	10
2009-3103	20343	756	Vista Original/SP1/SP2, Server 2008 SP2	yes	10
2008-0015	6105	3897	XP SP2/SP3	yes	9.3
2010-2568	2182	360	XP SP3/SP2 64 bit, Vista SP1/SP2	no	9.3
2012-0003	93	71	XP SP2/SP3, Server 2003 SP2, Vista SP2, Server 2008 SP2	yes	9.3

Table 4. Most attacked OS vulnerabilities.



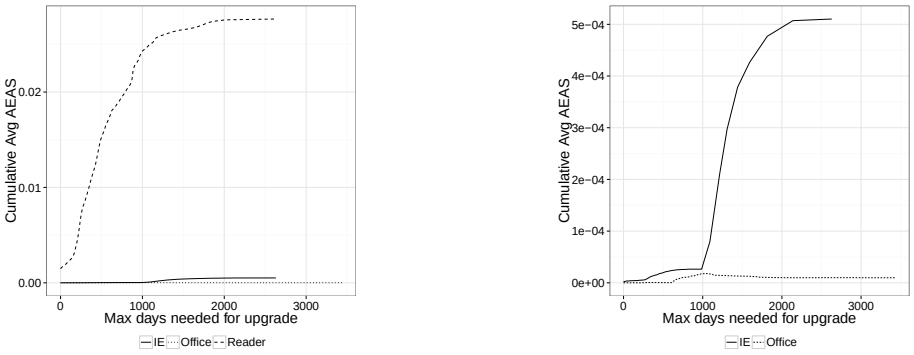
(a) Windows. (b) Reader. (c) IE.

Fig. 5. Time from installation to first attack.

the vulnerable component can be disabled, while keeping the host operational. As seen in Table 4, 6 out of 10 of the intrusion vectors correspond to vulnerable services or components that could be disabled (assuming that the relevant service/functionality is not necessary), suggesting that there is potential for further reduction of the OS attack surface. Notice, however, that in certain cases the components that would need to be disabled or removed may severely affect the functionality of the system. To better understand the potential to improve security by disabling vulnerable components, we consider the volume of attacks for each vulnerability in Table 4. If we exclude the skew introduced by Conficker, we observe that the number of remaining attacks could be reduced by 67.3%, thus significantly reducing the size of the exercised attack surface.

5.3 When do vulnerabilities get exploited?

In this section we explore several time-related aspects of attacks. We presented the exploitation prevalence EP^p, which indicates how many of the hosts that install a product experience at least one attack due to that product, in Table 2. We now explore *when* that attack happens with respect to the installation of the product. Figure 5 plots the estimated survival probability versus the number of months since installation, where survival means not having experienced any attack targeting one of the product's vulnerabilities. Our data set is right censored, meaning that some hosts leave our study (stop sending telemetry) without



(a) All products.

(b) Detail of IE and Office.

Fig. 6. Effect of product upgrade lag on a host's exercised attack surface ($AEAS_h^p$).

having experienced an attack; in this case, we know the period when these hosts were attack-free, but we don't know when they will experience their first attack. In statistical terms, these data points are right-censored. We estimate the survival probability using the Kaplan-Meier estimator [27], which accounts for censoring. The survival probability for most versions of Windows is nearly one even after four years of installation. Windows XP SP3 experiences more exploits, and so its survival probability drops with a fairly smooth slope down to about 0.96 after four years. The smooth slope seems to indicate that installation age is not strongly correlated with the likelihood of experiencing your first attack. Reader versions 7 through 9 also show a fairly smooth, though much steeper, drop in survival probability. Again, this indicates that a user has a similar probability of experiencing a first attack against Reader whether it's been installed for one month or ten months, though the slope does tend to decrease slightly as we move beyond twenty months. Hosts with IE 6 installed appear to have a higher probability of experiencing their first attack within the first ten months, after which the survival probability levels out with an estimated 94% of hosts experiencing no attack after four years. The plot for Office is not shown since all versions maintain over 99.9% survival for the entire period.

We now explore another time-related question: Does the length of time a user waits to upgrade after a new version comes out have an impact on exercised attack surface? To help answer this, we introduce the upgrade lag metric. In order to measure a user's upgrade lag with respect to a product version, we look at how long a user continues to use the version after a new version of the product is released. The user's upgrade lag for the product line is calculated as the maximum upgrade lag over all versions of the product. Note that even if only a single version of a product is ever installed on a machine, the upgrade lag is still defined. If a machine has only Reader 9 installed, and it is present for a period of time entirely before Reader 10 is released, the machine's upgrade lag for Reader is 0. If a machine has only Reader 9 installed, and it is present

from one month after Reader 10 is released until six months after Reader 10 is released, then the upgrade lag would be six months.

In Figure 6a, we plot the upgrade lag in days versus the cumulative average exercised attack surface (that is, the average $AEAS_h^p$ for all hosts with a lag less than or equal to a given lag) for each of the product lines. So, for instance, if you consider all hosts that have a maximum upgrade lag for Reader that is less than or equal to 500 days, on average their AEAS with respect to Reader is about 0.015 vulnerabilities per machine per month. The steady increase in the curve for Reader from zero upgrade lag until 1000 days of upgrade lag is a strong indication that machines that wait longer to upgrade Reader tend to experience more attacks. In Figure 6b, we zoom in to show the detail for IE and Office. For both IE and Office, we see a modest increase in exercised attack surface as upgrade lag increases. The sharp increase in the slope of the IE curve after 1000 days can be explained by the fact that roughly 6% of hosts with IE installed have the most vulnerable version, IE 6, installed long after IE 7 was released. These hosts, which account for over 90% of attacks against IE, cause the rapid increase after 1000 days. These results suggest that the upgrade lag is one factor that affects the attack surface in the deployment environment.

6 Discussion

In this paper, we propose several metrics for assessing the security of software products in their deployment environments. For example, we observe that, for most products, the exploitation ratio and/or the number of exploited vulnerabilities decrease with newer versions. Interestingly, anecdotal evidence suggests that cyber criminals are starting to feel the effects of this scarcity of exploits. While zero-day exploits have traditionally been employed in targeted attacks [28], in 2013 the author of the Blackhole exploit kit advertised a $100,000 budget for purchasing zero-day exploits [31]. The zero-day exploit for CVE-2013-3906 was nicknamed the "dual-use exploit" after being employed both for targeted attacks and for delivering botnet-based malware [32].

Qualitative analysis. While the coexistence of several security mechanisms in a product prevents us from measuring the individual impact of each of these mechanisms, it is interesting to note that improvements in our metrics are often associated with the introduction of system security technologies. Improved security was a primary design goal for Windows Vista and we find a decrease in the number of progressive exploited vulnerabilities in Windows Vista and Windows 7. This seems to be associated with the introduction of security technologies like ASLR, DEP, User Account Control and the concept of integrity levels. Among products, there is a notable decrease in the exploitation ratio and number of exploited vulnerabilities in IE 7 and Reader 10. Both these products started running the application in a sandbox, which adds an additional layer of defense by containing malicious code and by preventing elevated privilege execution on the user's system [30]. IE 7 also removed support for older technologies like

DirectAnimation, XBM, DHTML editing control in an attempt to reduce the surface area for attacks [33].

Operational utility of the proposed metrics. Our product-based metrics can be integrated in security automation frameworks, such as SCAP [16]. For example, knowing which vulnerabilities are exploited in the wild will allow system administrators to prioritize patching based on empirical data, rather than relying exclusively on the CVSS scores for this task. The exploitation ratios of different products can be incorporated in quantitative assessments of the risk of cyber attacks against enterprise infrastructures. The ability to determine whether a few exploits are responsible for most of the recorded attacks (as in the case of Conficker) will allow security vendors to focus on these vulnerabilities for reducing the volume of attacks against critical infrastructures in an efficient manner.

Our host-based metrics would be useful in infrastructures where not all hosts can be centrally managed, such as in enterprises that have *bring-your-own-device* (BYOD) policies. For example, the exercised attack surface metric captures the diversity of attacks against a host. This metric varies considerably from host to host, depending on the software installed and on the user behaviors; in particular, the exercised attack surface is correlated with a host's product upgrade lag. This information will allow administrators to subject the hosts more likely to be attacked to a higher level of scrutiny.

Agenda for future research. Our results illustrate the fact that, in the deployment environment, security is affected by factors that cannot be accounted for in the lab. Further research is needed to explore the opportunities for deriving security metrics from field data. For example, it is difficult to assess whether a single, potentially successful, attack exploiting vulnerability X is more or less devastating than a large number of attacks exploiting vulnerabilities other than X. As another example, the exercised attack surface metric cannot adequately capture the effects of a single powerful attack on a long-lived host H, since the effect of the attack on ES_H will be diluted by the amount of time H is under observation. To address this problem, we could define an *attack watermark* metric, which would represent the average number of unique vulnerabilities of a product P that are attacked on hosts running P during our observation period.

7 Conclusions

We believe that our ability to improve system security rests on our understanding of how to measure and assess security under real-world conditions. In this paper we analyze a large data set of security telemetry, available to the research community, to 1) expose trends in the exploitation of vulnerabilities, and 2) propose new field-measurable security metrics, capable of capturing the security of systems in their deployment environments, rather than in lab conditions. We focus on nine versions of the Windows operating system, and multiple versions of three popular applications. Our findings reveal that, combining all of the products we

study, only 15% of disclosed vulnerabilities are ever exploited in the wild. None of the studied products have more than 35% of their vulnerabilities exploited in the wild, and most of these are exploited within 58 days after disclosure. We show that the number of vulnerabilities in a product is not a reliable indicator of the product's security, and that certain vulnerabilities may be significantly more impactful than others. Furthermore, we observe that, even though the security of newer versions of Windows appears to have improved, the overall exposure to threats can be significantly impacted by "post-deployment" factors that can only be observed in the field, such as the products installed on a system, the frequency of upgrades, and the behavior of attackers. The impact of such factors cannot be captured by existing security metrics, such as a product's vulnerability count, or its theoretical attack surface. To address this, we introduce new, field-measurable security metrics. The count of vulnerabilities exploited in the wild and the exploitation ratio aim to capture whether a vulnerability gets exploited. The attack volume and exercised attack surface metrics aim to measure the extent to which hosts are attacked. Finally, the calculated survival probabilities and our study of the impact of software upgrades to security aim to reveal real-world temporal properties of attacks. These metrics can be incorporated in quantitative assessments of the risk of cyber attacks against enterprise infrastructure, and they can inform the design of future security technologies.

Acknowledgments

We thank the anonymous RAID reviewers for their constructive feedback. Our results can be reproduced by utilizing the reference data set WINE-2014-001, archived in the WINE infrastructure. This work was supported in part by a grant from the UMD Science of Security lablet.

References

1. Shin, Y., Meneely, A., Williams, L., Osborne, J.A.: Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Trans. Software Eng. **37**(6) (2011) 772–787
2. Zimmermann, T., Nagappan, N., Williams, L.A.: Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In: ICST. (2010) 421–428
3. National Vulnerability Database `http://nvd.nist.gov/`.
4. Howard, M., Pincus, J., Wing, J.M.: Measuring relative attack surfaces. In: Workshop on Advanced Developments in Software and Systems Security, Taipei, Taiwan (Dec 2003)
5. Manadhata, P.K., Wing, J.M.: An attack surface metric. IEEE Trans. Software Eng. **37**(3) (2011) 371–386
6. Microsoft Corp.: Microsoft Attack Surface Analyzer - Beta `http://bit.ly/AO4NNO`.
7. Coverity: Coverity scan: 2011 open source integrity report (2011)
8. National Institute of Standards and Technology: National Vulnerability database `http://nvd.nist.gov`.
9. Microsoft Corp.: A history of Windows `http://bit.ly/RKDHIm`.

10. Wikipedia: Source lines of code http://bit.ly/5LkKx.
11. TechRepublic: Five super-secret features in Windows 7 http://tek.io/g3rBrB.
12. Rescorla, E.: Is finding security holes a good idea? IEEE Security & Privacy **3**(1) (2005) 14–19
13. Ozment, A., Schechter, S.E.: Milk or wine: Does software security improve with age? In: Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15. USENIX-SS'06, Berkeley, CA, USA, USENIX Association (2006)
14. Clark, S., Frei, S., Blaze, M., Smith, J.: Familiarity breeds contempt: The honeymoon effect and the role of legacy code in zero-day vulnerabilities. In: Proceedings of the 26th Annual Computer Security Applications Conference. ACSAC '10, New York, NY, USA, ACM (2010) 251–260
15. Bozorgi, M., Saul, L.K., Savage, S., Voelker, G.M.: Beyond heuristics: learning to classify vulnerabilities and predict exploits. In: KDD, Washington, DC (Jul 2010)
16. Quinn, S., Scarfone, K., Barrett, M., Johnson, C.: Guide to adopting and using the security content automation protocol (SCAP) version 1.0. NIST Special Publication 800-117 (Jul 2010)
17. Ransbotham, S.: An empirical analysis of exploitation attempts based on vulnerabilities in open source software (2010)
18. Kurmus, A., Tartler, R., Dorneanu, D., Heinloth, B., Rothberg, V., Ruprecht, A., Schröder-Preikschat, W., Lohmann, D., Kapitza, R.: Attack surface metrics and automated compile-time os kernel tailoring. In: Network and Distributed System Security (NDSS) Symposium, San Diego, CA (Feb 2013)
19. Allodi, L., Massacci, F.: A preliminary analysis of vulnerability scores for attacks in wild. In: CCS BADGERS Workshop, Raleigh, NC (Oct 2012)
20. Allodi, L.: Attacker economics for internet-scale vulnerability risk assessment. In: Proceedings of Usenix LEET Workshop. (2013)
21. Symantec Corporation: A-Z listing of threats and risks. http://bit.ly/11G7JE5
22. Symantec Corporation: Attack signatures http://bit.ly/xQaOQr.
23. Open Sourced Vulnerability Database: http://www.osvdb.org.
24. : Symantec Attack Signatures http://bit.ly/1hCw1TL.
25. Dumitraș, T., Shou, D.: Toward a standard benchmark for computer security research: The worldwide intelligence network environment (wine). In: Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security. BADGERS '11, New York, NY, USA, ACM (2011) 89–96
26. : Information about Internet Explorer versions http://bit.ly/1oNMA97.
27. National Institute of Standards and Technology: Engineering statistics handbook http://www.itl.nist.gov/div898/handbook/index.htm.
28. Bilge, L., Dumitraș, T.: Before we knew it: an empirical study of zero-day attacks in the real world. In: ACM Conference on Computer and Communications Security, Raleigh, NC (Oct 2012) 833–844
29. : Microsoft security intelligence report, volume 16 http://download.microsoft.com/download/7/2/B/72B5DE91-04F4-42F4-A587-9D08C55E0734/Microsoft_Security_Intelligence_Report_Volume_16_English.pdf.
30. Adobe Reader Protected Mode http://helpx.adobe.com/acrobat/kb/protected-mode-troubleshooting-reader.html.
31. Krebs, B.: Crimeware author funds exploit buying spree. http://bit.ly/1mYwlUY (2013)
32. FireEye: The Dual Use Exploit: CVE-2013-3906 Used in Both Targeted Attacks and Crimeware Campaigns. http://bit.ly/R3XQQ4 (2013)
33. A Note about the DHTML Editing Control in IE7+ http://blogs.msdn.com/b/ie/archive/2006/06/27/648850.aspx.