Information and Software Technology 58 (2015) 304–318

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A Bayesian network model for likelihood estimations of acquirement of critical software vulnerabilities and exploits

Hannes Holm *, Matus Korman, Mathias Ekstedt

Department of Industrial Information and Control Systems, Royal Institute of Technology, 100 44 Stockholm, Sweden

ARTICLE INFO

Article history:
Received 11 July 2013
Received in revised form 29 June 2014
Accepted 2 July 2014
Available online 15 July 2014

Keywords:
Cyber security
Vulnerabilities
Exploits
Statistical model
Security metrics

ABSTRACT

Context: Software vulnerabilities in general, and software vulnerabilities with publicly available exploits in particular, are important to manage for both developers and users. This is however a difficult matter to address as time is limited and vulnerabilities are frequent.
Objective: This paper presents a Bayesian network based model that can be used by enterprise decision makers to estimate the likelihood that a professional penetration tester is able to obtain knowledge of critical vulnerabilities and exploits for these vulnerabilities for software under different circumstances.
Method: Data on the activities in the model are gathered from previous empirical studies, vulnerability databases and a survey with 58 individuals who all have been credited for the discovery of critical software vulnerabilities.
Results: The proposed model describes 13 states related by 17 activities, and a total of 33 different datasets.
Conclusion: Estimates by the model can be used to support decisions regarding what software to acquire, or what measures to invest in during software development projects.

1. Introduction

Software engineering is a complex activity that deals with limited resources [8]. In practice, this often results in faults that may cause software to exhibit unexpected and undesired behavior [18]. A particularly troubling class of faults are those that can be exploited by motivated actors to compromise assets, for instance, to obtain privileges of computer systems. If this is the case, the fault can be considered a *security vulnerability* [7]; an *exploit* is a software designed to enable its utilization.

Software vulnerabilities in general, and software vulnerabilities with publicly available exploits in particular, are essential to properly manage as their exploitation can severely impact the confidentiality, integrity, and availability of enterprise data and services. Various solutions, both technical (e.g., type-safe programming interfaces) and psychological (e.g., awareness training), have been proposed for mitigating software vulnerabilities. Yet, new issues are discovered in commodity software every day. Consequently, current practices are not sufficient.

As vulnerabilities are frequent, and the resources available for their management are limited [36], a method for measuring the relative security of a software would be beneficial; this would enable acquisition of the software that requires the least amount of vulnerability management.

Cyber security investments are typically made by some kind of enterprise decision maker, often a chief information security officer (CISO) [19]. Goodyear et al. [19] survey the properties of CISOs and finds that their responsibilities require more social and political skills than technical skills. Consequently, a CISO cannot be expected to have a deep understanding of software vulnerabilities and exploits, or time to spend on their management. Tools that could support the CISO with this task would thus be of value, even if they come with a certain degree of uncertainty. They could also support decision makers of software engineering projects [39] in regard to what practices and measures to invest in.

This paper describes a Bayesian network [17] based model that can be used to estimate the likelihood that a professional penetration tester is able to obtain knowledge of critical software vulnerabilities and exploits for these vulnerabilities under different conditions. The purpose of the model is to provide insight into how likely different activities of vulnerability and exploit acquisition are to succeed given different circumstances, and provide practitioners with a model that can be used to estimate the availability of vulnerabilities and exploits for their software. Furthermore, this research recognizes that management of individual software vulnerabilities only constitutes a very small part of the total workload for a decision maker in the industry (e.g., a CISO).

* Corresponding author.
E-mail address: hannesh@ics.kth.se (H. Holm).

http://dx.doi.org/10.1016/j.infsof.2014.07.001
0950-5849/

Nomenclature

Abbreviations and Acronyms

AIC	Akaike Information Criterion
CDF	Cumulative Distribution Function
CPE	Common Platform Enumeration
CPT	Conditional Probability Table
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
EXP	Exponential distribution
GAM	Gamma distribution
LI	Linear Interpolation
LN	Log-normal distribution
N	Normal distribution
NVD	US National Vulnerability Database
OSVDB	Open Source Vulnerability Database
PAR	Pareto distribution
PDF	Probability Density Function
WBL	Weibull distribution

Thus, a requirement for the model is that it should require little effort to use.

The remainder of the paper is structured as follows: Section 2 presents related work. Section 3 presents the overall characteristics of the proposed model. Section 4 describes the method used to evaluate what statistical model that is best fit for modeling each dataset. Section 5 describes the data employed by the model. Section 6 describes how the model has been implemented to allow estimations and illustrates its use trough an example. Section 7 discusses the model and Section 8 concludes the paper.

2. Related work

A group of models, often referred to as Vulnerability Discovery Models (VDMs) [4], have been proposed for predicting the occurrence of software vulnerabilities. Alhazmi and Malaiya [3] propose an S-shaped VDM to predict occurrence of vulnerabilities (originally only for operating systems, later for software in general). The S-shape is chosen as the authors believe that vulnerability discovery begins with a long learning phase, when software testers learn how the software functions. This phase is then followed by a linear accumulation phase, when testers are familiar with it and many vulnerabilities are discovered. The third phase concerns when the software has been replaced by newer variants, leading to a smaller user-base and less attracted testers. Similar models are proposed by Kim et al., Woo et al. and Joh et al. [29,44,26].

VDMs are useful for understanding the different phases of the software vulnerability life-cycle. However, they are not particularly helpful for enterprise decision makers as they require data from the majority of a software's life-cycle to be effective (when such information is available the software is likely not employed anymore and few novel vulnerabilities are discovered). They furthermore do not consider the impact of different measures on the occurrence of vulnerabilities (e.g., code analyzers); i.e., they do not consider the effectiveness of different means of improving software security.

Other models concern measuring the security of systems in operation; significant such models are described next.

McQueen et al. [31] propose a Markovian model for estimating the time required to compromise (TTC) a computer system through its vulnerability. The authors suggest that TTC can be described as a random process model that depends on three sub-processes associated with attacker actions aimed at the exploitation of vulnerabilities. The first sub-process is the scenario in which an attacker has a readily available exploit for one or more existing vulnerabilities. The second sub-process is a scenario in which there are one or more vulnerabilities on a system, but the attacker does not have readily available exploits for any of them. The third sub-process corresponds to a scenario in which there are no existing vulnerabilities for the system. Each of these scenarios has a probability of occurrence and an estimated mean time to be completed. For sub-process two and three, an attacker can either wait for an applicable exploit or vulnerability to be disclosed in the public domain, or manually attempt to devise/probe for a new one.

Similarly to [31], NetSPA [25], MulVAL [24], and the CySeMoL [42] also aim to provide estimates on the difficulty of successful cyber attacks; however, without measuring the time until success. NetSPA creates a network model, computes its network reachability and identifies possible attack paths based on firewall rules and network vulnerability scans. To accomplish this, NetSPA treats all vulnerabilities as exploitable by an attacker. MulVAL is similar to NetSPA in the sense that it also predicates its estimates on a network model based on vulnerability scans; however, it assumes that each vulnerability is exploitable with a certain probability of success. CySeMoL also bases its estimates on a network model and computes the probabilities corresponding to attack success. However, it covers a larger selection of (less granular) attacks and defenses than NetSPA or MulVAL. All three tools fail to include various methods that an attacker can employ to gain exploits and vulnerabilities. For instance, that an attacker can wait for new exploits or vulnerabilities to be released [31], or simply purchase them [33].

To enable statistical analysis, security estimation models such as [31,42] require data that describe the various activities involved.

One source of such data is archival data on disclosed vulnerabilities and exploits. A study on this topic was conducted by Shahzad et al. [41] who performed an exploratory study of archival vulnerability data spanning over 23 years, investigating a number of aspects such related to the life cycle of vulnerabilities (e.g., vulnerability disclosure and exploit release dates). In a similar more recent study, Marconato et al. [30] analyzed the distribution of time between events of the life cycle of a vulnerability (where all events were found to follow the Beta distribution) and propose a probabilistic model evaluating security risks faced by a system in operation. These studies thus hint towards the expected time required to wait for a vulnerability or exploit to be publicly released. Unfortunately, they do not consider the release dates of vulnerability, exploit, and patch in combination. In other words, they do not consider whether a vulnerability is patchable upon the release of its corresponding exploit or not.

Another source of data are empirical studies on the exploitation process. Jonsson and Olovsson [27] studied attacker behavior through an experiment, during which student teams attacked a distributed UNIX system. The authors found that the working time for the 59 observed breaches followed the exponential distribution. Holm et al. [22] studied records of an international cyber defense exercise, comparing different software vulnerability metrics against TTC values of successful attacks. Holm [21] studied all cyber intrusions that occurred across more than 260,000 computer systems over a period of almost three years. Using this dataset, the author studied the time from installation of a computer to the first intrusion of that system, and the time between intrusions of a system. The author tested six continuous distributions – the exponential, log-normal, normal, Weibull, gamma and the generalized

Pareto – and found that the Pareto was best at modeling time to first intrusion and the log-normal at modeling time between intrusions. Unfortunately, these studies only take into account a few steps included in the model presented in this study, and under rather uncertain conditions.

In summary, there are various models measuring the occurrence of vulnerabilities and the difficulty of performing cyber attacks, and various empirical studies used to provide data to these models. However, there is as of yet no reliable and holistic model describing how an attacker can gain knowledge of software vulnerabilities and exploits for these vulnerabilities under different scenarios. This is something that the current paper seeks to mitigate.

3. Studied model

This research presents a model that can be used to estimate the likelihood of an attacker obtaining knowledge of software vulnerabilities and exploits for these vulnerabilities under different circumstances. An overview of the states (A-M) and activities (1–15) in this model can be seen in Fig. 1. This model is based upon the activities within the model by McQueen et al. [31], but differs regarding five primary attributes:

- It does not assume postulated probability distributions. Instead, it empirically examines what probability distribution that is best fit for modeling each activity (cf. Section 4).
- It includes vulnerability and exploit purchasing [33].
- It details the availability of vendor-issued software patches mitigating vulnerabilities. If no patch exist, then the only means of preventing an attack exploiting it are by employing intrusion detection systems (that generally are prone to false negatives), by disabling the vulnerable functionality (that often is not viable), or by manually patching the flaw (that typically is not possible given practical resource constraints, especially for closed-source software).
- Its estimates depend on the characteristics of the targeted software.
- It includes estimates by professional penetration testers on the effort required to discover vulnerabilities and develop exploits, rather than assuming that this is related to the disclosure dates of archival vulnerabilities and exploits.

The origin state of the model is that the attacker has decided to obtain knowledge of vulnerabilities and exploits for these vulnerabilities for a certain software (state A). This state can transition into

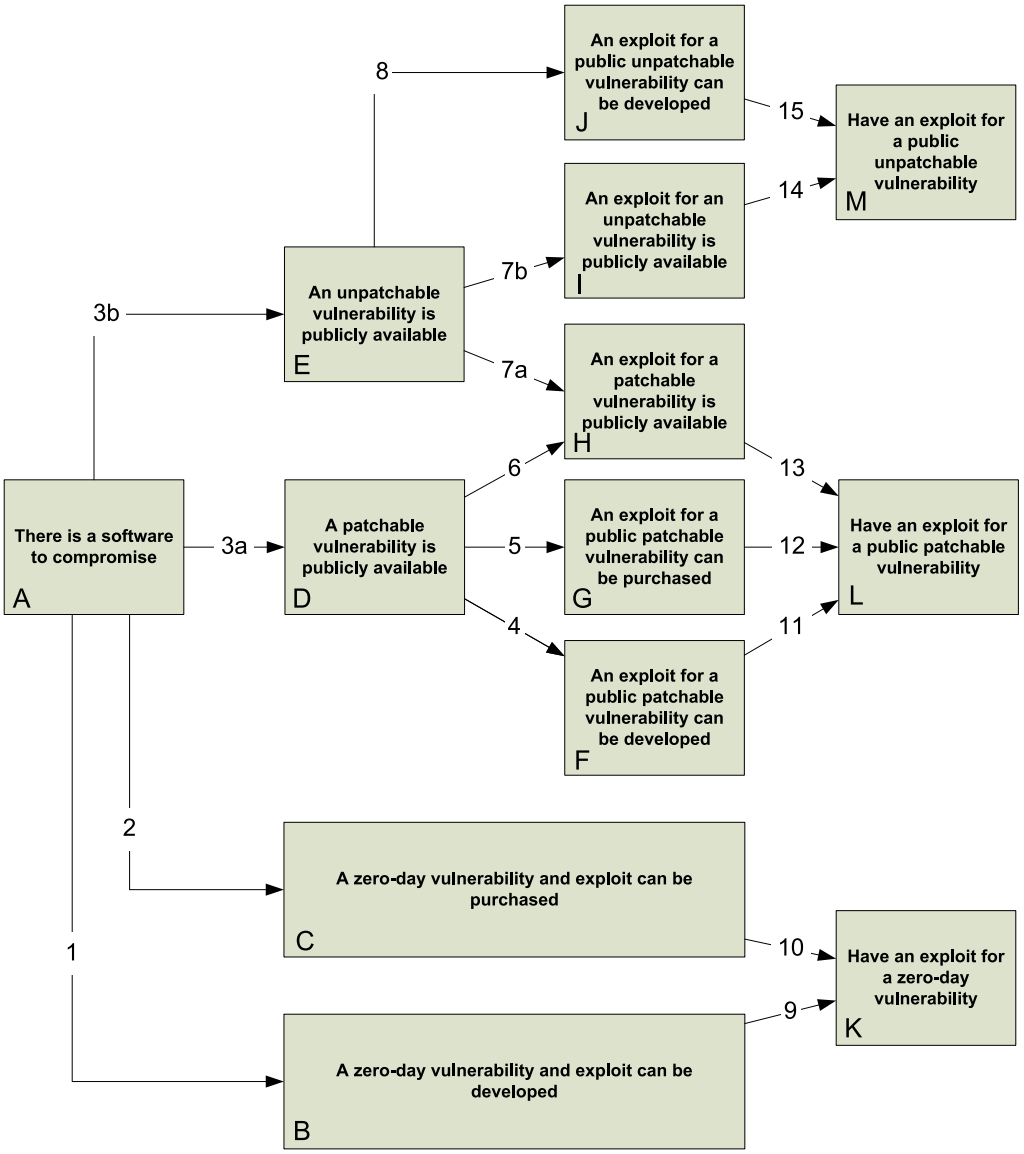

Fig. 1. A statistical model for methods of obtaining software vulnerabilities and exploits.

four states: discovery of zero-day (state B), purchase of zero-day (state C), finding information about a public patchable vulnerability (state D), and finding information about a public unpatchable vulnerability (state E). The attacker has an exploit for a zero-day vulnerability (state K) if state B or state C is true. If the attacker finds information about a public vulnerability, there is a possibility that an exploit can be developed (state F, J) or purchased (state G). Its exploit could also be publicly released (state H, I). If state F, G or H is true, the attacker has an exploit for a patchable vulnerability (state L); if state I or J is true, the attacker has an exploit for an unpatchable vulnerability (state M). The likelihood of each state being true depends on the likelihood of its related activities (or steps) being true; these in turn depend on the characteristics of the software and the resources of the attacker. All of these aspects are related as a Bayesian network and elaborated in Section 5.

The relation between states E and H (activity 7a) might seem confusing as the release of a patch logically should result in a transition from state E to state D. The reason for this design is that states D and E correspond to highly different vulnerability disclosure activities: In state D, where a patch is available at the time of disclosure, the vulnerability has (most likely) undergone a responsible disclosure process, perhaps due to having been discovered by the software developers themselves. In state E, where the vulnerability is disclosed but no patch is available, it is more likely the outcome of an irresponsible disclosure process. Thus, the time from vulnerability to exploit for these two scenarios could be significantly different (no matter if a patch is issued or not). As a consequence, activity 7a goes from state E to state H rather than from state E to state D (this reasoning is supported by the data itself, cf. Section 5.6).

To enable statistical estimations on the likelihood of the states in the studied model (cf. Fig. 1) being true, there is a need to populate all activities with relevant data. An overview of the data used for this purpose can be seen in Table 1. Data from previous academic studies were used where possible (activity 1, 2); archival data where applicable (activity 3a, 3b, 5, 6, 7a, and 7b); a survey where necessary (activity 4, 8). Activity 9–15 describe transitional steps (with a probability $P = 1$) used to aggregate the outcome into three possible states: having an exploit for a zero-day vulnerability (activity 9, 10), having an exploit for a patchable disclosed vulnerability (activity 11, 12, 13) or having an exploit for an unpatchable disclosed vulnerability (activity 14, 15).

The three main limitations in terms of scope for the model are described in the next three paragraphs.

Some vulnerabilities are more critical than others. This research focuses purely on software vulnerabilities that can enable an attacker with root/administrator privileges of a computer from a remote location. This type of vulnerability was chosen as it arguably is most severe category of software flaw and thus is the most important to manage for practitioners.

The model considers only software programs that need be compiled (e.g., C++ applications). In essence this means that it cannot be used to describe interpreted software vulnerabilities (e.g., PHP applications). This delimitation is taken partly due to that these types of software less frequently compete in the same product segment, and partly due to that vulnerability discovery and exploit development for these two types of software is radically different. In particular, exploitation of compiled software (typically) requires an understanding of low-level functions such as computer memory management (heap, stack and CPU registers) and assembly language. Exploitation of interpreted software, on the other hand, (typically) only requires knowledge of the high-level language(s) that the software was written in (e.g., PHP, Perl, SQL or XPath). Thus, the knowledge required for exploiting interpreted software is in general arguably lower than for exploiting compiled software. It is however possible to extend the model with activities relevant for exploitation of interpreted software, for instance, using the data from [23].

The skill and resources of the attacker can have major impact on the outcome [38]. The model described in this paper is only valid for an attacker profile of a professional penetration tester that is motivated to gain knowledge of vulnerabilities and obtain exploits for some targeted software. In other words, an individual whose job is to devise exploits and sometimes also to discover novel vulnerabilities. This attacker model was chosen based on a set of interviews with cyber security decision makers and professional penetration testers made during a previous study Sommestad et al. [42]. All data in the model has been gathered with this profile in mind. However, with minor modifications, the model could be employed also for other attacker profiles. For instance, an attacker with minor technical experience cannot discover vulnerabilities or devise exploits; however, such an individual can buy or wait for exploits to be disclosed in the public domain.

Table 1
Data part of the model.

Activity	Source	Section
1	[43]	5.1
2	[20,33]	5.2
3a, 3b	NVD, OSVDB	5.3
4, 8	Survey with 58 experts	5.4
5	ExploitHub	5.5
6, 7a, 7b	NVD, OSVDB	5.6

4. Analyzing goodness-of-fit

To fulfill the purpose of this research, there is a need to populate the model described in Fig. 1 with well suited probability density functions (PDFs). An important requirement is thus to choose the distribution best fit for modeling each activity. Consequently, there is a need to have a set of distributions to test and a method for comparing their goodness of fit.

Based on the assumptions, choices and findings of previous studies on the topic (cf. Section 2), the present research analyzes the fit of six distributions – the exponential (EXP) [31,27,21], log-normal (LN) [3,21], normal (N) [21], Weibull (WBL) [21], gamma (GAM) [31,21], and generalized Pareto (PAR) [21]. Marconato et al. [30] tested the Beta distribution (that models outcome in the [0,1] domain) by translating continuous time data to relative frequencies corresponding to a predefined discrete number of bins. The present research delimits from the Beta due to the loss of information involved in such a translation (in the work by Marconato et al. [30], a bin can consist of several hundred days).

There are various metrics that can be applied to measure how well a statistical distribution model observed data. For example, Kolmogorov–Smirnov, Cramér–von Mises, Anderson–Darling, Shapiro–Wilk and Chi Square. This research utilizes the Akaike Information Criterion (AIC) [2], a standard technique for ranking alternative models, to compare the relative goodness of fit for the distributions. The AIC extends the method of maximum likelihood estimation with the situation of multimodel choice [2]. As pointed out by Burnham and Anderson [9], the AIC links Boltzmann's entropy, Kullback–Leibler information and maximum likelihood – thus tying together information theory with statistics. Benefits of employing AIC in front of hypothesis testing techniques are described by [9]. The arguably greatest benefit is its potential regarding model selection, as it is independent of the order that models are computed.

Essentially, the AIC describes the relative fit of a distribution in relation to the others' that are tested. The lower its AIC, the better a

distribution fits the data. To evaluate distributions against each other, the difference $\Delta_i = AIC_i - AIC_{\min}$ is formed, where AIC_i is the value of the model being evaluated and $AIC_{\min}$ is the value of the model with the lowest AIC. By definition $AIC_{\min}$ thus has $\Delta_i = 0$.

If there are too many parameters (K) in relation to the size of the sample (n), AIC may perform poorly. According to Burnham and Anderson [9], the ratio n/K should be above 40 for AIC to perform well. If analyzing a smaller sample size it is recommended to employ AIC_c, a version of AIC with higher-order corrections [9]. For the present study, a K of 2 was employed, as this is the largest number of parameters used in any of the statistical distributions considered – thus giving a sample size threshold of 80. When sample sizes smaller than 80 were available, AIC_c was used for analysis in front of AIC.

5. Data in the studied model

This section describes how the states and activities in the studied model (cf. Fig. 1) are populated with data. Sections 5.1, 5.2, 5.3 concern vulnerability availability estimates (state A—E and activity 1—3); Sections 5.4, 5.5, 5.6 concern exploit availability estimates (state F—M and activity 4—15). All AIC and AIC_c scores mentioned in this chapter are given in Appendix A.

5.1. Activity 1: Discover a zero-day vulnerability

If there is no disclosed vulnerability available, or if the attacker wants to exploit a software vulnerability that is unknown to the public domain (and thus also the developer), then (s)he can attempt to discover a new vulnerability (often called a zero-day) [6].

5.1.1. Data collection methodology

This research found one study related to effort estimates on vulnerability discovery of application binaries: Sommestad et al. [43] studied the effort required by a professional penetration tester to discover a CVSS high severity zero-day with 5%, 50% and 95% likelihood, given the presence or absence of four conditions: (1) if the targeted software has been scrutinized before, (2) whether the attacker has access to the application source code, (3) whether the software is written in a (memory) "safe" language (e.g., C# or Java) or using a (memory) "safe" dialect (e.g., Cyclone), and (4) if the software has been analyzed by static code analyzers and improved based on the result. As all possible combinations between these four measures were tested, a total of 16 scenarios were studied. For each scenario, it is assumed that the attacker is able to study the application binary (e.g., using a debugger).

Sommestad et al. [43] employed the judgment by 17 domain experts to gather estimates on the topic. The judgment of these experts were weighted using Cooke's classical method [12]. Essentially, this involves administrating weights according to the respondents' performance on a knowledge test measuring both their certainty and accuracy. Experience shows that this method outperforms both the best expert and the "equal weight" combination of experts' estimates [13].

5.1.2. Results

The resulting estimates for the 16 studied scenarios can be seen in Table 2. As only three points are known for each PDF, it is not reliable to perform distribution fitting. A better choice is to use linear interpolation (LI) to find the probability P_1 of discovering a zero-day given T_W work days spent on one of the 16 vulnerability discovery projects.

For example, pose the 7th vulnerability discovery project: An attacker does not have the software's source code. The software has been previously scrutinized by others. It has not been developed in languages "safe" to buffer overflows. It has been tested for security issues using static code analysis tools. The probability P_1 corresponding to this scenario can be seen in Eq. (1). If we also make an assumption on the number of work days that the attacker spends probing for a vulnerability, analytically solve the probability corresponding to this project. For instance, given a $T_W = 50, P_1 = 52.2\%$.

$$P_1 = \mathrm{P}(X \leqslant T_W \mid X \in \mathrm{LI}(2, 9, 855) \text{ if Project } 7 \quad (1)$$

Finally, this research assumes that the time required to write the exploit code is included in the estimates provided by Sommestad et al. [43]. This is done as the estimates in Sommestad et al. [43] include the attacker identifying the bug as being a vulnerability of high severity. To accomplish this, there is generally a need for the attacker to understand how to write the exploit code itself. As exploits generally are less than a hundred lines of code, writing it can be expected to require minor effort.

5.2. Activity 2: Purchase a zero-day vulnerability

If an attacker is unable to either find a disclosed vulnerability or discover a new one, then (s)he can, given that a sufficient amount of money is available, purchase a zero-day.

5.2.1. Data collection methodology

This study found two sources for zero-day prices: Greenberg [20]; Miller [33]. These studies denote a total of 29 rough prices (min, max or mean USD) of zero-days. Distribution fitting is conducted for these 29 samples. It should be noted that Radianti et al. [40] found significantly lower prices during their study of vulnerability black markets. The present research did not incorporate their findings due to two primary reasons: (1) their data does not concern final selling prices, and (2) it is unclear if the exploits are for zero-days or disclosed vulnerabilities (and if so, whether a patch exist).

It should also be noted that it can be difficult to find a relevant zero-day for purchase. The time required to find a zero-day to purchase is not included in the model due to the difficulty of providing valid estimates for it. In the model it is simply assumed that money is the only relevant factor for zero-day purchasing.

5.2.2. Results

As described in Table 3, the exponential distribution is best fit for modeling the data presented by Greenberg [20]; Miller [33].

The probability of activity 2 (P_2) in the model (cf. Fig. 1) being successful can be calculated under the assumption that the purchase of a vulnerability also provides its corresponding exploit, that a vulnerability is available for procurement regardless of the targeted software, and that M USD is available for the purchase (cf. Eq. (2)). For instance, the likelihood of a zero-day being available for USD 7000 is 8.8%, and for USD 70,000 it is 60.2%.

$$P_2 = \mathrm{P}(X \leqslant M \mid X \in \mathrm{EXP}(1.32 * 10^{-5})) \quad (2)$$

5.3. Activity 3: find a public vulnerability

Typically, software vulnerabilities are disclosed in the public domain at some point in time after their discovery. A known flaw could render the time-consuming process of discovering a new vulnerability unnecessary for an attacker, especially if no mitigating software patch yet exists. If there is no disclosed vulnerability at the present time, an attacker can simply wait for one to be released (although waiting time might be infinite in some cases).

Table 2
Activity 1 – effort (in work days) required to discover a zero-day with a certain likelihood (reprinted from [43]).

Project	Scrutinized	SourceCode	SafeLanguage	CodeAnalyzers	Low (5%)	Median (50%)	High (95%)
1	Yes	Yes	Yes	Yes	3	13	74
2	Yes	Yes	Yes	No	1	3	26
3	Yes	Yes	No	Yes	0	13	26
4	Yes	Yes	No	No	0	1	7
5	Yes	No	Yes	Yes	1	12	855
6	Yes	No	Yes	No	0	10	27
7	Yes	No	No	Yes	2	9	855
8	Yes	No	No	No	1	4	257
9	No	Yes	Yes	Yes	1	6	27
10	No	Yes	Yes	No	0	4	9
11	No	Yes	No	Yes	0	3	17
12	No	Yes	No	No	1	3	8
13	No	No	Yes	Yes	1	14	344
14	No	No	Yes	No	1	7	27
15	No	No	No	Yes	1	6	18
16	No	No	No	No	0	3	9

Table 3
Cost in USD for purchasing a zero-day vulnerability.

Activity	2
Samples	29
Median	50000
Best fit	EXP (A)
Parameter A	$1.32 * 10^{-5}$

5.3.1. Data collection methodology

While there are various studies on archived vulnerabilities (e.g., [41,30]), this research found no previous study that considers the time between critical vulnerabilities from the perspective of a software, and considers patch and vulnerability disclosure dates in combination. Consequently, data was gathered to fulfill this perspective.

The US National Vulnerability Database (NVD) [35] is the most reknown database for disclosed vulnerabilities. Currently, the NVD contains over 54,000 vulnerabilities, each denoted by its unique Common Vulnerabilities and Exposures identification number (CVE) [34]. Each CVE is associated with a number of attributes, of which two are of especial importance to the present study: The Common Vulnerability Scoring System (CVSS) [32] and the Common Platform Enumeration (CPE) [10]. The CVSS is a collection of metrics used to quantify characteristics of vulnerabilities. For the topic of this study, the CVSS Base Score is of interest. Base Score denotes the severity of a vulnerability on a scale from 0–10, where 0.0–3.9 indicates Low severity, 4.0–6.9 indicates Medium severity, 7.0–8.9 indicates High severity, and 9.0–10.0 indicates Critical severity. All vulnerabilities in the Critical category concerns remote attacks resulting in root/admin privileges. Consequently, this category of vulnerabilities is used in the present study. The CPE is a standard for providing each software product (and version) a unique identification number. An example CPE is "*cpe:/a:adobe:acrobat:9.1*", where "*/a*" denotes that it is an application software. Other possible attributes are "*/o*" (operating system) and "*/h*" (hardware, e.g., a hardware router).

As vulnerabilities listed in the NVD are available for download (stored in XML), all disclosed Critical vulnerabilities could be extracted. However, the NVD unfortunately does not describe the potential patch release date corresponding to each vulnerability. For this purpose, another vulnerability database, the Open Source Vulnerability Database (OSVDB) [37], was consulted. As the OSVDB does not provide its database for download (and local processing), a multi-threaded web crawler[1] was constructed to collect data for the Critical vulnerabilities extracted from NVD. The harvesting occurred during the 12th of December 2012 – any information uploaded post this date is thus not part of analysis.

A total of 3217 Critical vulnerabilities (related to 35,260 CPE's) fulfilled the requirement of having both disclosure date and patch date available. It should be noted that this does not mean that only 3217 critical vulnerabilities are patchable. This is simply the number of flaws that have dates regarding when vendor solutions were issued available in OSVDB. For each CPE, the number of days between vulnerabilities were extracted for two different scenarios – whether or not a patch mitigating the vulnerability was available at the time of its disclosure. Sometimes multiple vulnerabilities are disclosed simultaneously for the same software, yielding zero time differences between their releases. As this would disable distribution fitting of several important probability distributions (e.g., the log-normal and Weibull) all zeroes are replaced with numbers arbitrarily close to zero (10^{-41}). This allows fitting all relevant distributions while having negligible impact on results.

The observations for each CPE are then categorized according to their product types (application, operating system or hardware), yielding six different datasets (two for each product type corresponding to the two studied vulnerability scenarios). More granular classifiers would be beneficial; unfortunately, to the authors' knowledge this is the only valid information that is available for this purpose. Of these six datasets, the two concerning hardware vulnerabilities were excluded due to being too few (49 out of the total 3217 flaws). Consequently, 3168 vulnerabilities (related to 34,159 CPE's) are studied.

5.3.2. Results

There are four different datasets to conduct distribution fitting for. The best suited distributions for these datasets can be seen in Table 4. The sample sizes might seem surprisingly large. This is due to that a CVE typically is associated with multiple CPE's.

Table 4
Characteristics of days between disclosed vulnerabilities.

Software	Application		Operating system	
Patch	Yes	No	Yes	No
Activity	3a	3b	3a	3b
Samples	189016	50285	5022	100
Median	10^{-41}	217	10^{-41}	105
Best fit	PAR (A,B)	GAM (A,B)	GAM (A,B)	LN (A,B)
Parameter A	29.06	0.036	0.014	4.85
Parameter B	$1.38 * 10^{-41}$	7755.7	3630.1	0.98

[1] http://www.ics.kth.se/Scripts/Web_Crawlers.zip.

When waiting for a disclosed patchable vulnerability, there is a likelihood p that it will be released post the patch mitigating the flaw (activity 3a). Similarly, there is a likelihood $1-p$ that it will be released prior to the patch (activity 3b). Four attributes are of relevance for activity 3a and 3b: the number of calendar days since the targeted software (version) was released (T_S), the number of calendar days that the attacker spends waiting (T_W), whether the targeted software is an operating system or application, and if the software is publicly known. The probability corresponding to each of the four activities can be found in Eqs. (3) and (4).

$$P_{3a} = \begin{cases} \frac{189016}{239301} * P(X \leqslant T_S + T_W \mid X \in \text{PAR}(29.06, 1.38 * 10^{-41})) & \text{if Application} \\ \frac{5022}{5122} * P(X \leqslant T_S + T_W \mid X \in \text{GAM}(0.014, 3630.1)) & \text{if OS} \\ 0 & \text{if software is not publicly known} \end{cases} \quad (3)$$

$$P_{3b} = \begin{cases} \frac{50285}{239301} * P(X \leqslant T_S + T_W \mid X \in \text{GAM}(0.036, 7755.7)) & \text{if Application} \\ \frac{100}{5122} * P(X \leqslant T_S + T_W \mid X \in \text{LN}(4.85, 0.98)) & \text{if OS} \\ 0 & \text{if software is not publicly known} \end{cases} \quad (4)$$

A problem with Eqs. (3) and (4) is however that they only consider the possibility of a single vulnerability being disclosed, when in reality, multiple vulnerabilities can be disclosed within the same period of time. For instance, P_{3b} (given an application) max out at $\frac{50285}{239301} = 0.21$, as this expression is a constant. However, if two vulnerabilities are disclosed, then the likelihood that at least one them is unpatchable is $0.21 \cup 0.21 = 0.375$. The likelihood of two disclosed vulnerabilities is although lower than one disclosed vulnerability (P^2 compared to P^1). Eq. (5) shows this max solving problem,[2] with the constraint of n disclosed vulnerabilities. In Eq. (5), P_y refers to the constant expressions in Eqs. (3) and (4); P_z refers to the probabilities gained from the PDFs in these equations. P_y increase and P_z decrease with a larger n; when T_W or T_S goes towards infinity, P_3 goes towards 1.

$$\arg\max_n f(n) := \left\{ (P_{y1} \cup P_{y2} \cup \ldots \cup P_{yn}) * (P_z^n) \right\} \quad (5)$$

For instance, P_{3a} (given an application) is solved by Eq. (6). The results for P_{3a}, given an example scenario with an application, $T_S = 0$, and $T_W = 100$, are described in Fig. 2. As can be seen, a max of $P_{3a} = 92\%$ is reached at $n = 3$.

$$\arg\max_n f(n) := \left\{ (0.79_1 \cup \ldots \cup 0.79_n) * (\text{P}(X \leqslant 0 + 100 \mid X \in \text{PAR}(29.06, 1.38 * 10^{-41}))^n) \right\} \quad (6)$$

5.4. Activity 4,8: develop exploit for disclosed vulnerability

If a vulnerability is disclosed, but no exploit is made available, then a proficient attacker can utilize the information made public about the vulnerability and devise an exploit for it himself/herself.

Unfortunately, there are no data on the effort required to devise an exploit for a disclosed vulnerability. One could, as McQueen et al. [31], argue that it should be similar to the time from vulnerability disclosure to exploit release. However, the error with this assumption can be illustrated with an example: Pose that N individuals are attempting to conceive an exploit for a vulnerability that was disclosed at time T_V. At a time T_E one of these individuals manages to successfully devise an exploit. To be credited as the original inventor of the exploit, that individual publishes this exploit online. T_E will consecutively be detailed on OSVDB and any archival data study will assume that the effort to conceive an exploit for this vulnerability is $(T_E - T_V)$. There are two major problems with trusting this estimate: (1) By doing so, one disregard the effort spent by the $(N-1)$ individuals who were not first at devising the exploit (or chose not to publish it). (2) It is unknown when the individual(s) who devised the exploit started to work on it, how devoted (s)he was to this work and whether or not it was released immediately upon creation.

Consequently, there is a need to utilize a data source other than archival data for accurate estimates. In this study we employ expert judgment in a similar fashion to that previously explained in Section 5.1.

5.4.1. Studied variables

The main objective for the survey was to gain effort estimates (in work days) regarding development of exploits for disclosed CVSS critical software vulnerabilities. As not all respondents might be familiar with the CVSS, the vulnerability was specified as "a publicly disclosed software vulnerability that can allow arbitrary code execution with admin/root privileges".

The effect of two variables were studied: (1) the availability of a patch mitigating the vulnerability, and (2) whether the application's or the patch's source code are available to the attacker. Availability of a patch relates to two possible outcomes in the process model: whether the targeted vulnerability is patchable (activity 11) or not (activity 15). Presence of a patch can help an attacker to further identify the location of a vulnerability. Availability of source code

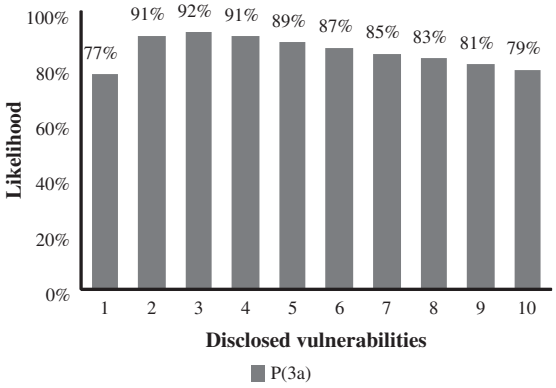

Fig. 2. P_{3a} as a function of disclosed vulnerabilities.

[2] Solving the largest possible value from an equation that has multiple solutions.

can aid an attacker at discovering "deep" vulnerabilities that might be difficult to find using a debugger; it can however also aid defenders at quickly mitigating flaws [14]. All combinations between these variables were studied, providing four different scenarios for analysis.

To delimit the uncertainty of responses, while at the same time enabling results valid to the scope of this research, a set of variables were also fixed. For all questions, it was assumed that: (1) there was no exploit code (or parts of it) available for the targeted vulnerability, (2) the vulnerable software was not written in an interpreted language, and (3) there was as much information available about the vulnerability as is typical for the four studied scenarios.

5.4.2. Invited experts

Sometimes CVEs detailed in OSVDB have information about the individuals or organizations that are credited for discovering the vulnerability or devising its exploit. We collected all email addresses of those credited for severe vulnerabilities as these arguably are the optimal respondent for this type of question.

A total of 1368 email addresses were discovered based on this method. This set was then manually inspected for non-relevant addresses, for instance, email addresses related to entire organizations or email lists. This inspection reduced the set to 1323 email addresses.

All 1323 respondents were sent personalized emails (by relating their addresses to the CVEs that they had discovered) inviting them to a web survey. As recommended by Elvey-Kirk [11], motivators were presented to the survey participants: (1) helping the community as whole and (2) being able to compare their answers to other experts' answers after the survey was completed. Out of the 1323 emails sent, 576 bounced. Several respondents also denoted that they had received multiple invites as they had several email addresses logged in the OSVDB. Consequently, the unique number of respondents who received the survey is likely substantially lower than 747. The survey was online from the 21st of January to the 7th of February 2013 (with a reminder sent during the 4th of February 2013). A total of 58 respondents fully answered the survey during this time.

5.4.3. Elicitation instrument

The survey consisted of two parts (given on a single page). First, the respondents were given an introduction to the survey that explained its purpose and its outline. In this introduction they also provided information about themselves, e.g., years of experience related to software vulnerability discovery and exploitation.

The second part concerned the effort estimates. The beginning of this section trained the respondents regarding the answering format used in the survey; the remainder of the section concerned the questions of interest to the study. For these, the respondents were asked to specify the number of work days required to develop an exploit for a disclosed vulnerability given the overall assumptions and the question-specific conditions. Each of the four survey questions asked the respondents to provide probability distributions that expressed their beliefs (by setting the 5th percentile, the 50th percentile (the median), and the 95th percentile). This is a common format to use for effort estimates [28] and in prediction in general [5]. In the survey the respondents specified their distribution by clicking buttons or entering values to draw a dynamically updated graph over their probability distribution. In Fig. 3 the format presented to respondents is exemplified.

The reliability of the survey tool was tested using a number of methods:

1. The question format was tested using Cronbach's alpha [15] in a previous pilot study with 34 respondents (by phrasing the same question in four different ways). Results from this test showed $\alpha = 0.817$, which indicates good internal consistency of the instrument.
2. A telephone interview was conducted with a respondent that has been credited for the discovery of various critical software vulnerabilities. During this interview the respondent was given the task to fill in the survey without any help from the researchers (followed up by a dialogue). Several improvements were made to the tool after this session, the most significant being that the response variable was changed from hours to work days. Furthermore, after filling the survey in, the respondent was provided with two alternative answering formats: (1) by only estimating means or (2) by estimating both means and variances. The respondent opted for the three-point estimate due to the great uncertainty involved in the studied scenarios.
3. The last question of the survey asked the respondents to detail any issues with the elicitation tool itself. No respondent indicated any issues with the tool as such; however, multiple respondents suggested replacing the variable of source code with something else as they perceived access to source as irrelevant. For instance, one respondent stated "*The difficulty of exploit development is rarely connected to the availability of source code. Today, it is mostly about outside conditions like operating system protections*". However, the results indicate that the availability of source code has significant impact (see the next section).

5.4.4. Results

The 58 participants were associated (based on their connecting IPs) with a total of 25 countries and one TOR[3] exit node. A majority (16) of the respondents were participating from the United States (from 10 different states), but a number of other countries were also observed [e.g., Germany (6 respondents), United Kingdom (4), Finland (3), Algeria (2), and Russia (1)]. The mean experience related to vulnerability discovery and exploit development was 9.8 years and the mean perceived competence was 51.6%. Perceived competence has a scale from 1% to 100%, where 1% meant that the respondent perceived itself to be more knowledgeable on the topic than 99% of the community. The respondents generally positioned themselves towards practice rather than research (a mean of 44.2 on a scale from 1: only work with industry/practice to 100: only work with academia/research).

To achieve reliable estimates on the number of work days required to develop an exploit for a disclosed vulnerability, it is important to account for both the variability within each expert's estimates and the variability between estimates by different experts. To accomplish this, distribution fitting is first done for each three-point estimate (5/50/95); this enables accounting for the uncertainty between experts. Then, the PDF corresponding to each of the four scenarios is constructed through linear interpolation (LI).

Table 5 shows the distributions best fit for modeling each 3-point estimate for the four studied scenarios. For example, given a scenario where a patch mitigating the vulnerability is available, but the source code for the application or patch is unavailable, the variability between experts regarding the 5% likelihood is best modeled by a gamma distribution with a shape parameter $k = 0.041$ and scale parameter $\theta = 85.5$ (with a median of 2 work days).

Eqs. (7) and (8) show the probability P_4 of devising an exploit for a disclosed vulnerability given that a patch is available, and

[3] A proxy service for anonymity, https://www.torproject.org/.

P_8 given that a patch is unavailable, when the attacker is able to spend T_W work days on the activity.

$$P_4 = \begin{cases} P(X \leqslant T_W \mid X \in LI(GAM(0.025, 45.5), GAM(0.25, 12.5), LN(0.075, 1.63))) & \text{if Source} = \text{True} \\ P(X \leqslant T_W \mid X \in LI(GAM(0.041, 85.5), GAM(0.13, 56.8), GAM(0.12, 221.3))) & \text{if Source} = \text{False} \end{cases} \quad (7)$$

$$P_8 = \begin{cases} P(X \leqslant T_W \mid X \in LI(GAM(0.028, 82.4), LN(0.20, 1.25), LN(0.061, 1.94))) & \text{if Source} = \text{True} \\ P(X \leqslant T_W \mid X \in LI(GAM(0.046, 111.3), GAM(0.22, 59.4), GAM(0.19, 222.5))) & \text{if Source} = \text{False} \end{cases} \quad (8)$$

5.5. Activity 5: Purchase exploit for disclosed vulnerability

If a disclosed vulnerability is available for the software of interest, then an applicable exploit sometimes is available for purchase.

5.5.1. Data collection

This research found one public source where exploits of disclosed vulnerabilities can be purchased: the ExploitHub [16], a venue where exploit authors sell exploits for publicly disclosed vulnerabilities. In practice an attacker might have other means of purchasing exploits, e.g., from friends or underground markets. Such activities are not part of this process model.

A web crawler[4] was constructed to harvest all exploits available on ExploitHub. Data was collected during the 21st of December 2012. A total of 86 CVSS critical application exploits and 12 CVSS critical operating system exploits, all corresponding to patchable vulnerabilities, were found.

5.5.2. Results

An overview of results (costs in USD) from the distribution testing can be seen in Table 6. The log-normal distribution is best fit for modeling cost of exploits for both applications and operating systems.

Furthermore, having enough money to buy an exploit from ExploitHub does not necessarily mean that the sought exploit can be acquired as it might be unavailable. To obtain availability estimates on exploits in ExportHub, this research compares the number of CVSS critical vulnerabilities in ExploitHub to the total number of disclosed CVSS critical vulnerabilities that have patches, but no public exploits (the studied scenario). The prior are the sample sizes in Table 6; the latter can be found by querying the OSVDB. The data gathered from OSVDB show that there are a total of 2706 application vulnerabilities and 241 operating system vulnerabilities that fulfill this scenario. Using this information, availability estimates can be made. Eq. (9) shows the likelihood P_5 of purchasing an exploit for a disclosed vulnerability, given that M USD is available.

$$P_5 = \begin{cases} \frac{86}{2706} * P(X \leqslant M \mid X \in LN(4.68, 0.78)) & \text{if Application} \\ \frac{12}{241} * P(X \leqslant M \mid X \in LN(4.76, 1.07)) & \text{if OS} \end{cases} \quad (9)$$

5.6. Activity 6,7: Find a public exploit

If a publicly disclosed vulnerability exist then there might also be a public ready-to-use exploit available, or one made available if enough time has passed. Furthermore, in the same fashion as vulnerabilities (cf. Section 5.3), an exploit can be published either before or after that a patch mitigating the vulnerability is released.

5.6.1. Data collection methodology

While there are various studies on archived vulnerabilities and exploits (e.g., [41,30]), this research found no previous study that considers whether or not a vulnerability is patchable upon the release of its corresponding exploit. New archival data was thus collected to enable studying this property.

A total of 3168 critical vulnerabilities in operating systems and applications were used to find the number of calendar days between vulnerabilities that did or did not have a patch mitigating the vulnerability available at the time of disclosure. The analysis of calendar days between vulnerability disclosure and exploit release is based on the same set of vulnerabilities. However, not all vulnerabilities have exploit release dates available. Out of the 3168 vulnerabilities in the pool, 220 detailed exploit release dates. It should be noted that this, as for vendor solution date, does not mean that only 220 critical vulnerabilities have exploits released. It is simply the number of vulnerabilities that have exploit release dates available in OSVDB.

Sixteen of these vulnerabilities had an exploit released before the vulnerability was disclosed. As this type of activity is not part of the process model, these vulnerabilities are excluded from the analysis. Consequently, 204 vulnerabilities constitute the dataset for analysis of time between vulnerability disclosure and exploit release.

Sometimes an exploit is released simultaneously to the disclosure of its corresponding vulnerability. As this would disable distribution fitting of several important probability distributions (e.g., the log-normal and Weibull) all zeroes are replaced with numbers arbitrarily close to zero (10^{-41}), allowing fitting all relevant distributions while having negligible impact on results.

5.6.2. Results

There are three activities that need be considered: (1) If one has a patchable vulnerability, then the only possible next activity is an exploit for a patchable vulnerability (activity 6). (2) If one has a disclosed vulnerability that not yet is patchable, then there is a likelihood p that its exploit will be released post the patch mitigating the flaw (activity 7a). (3) Similarly, there is a likelihood $1 - p$ that the exploit will be released prior to the patch (activity 7b). These three possible activities can be extracted for both operating systems and applications (designated by CPE), giving six datasets for distribution fitting.

The best suited distributions for these datasets can be seen in Table 7. The data for activity 7a (given an operating system) only had a single sample available. For this dataset, the exponential distribution was assumed due to being best fit for modeling 7a for applications. In Table 7, *PatchVulnerability* refers to whether the vulnerability was patchable before it was disclosed; *PatchExploit*

[4] http://www.ics.kth.se/Scripts/Web_Crawlers.zip.

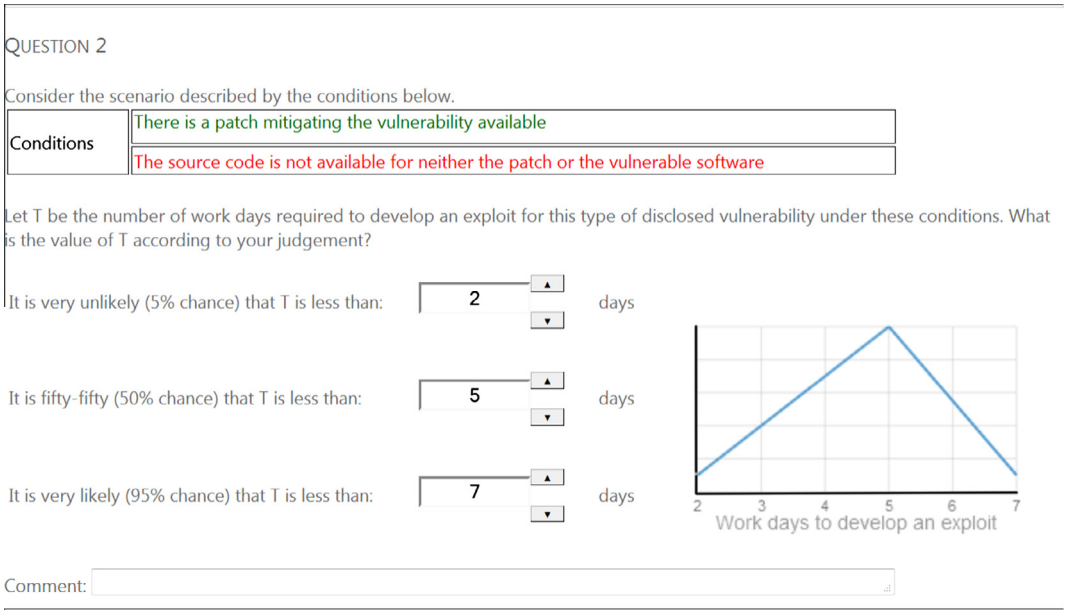

Fig. 3. Question and answering format used in the survey.

Table 5
Effort (in work days) required to develop an exploit for a disclosed vulnerability with a certain likelihood. 58 samples are used for all tests.

			Median effort, best fit distribution		
Activity	Patch	Source	5%	50%	95%
4	Yes	Yes	1, GAM (0.025,45.5)	2, GAM (0.25, 12.5)	5, LN (0.075,1.63)
4	Yes	No	2, GAM (0.041,85.5)	5, GAM (0.13, 56.8)	8, GAM (0.12, 221.3)
8	No	Yes	1, GAM (0.028,82.4)	3, LN (0.20,1.25)	6, LN (0.061,1.94)
8	No	No	4, GAM (0.046,111.3)	7, GAM (0.22,59.4)	11, GAM (0.19,222.5)

Table 6
Cost (in USD) of exploits for disclosed vulnerabilities.

Software	Application	Operating system
Activity	5	5
Samples	86	12
Median	100	100
Best fit	LN (A,B)	LN (A,B)
Parameter A	4.68	4.76
Parameter B	0.78	1.07

refers to whether the vulnerability was patchable before its exploit was released.

The calendar days since the disclosure of the vulnerability (T_D) and the number of days spent waiting (T_W) are of importance for all activities. T_D is however only possible to estimate given that vulnerability characteristics are available, e.g., when a previous security audit has been used to identify vulnerabilities (given this scenario, P_{3a} or $P_{3b} = 1$). T_D is set to 0 in a situation when vulnerability information is unavailable (i.e., it is assumed that no disclosed vulnerability was available from the start). It should however be noted that it is possible to create more detailed rules for this variable, e.g., as the difference between the time since release of the software and the expected time between vulnerabilities of that type (cf. Section 5.3.2). Eqs. (10)–(12) shows the probabilities related to activity 6 (P_6), 7a (P_{7a}) and 7b (P_{7b}), respectively.

$$P_6 = \begin{cases} P(X \leqslant T_D + T_W \mid X \in \text{GAM}(0.033, 1313.37)) & \text{if Application} \\ P(X \leqslant T_D + T_W \mid X \in \text{GAM}(0.039, 567.91)) & \text{if OS} \end{cases} \quad (10)$$

$$P_{7a} = \begin{cases} \frac{8}{55} * P(X \leqslant T_D + T_W \mid X \in \text{EXP}(0.073)) & \text{if Application} \\ \frac{1}{11} * P(X \leqslant T_D + T_W \mid X \in \text{EXP}(0.0085)) & \text{if OS} \end{cases} \quad (11)$$

Table 7
Characteristics of days from vulnerability disclosure to exploit release.

Software	Application			Operating system		
PatchVulnerability	Yes	No	No	Yes	No	No
PatchExploit	Yes	Yes	No	Yes	Yes	No
Activity	6	7a	7b	6	7a	7b
Samples	118	8	47	20	1	10
Median	3	8	10^{-41}	8	117	10^{-41}
Best fit	GAM (A,B)	EXP (A)	PAR (A,B)	GAM (A,B)	EXP (A)*	PAR (A,B)
Parameter A	0.033	0.073	11.83	0.039	0.0085	15.33
Parameter B	1313.37	–	$1.19 * 10^{-41}$	567.91	–	$1.27 * 10^{-41}$

* Assumed based on result for application.

$$P_{7b} = \begin{cases} \frac{47}{55} * P(X \leqslant T_V + T_W \mid X \in \text{PAR}(11.83, 1.19 * 10^{-41})) & \text{if Application} \\ \frac{10}{11} * P(X \leqslant T_V + T_W \mid X \in \text{PAR}(15.33, 1.27 * 10^{-41})) & \text{if OS} \end{cases} \quad (12)$$

In the same fashion as for disclosed vulnerabilities (cf. Section 5.3.2), a problem with Eqs. (11), (12) is that they only consider the possibility of a single exploit being released, when in reality, multiple vulnerabilities means that multiple exploits can be released. Thus, we again have a max solving problem (cf. Eq. (5)), this time with the constraint of m exploits and a second constraint of $m \leqslant n$, where n is the number of vulnerabilities in activity 3b. I.e., the number of exploits cannot exceed the available number of vulnerabilities.

For instance, P_{7a} (given an application) is solved by Eq. (13). Given an example scenario with an application, $T_D = 0, T_W = 100$, and $n = 4, P_{7a}$ reaches a maximum of 79% at $m = 3$.

$$\arg\max_m f(m) := \left\{(0.15_1 \cup \ldots \cup 0.15_m) * (\text{P}(X \leqslant T_D + T_W \mid X \in \text{EXP}(0.073))^m)\right\}, \quad \text{where } m \leqslant n \text{ of activity 3b.} \quad (13)$$

6. Model implementation

6.1. Calculation framework

The entire model contains a total of 13 states and 17 activities, connected as a Bayesian network [17]. The probability of each activity being successful (i.e., true) is described by a Conditional Probability Table (CPT). The likelihood value corresponding to each CPT cell is extracted based on the cumulative distribution function (CDF) related to the cell and the time parameter designated by the user of the model. For example, activity 1 (zero-day discovery, cf. Section 5.1) consists of a CPT with 16 different cells, one for each project type/configuration. If the user has specified the configuration of project type 1, then the probability corresponding to it is solved based on the number of work days available for the vulnerability discovery project (defined by the user) and the CDF corresponding to linear interpolation of the three values [2,9]. For instance, if it concerns project type 1 and the attacker has 50 work days available, this cell of the CPT has the probability 52.2%.

During the beginning of the calculation process all probabilities related to CPT cells in the Bayesian network are resolved as presented in the previous paragraph. As extracting values from CDFs (given the same input) is a deterministic process, there is no need for sampling during this step. When all CPTs have been populated with probabilities, the Bayesian network is solved through Monte Carlo sampling. If a state can be achieved by multiple activities (states H, K, L, and M) the union probability related to these activities is employed. The reader is referred to [17] for more detailed information on solving Bayesian networks.

6.2. Usage of the model

The model described in this paper has been implemented in Matlab to allow simulating the outcome of the different activities in the model. From a user perspective, there are two activities involved: (1) defining the characteristics of the studied scenario (Table 8 summarizes the required information), and (2) calculating likelihood estimates for this scenario.

An example application of the model for the web browser *Internet Explorer 10* can be seen in Fig. 4; the properties of this software are presented in Table 8. Internet Explorer 10 is a closed-source application software that, for the example, was released 5 days prior to the employment of the model. This software has previously been scrutinized and was developed in a memory unsafe language (C++) with the aid of code analyzers. The attacker targeting it has 1000 USD and 7 days available for each activity.

If the user wishes to simulate the probability of exploits and vulnerabilities within a given duration, rather than from the release date of the software (as for this example), there is simply a need to set the time since release of the software (variable nr. 2) to zero.

The results of these calculations for the example described in this section are detailed in Fig. 4. For instance, the likelihood of state H being true is

$$P_H = P_D P_6 \cup P_E P_{7a}, \quad \text{where} \quad (14)$$

$$\begin{aligned} &P_D = 89.8\%, \\ &P_6 = \text{P}(X \leqslant 7 + 0 \mid X \in \text{GAM}(0.033, 1313.37)) = 85.2\%, \\ &P_E = 25.8\%, \\ &P_{7a} = \arg\max_m f(m) := \{(0.15_1 \cup \ldots \cup 0.15_m) \\ &\qquad * (\text{P}(X \leqslant 7 + 0 \mid X \in \text{EXP}(0.073))^m)\} = 6.1\% \end{aligned} \quad (15)$$

Thus, $P_H = 0.898 * 0.852 + 0.278 * 0.061 - 0.898 * 0.852 * 0.278 * 0.061 = 76.9\%$.

Given the properties of the example, it is 38.3% likely that this attacker is able to obtain an exploit for an unpatchable vulnerability (P_M), 95.7% likely that an exploit for a patchable vulnerability is obtainable (P_L), and 38.0% likely that a zero-day is obtainable (P_K).

The resources of the attacker are difficult to know for a decision maker in the industry. Consequently, it could be useful to vary these for a simulated scenario. Fig. 5 illustrates how the probability of state K, L, and M vary for the example scenario given an attacker who spends between 1 and 100 days ($1 \leqslant T_W \leqslant 100$) on each activity. As can be seen, this software is rather likely to have a public exploit for a patchable vulnerability at the moment the attacker decides to target it ($P_L \approx 90\%$ at $T_W = 1$). P_L then increases to approximately 98% at $T_W = 21$, where it remains rather stationary. The probability of a public exploit for an unpatchable vulnerability begins at $P_M \approx 35\%$, and gradually increases until $T_W = 100$, where

Table 8
Information required to produce likelihood estimates.

Nr.	Variable	States	Activity	Internet explorer 10
1	Time per activity	Days	All	7
2	Time since release of software*	Days	3a, 3b	5
3	Time since disclosure of vulnerability*	Days	6, 7a, 7b	0
4	Software type	OS/Application	3a, 3b, 6, 7a, 7b	Application
5	Software public	Yes/No	3a, 3b	Yes
6	Source available	Yes/No	1, 4, 8	No
7	Software scrutinized by others	Yes/No	1	Yes
8	Code analyzers used	Yes/No	1	Yes
9	Safe language used	Yes/No	1	No
10	Available funds	USD	2, 5	1000

* Not necessary to detail.

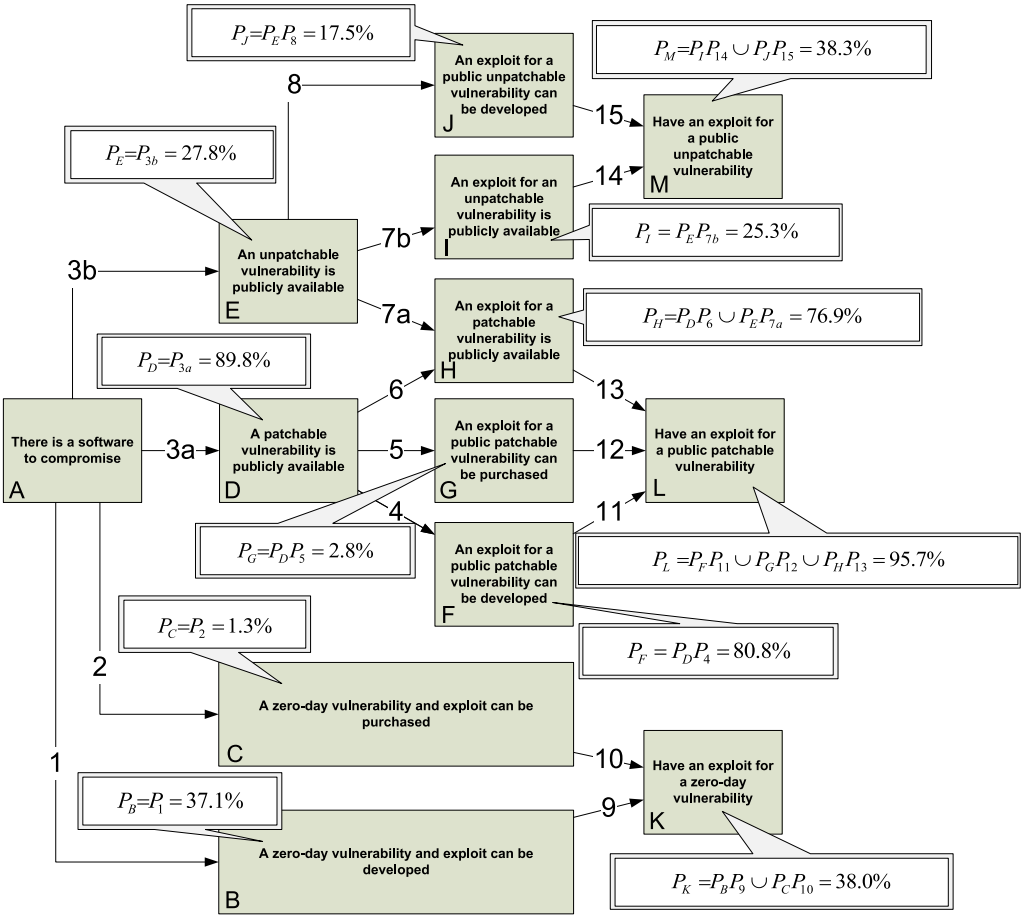

Fig. 4. The model applied for Internet Explorer 10.

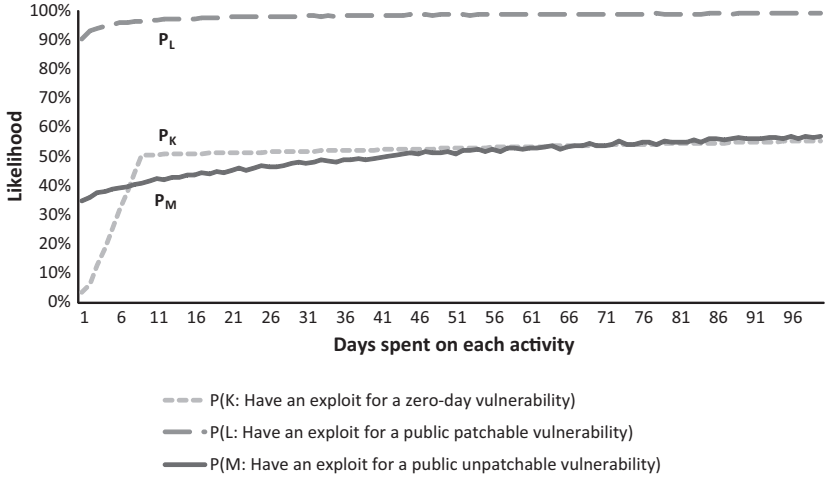

Fig. 5. Outcome given different amounts of time spent on each activity.

$P_M \approx 57\%$. The probability of an exploit for a zero-day vulnerability starts at $P_K \approx 2\%$, and then rapidly increases to approximately 50% at $T_W = 9$ (the median). After this P_K increases slower and rather linearly along T_W, ending with $P_K \approx 55\%$ at $T_W = 100$.

A simulation such as this illustrates interesting characteristics of exploit acquirement. For instance, pose that an attacker seeks an exploit for an unpatchable, but not necessarily undetectable vulnerability. If this attacker has a work week or less to spend, a zero-day is less likely to be obtained than an exploit for an unpatchable disclosed vulnerability; however, if more time is available zero-day discovery might actually be rewarding.

7. Discussion

There are various aspects of the model described in this paper that could be discussed – this section covers a selection of such key topics.

A perfect statistical model provides estimates that always equals the actual outcome. For cyber security estimation models, prediction quality is an extremely difficult task to evaluate as data are scarce - what relevant data that is available is typically used to populate the model. Some aspects can however be critically discussed. In particular, whether the model covers the activities and states of interest and whether the data used to populate the model are reliable.

Does the model include the activities and states of interest? While we believe that most significant activities and states are covered, it is clear that the proposed model is not completely exhaustive. In particular, a typical professional penetration tester has both zero-days and non-public exploits for known vulnerabilities [31]; or friends who are willing to share flaws. One could also in theory purchase an exploit for an unpatchable vulnerability. However, this research found no data on this topic and thus only covers costs for exploits that correspond to patchable vulnerabilities (activity 5). Activities such as these are not part of the model due to being difficult to empirically estimate.

On the validity of the surveys. Activities 1, 2, 4 and 8 in the model concern survey based data extracted from security professionals (Greenberg [20], Miller [33], Sommestad et al. [43] and the present study). The data provided by Greenberg [20] are gained from interviews with unknown experts conducted under unclear circumstances and the data by Miller [33] are the author's personal experiences. The validity of the datums provided by these studies can thus be questioned. That said, the fact that a recent Rand report on this topic [1] only mention these two sources suggests that there indeed are no better alternatives available. The surveys by Sommestad et al. [43] and the present study are based on the guidelines in Cooke's classical method [12], a best-practice expert elicitation methodology. The internal consistency of these surveys was ensured by employment of Cronbach's alpha [15].

On the validity of the archival data. Activities 3a, 3b, 5, 6, 7a and 7b in the model concern archival data extracted from the NVD, the OSVDB and the ExploitHub. There are various other databases that can be employed to study vulnerabilities than the NVD. For instance, Qualys,[5] Secunia[6] and Symantec[7] all provide databases that present analyzed security vulnerabilities and exploits. These databases however rarely contain information about novel vulnerabilities not available in NVD, and if they do, this information is bound to be quickly added to the NVD. The purpose of these databases is rather to provide fine-grained analysis of existing threats and mitigation options (something that is out of scope of the present study). Thus, the NVD is arguably the most valid database for this research. Regarding the OSVDB and the ExploitHub: There are many vulnerabilities that the OSVDB do not contain patch and exploit release dates for, and the ExploitHub contains relatively few exploits for sale. That said, the employed sources of data are arguably the optimal given the scope of the model as no reasonable alternatives are available as of yet. The abstract nature of the model also means that its estimates should not be viewed as the absolute truth, but rather for ranking different alternatives. For this purpose, the additional uncertainty prescribed by small sample sizes might be reasonable given the circumstances. Most aspects of the described model are also possible to gather automatically, utilizing the same scripts as was used during the present study. Future studies could thus extend the employed datasets with little effort.

What about estimates on the likelihood of successful exploitation? Availability of an exploit does not ensure successful attack. For instance, there could be an anti-malware or deep-packet inspection firewall present to prevent the attack. Consequently, the likelihood estimates produced by the described model should not be seen as likelihood estimates for successful attacks. That said, future studies could extend it with estimates also on this topic.

How much effort does usage of the model require? An important requirement for the model is that it should be easy to apply in practice. While this paper has not included any empirical evidence to ascertain this, the input data required for likelihood estimations (cf. Table 8) should need minor effort to gather. The arguably only difficult variables to describe are the resources of the attacker (time and money). This topic is discussed next.

What resources are reasonable for an attacker? It is naturally very difficult for any industry decision maker to know how many individuals who are targeting a specific software, how much effort they spend, and how much funding they have. As the prescribed model requires such details to be specified, it could be problematic for its practical usefulness. The solution to this problem is two-folds: (1) there is a need to study how the likelihood estimates produced by the model vary given different attacker resources, and (2) there is a need to choose an appropriate range for such simulations. While the prior is managed by the model (cf. Fig. 5), the latter might be trickier to define. It is naturally dependent on both the attractiveness of the software and the vulnerability. A comment by one of the participants of the survey (cf. Section 5.4) however give some insight into this property: "*I can tell you most engagements have a time limit for poc* [proof of concept exploit] *construction, sometimes this is a day sometimes this is a few weeks*". Relating back to the probabilities described in Fig. 5: those corresponding to a time spent between 1 and ≈20 days in might thus be more realistic than the rest.

What about other attacker profiles such as script kiddies? The described model is only valid for an attacker profile of a professional penetration tester. However, it could be easily altered to allow simulations of less experienced attackers: A common threat for enterprises is inexperienced attackers who exploit things for the fun of it, not to gain a monetary profit. These attackers are typically not able to discover vulnerabilities or devise exploits. Furthermore, they generally do not have the monetary resources to purchase vulnerabilities or exploits. To account for this type of attacker, there is thus simply a need to define $P_1 = P_2 = P_4 = P_5 = P_8 = 0$. Other types of attackers might be trickier to model.

8. Conclusions

This paper presents a Bayesian network based model that can be used to estimate the likelihood that a professional penetration tester is able to obtain information about critical vulnerabilities and exploits for these vulnerabilities corresponding to a desired software under different circumstances. This model can support enterprise decision making in terms of what software to acquire, and software engineering projects in regard to what practices and measures to invest in.

Data of the activities in the model are gathered from previous empirical studies, online databases and a survey with 58 individuals who all have been credited for the discovery of critical software vulnerabilities. While the model currently only is valid for an attacker profile of a professional penetration tester, it can be easily modified to also account for script kiddies. The model requires little effort to use.

The largest concern with the model lies with a subset of the data used to populate it; however, as the OSVDB is extended with new vulnerability information every day (and new databases are initiated), future studies utilizing the same methodology as is described in this paper should serve to reinforce the reliability of estimates. Given more samples, it could also become possible to make the model more fine-grained. The most extreme possibility

[5] http://www.qualys.com/research/knowledge/.
[6] http://secunia.com/community/advisories/search/.
[7] http://www.symantec.com/deepsight-products.

Table A.9
AIC (given 80 or more samples) and AIC_c (given less than 80 samples) scores for the different tested datasets and distributions.

Activity	Samples	EXP	GAM	LN	N	PAR	WBL
2	29	711	713	721	735	713	713
3a:1	189016	1455953	−23850903	−23906565	2153066	−24201573	−23844469
3a:2	5022	49918	−558859	−558656	64733	−564998	−557586
3b:1	50285	33713	−74394	−70208	36137	−69188	−71523
3b:2	100	1267	1267	1254	1356	1269	1268
4:1:5%	58	131	−3631	−3531	212	131	−3561
4:1:50%	58	246	105	311	267	245	149
4:1:95%	58	411	404	351	606	364	389
4:2:5%	58	260	−1816	−1685	354	256	−1739
4:2:50%	58	350	−57	129	423	350	4
4:2:95%	58	496	34	213	646	460	89
8:1:5%	58	214	−3002	−2893	315	208	−2929
8:1:50%	58	297	292	282	345	299	296
8:1:95%	58	434	431	388	607	401	420
8:2:5%	58	304	−1492	−1353	386	302	−1414
8:2:50%	58	411	236	433	504	406	276
8:2:95%	58	546	323	508	680	506	355
5:1	86	1039	1031	1011	1128	1041	1038
5:2	12	155	158	155	174	157	158
6:1	118	1133	−4373	−4265	1454	−4293	−4285
6:2	20	166	−617	−594	200	−595	−599
7a:1	8	60	63	62	69	64	64
7a:2	1	–	–	–	–	–	–
7b:1	47	107	−4476	−4524	267	−4617	−4498
7b:2	10	33	−882	−889	64	−906	−885

of this would be to provide likelihood estimates for each CPE. In other words, the end-user denotes what software and version that is in question, and the model produces estimates unique for this CPE.

Finally, it would be valuable to survey how many non-public critical exploits that different types of attackers typically knows, and what types of applications these generally concern. This is arguably the most significant activity that is missing in the described model.

Appendix A. Goodness-of-fit results

Table A.9.

References

[1] L. Ablon, M.C. Libicki, A.A. Golay, Markets for Cybercrime Tools and Stolen Data: Hackers' Bazaar, Rand Corporation, 2014.
[2] H. Akaike, Factor analysis and AIC, Psychometrika 52 (1987).
[3] O. Alhazmi, Y. Malaiya, Application of vulnerability discovery models to major operating systems, IEEE Trans. Reliab. 57 (1) (2008) 14–22.
[4] O.H. Alhazmi, Y.K. Malaiya, Modeling the vulnerability discovery process, in: Software Reliability Engineering, 2005. 16th IEEE International Symposium on ISSRE 2005, IEEE, 2005, p. 10.
[5] J. Armstrong, Principles of Forecasting: A Handbook for Researchers and Practitioners, vol. 30, Springer, 2001.
[6] L. Bilge, T. Dumitras, Before we knew it: an empirical study of zero-day attacks in the real world, in: Proceedings of the 2012 ACM Conference on Computer and Communications Security, ACM, 2012, pp. 833–844.
[7] M. Bishop, Introduction to Computer Security, Addison-Wesley Professional, 2004.
[8] B.W. Boehm, Software engineering economics, IEEE Transactions on Software Engineering (1) (1984) 4–21.
[9] K. Burnham, D. Anderson, Model Selection and Multimodel Inference: A Practical Information-theoretic Approach, Springer-Verlag, 2002.
[10] A. Buttner, N. Ziring, Common platform enumeration (CPE) specification, 2009. <http://cpe.mitre.org>.
[11] S. Cavusgil, L. Elvey-Kirk, Mail survey response behavior: a conceptualization of motivating factors and an empirical study, Eur. J. Market. 32 (11/12) (1998) 1165–1192.
[12] R. Cooke, Experts in Uncertainty: Opinion and Subjective Probability in Science, Oxford University Press, USA, 1991.
[13] R.M. Cooke, L.L. Goossens, Tu delft expert judgment data base, Reliab. Eng. Syst. Safety 93 (5) (2008) 657–674.
[14] C. Cowan, Software security for open-source systems, Security Privacy, IEEE 1 (1) (2003) 38–45.
[15] L. Cronbach, Coefficient alpha and the internal structure of tests, Psychometrika 16 (3) (1951) 297–334.
[16] ExploitHub, ExploitHub – a marketplace for vulnerability testing, 2013. <https://www.exploithub.com/> (accessed 8.2.13).
[17] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Mach. Learn. 29 (2–3) (1997) 131–163.
[18] A. Fuggetta, Software process: a roadmap, in: Proceedings of the Conference on the Future of Software Engineering, ACM, 2000, pp. 25–34.
[19] M. Goodyear, H.T. Goerdel, S. Portillo, L. Williams, Cybersecurity management in the states: the emerging role of chief information security officers, IBM Center for the Business of Government, 2010.
[20] A. Greenberg, Shopping for Zero-Days: A Price List for Hackers' Secret Software Exploits, 2012. <http://www.forbes.com/sites/andygreenberg/2012/03/23/shopping-for-zero-days-an-price-list-for-hackers-secret-software-exploits/>, (accessed 7.2.13).
[21] H. Holm, A large-scale study of the time required to compromise a computer system, IEEE Trans. Dependable Secure Comput. 11 (1) (2014) 2–15.
[22] H. Holm, M. Ekstedt, D. Andersson, Empirical analysis of system-level vulnerability metrics through actual attacks, IEEE Trans. Dependable Secure Comput. 9 (6) (2012) 825–837.
[23] H. Holm, M. Ekstedt, T. Sommestad, Effort estimates on web application vulnerability discovery, in: 46th Hawaii International Conference on System Science (HICSS), IEEE, 2013, pp. 5029–5038.
[24] J. Homer, X. Ou, D. Schmidt, A sound and practical approach to quantifying security risk in enterprise networks, 2009. people.cis.ksu.edu.
[25] K. Ingols, M. Chu, R. Lippmann, S. Webster, S. Boyer, Modeling modern network attacks and countermeasures using attack graphs, in: Computer Security Applications Conference, 2009 – ACSAC '09, 2009 (Annual), pp. 117–126.
[26] H. Joh, J. Kim, Y.K. Malaiya, Vulnerability discovery modeling using Weibull distribution, in: Software Reliability Engineering, 2008, 19th International Symposium on ISSRE 2008, IEEE, 2008, pp. 299–300.
[27] E. Jonsson, T. Olovsson, A quantitative model of the security intrusion process based on attacker behavior, IEEE Trans. Softw. Eng. 23 (4) (1997) 235–245.
[28] H. Kerzner, Project Management: A Systems Approach to Planning, Scheduling, and Controlling, Wiley, 2009.
[29] J. Kim, Y.K. Malaiya, I. Ray, Vulnerability discovery in multi-version software systems, in: High Assurance Systems Engineering Symposium, 2007, 10th IEEE HASE'07, IEEE, 2007, pp. 141–148.
[30] G.V. Marconato, M. Kaniche, V. Nicomette, A vulnerability life cycle-based security modeling and evaluation approach, Comput. J. (2012).
[31] M. McQueen, W. Boyer, M. Flynn, G. Beitel, Time-to-compromise model for cyber risk reduction estimation, Quality Protect. (2006) 49–64.
[32] P. Mell, K. Scarfone, S. Romanosky, Common vulnerability scoring system, Security Privacy, IEEE 4 (6) (2006) 85–89.
[33] C. Miller, The legitimate vulnerability market: the secretive world of 0-day exploit sales, in: Workshop on the Economics of Information Security (WEIS), 2007, pp. 7–8.
[34] MITRE, Exposures, the standard for information security vulnerability names, Common Vulnerabilities and Exposures: The Standard for Information Security Vulnerability Names, 2007. <http://cve.mitre.org>.
[35] NIST, US National Vulnerability Database, 2013. <http://nvd.nist.gov/>, (accessed 8.2.13).

[36] M. Nyanchama, Enterprise vulnerability management and its role in information security management, Inform. Syst. Security 14 (3) (2005) 29–56.
[37] OSVDB, The Open Source Vulnerability Database, 2013. <http://www.osvdb.org/>, (accessed 8.2.13).
[38] A. Ozment, Improving vulnerability discovery models, in: Proceedings of the 2007 ACM Workshop on Quality of Protection, ACM, 2007, pp. 6–11.
[39] D.E. Perry, A.L. Wolf, Foundations for the study of software architecture, ACM SIGSOFT Softw. Eng. Notes 17 (4) (1992) 40–52.
[40] J. Radianti, E. Rich, J. Gonzalez, Vulnerability black markets: empirical evidence and scenario simulation, in: 42nd Hawaii International Conference on System Sciences, 2009 – HICSS '09, 2009, pp. 1–10.
[41] M. Shahzad, M. Shafiq, A. Liu, A large scale exploratory analysis of software vulnerability life cycles, in: 34th International Conference on Software Engineering (ICSE) 2012, 2012, pp. 771–781.
[42] T. Sommestad, M. Ekstedt, H. Holm, The cyber security modeling language: a tool for assessing the vulnerability of enterprise system architectures, Syst. J., IEEE 7 (3) (2012) 363–373.
[43] T. Sommestad, H. Holm, M. Ekstedt, Effort estimates for vulnerability discovery projects, in: 45th Hawaii International Conference on System Science (HICSS), IEEE, 2012, pp. 5564–5573.
[44] S.-W. Woo, O.H. Alhazmi, Y.K. Malaiya, An analysis of the vulnerability discovery process in web browsers, Proc. 10th IASTED SEA 6 (2006) 13–15.