THE WORK-AVERSE ATTACKER MODEL

Empirical evidence from the Symantec/WINE repository of IT attacks in the wild.
Complete Research

Allodi, Luca University of Trento, Italy, luca.allodi@unitn.it

Massacci, Fabio University of Trento, Italy, fabio.massacci@unitn.it

Abstract

In this paper we present and validate a novel attacker model based on the economic notion that the attacker has limited resources to forge a new attack. We focus on the vulnerability exploitation case, whereby the attacker has to choose whether to exploit a new vulnerability or keep an old one. We postulate that most vulnerabilities remain unattacked, and that the exploit development cycle relates to software updates rather than to the disclosure of new vulnerabilities. We develop a simple mathematical model to show the mechanisms underlying our observations and name it "The Work-Averse Attacker Model". We then leverage Symantec's data sharing platform WINE to validate our model by analysing records of attacks against more than 1M real systems. We find the 'Model of the Work-Averse Attacker' to be strongly supported by the data and, in particular, that: (a) the great majority of attacks per software version is driven by one vulnerability only; (b) an exploit lives two years before being substituted by a new one; (c) the exploit arrival rate depends on the software's update rate rather than on time or knowledge of the vulnerability.

Keywords: Attacker model; Vulnerability exploitation

1 Introduction

Many activities related to system risk management, decision making, and regulation rely on a model of the attacker. This is in accordance with the classic warfare mantra "Know your enemy" that postulates that the best strategy can only stem from a clear understanding of one's adversary. This same principle has been adopted in finance and physical security (Kaufmann, 2004), but is seldom applied in cybersecurity. From a popular and scientific standpoint, the 'hacker' is often modelled as a dedicated attacker that can and will exploit any vulnerability to get into a system (Schneier 2005; Dolev and Yao, 1983). However, recent findings point toward the opposite direction: the economic attacker relies less and less on a multitude of exploits (Nayak et al., 2014), and more on existing infrastructures and capabilities that he/she can leverage from to efficiently deploy his or her attack (Grier et al., 2012). Previous research has found a Pareto-like effect in the volume of attacks per vulnerability, showing that as little as 10% of vulnerabilities are responsible for about 90% of attacks in the wild (Allodi, 2015). Clearly, a model that thinks of the attacker as a player that exploits all vulnerabilities (Howard, Pincus, and Wing, 2003) and explores all attack paths (Wang et al., 2008) is not compatible with the aforementioned findings, according to which the attacker focuses on a handful of attacks and ignores the majority of vulnerabilities. In this regard, we distinguish two different types of attackers: 'dedicated' and 'general' attackers. The former type of attacker is a motivated one that aims at a particular system or network. In this case, the attacker's willingness to deploy the attack may justify his dedication in developing new attacks or finding alternatives paths in the target system (e.g. by means of social engineering or zero-day attacks). By definition, this attacker generates only relatively few attacks (Bilge and Dumitras, 2012) and is generally a threat only

in particular industrial or espionage scenarios. Because of his/her nature, a general model of this type of attacker is hard to make mainly for two reasons: 1) the attacker's motivation and target can not be predicted a-priori; 2) there is little or no data to validate the model. For this reason, in this scenario an adequate 'attack reaction model' (as opposed to an 'attacker model') becomes an important factor in the risk management process (Baskerville, Spagnoletti, and Kim, 2014). For the population of users at large, however, risk appears to come from a 'general' attacker that, rather than aiming at a particular user, 'phishes' from the sea of vulnerable users (Allodi, Kotov, and Massacci, 2013; Grier et al., 2012).
In this paper we focus on this second perspective, and present the '*Work-Averse Attacker Model*' as an alternative model for the general attacker that aims at compromising 'as many systems as possible' rather than 'any system'. This distinction stems from the observation that any attacker is economically bounded by some amount of effort he/she can invest in forging the attack. Because of this, the attacker needs to choose whether to exploit a new vulnerability or to keep an old one. We postulate that an attacker would not be interested in exploiting a vulnerability unless the fraction of systems it allows him to attack is significantly higher than otherwise. As a consequence, the attacker develops only few exploits and, because he/she is interested in successfully attacking a system, the rate at which he/she produces new exploits depends on the rate at which the attacked software gets upgraded, rather than strictly on time and knowledge of new vulnerabilities as otherwise often assumed in literature (Arora et al., 2004; Younis, Malaiya, and Ray, 2014). We use Symantec's records of attacks in the wild, shared through the WINE platform[1], to validate our model. To do so, we keep record of the history of attacks received by more than 1 Million Symantec costumers worldwide and use it to verify the hypotheses emerging from our model. We find strong evidence supporting the 'Work-Averse Attacker Model' and, in particular, that:

1. For any given software version, one vulnerability only drives the great majority of attacks in time;
2. An old attack gets substituted, at large, by a new one only after about two years;
3. The arrival rate of exploits depends on a software's update rate rather than strictly on time or knowledge of vulnerability.

The paper is structured as follows: Section 2 gives an overview of the relevant literature on the attacker model problem. We then present the *Work-Averse Attacker Model* and formulate the hypotheses that emerge from the model (Section 3). In Section 4 and 5 we present the data and verify our hypotheses, respectively. Finally, in Section 6 we discuss our findings' implications and conclude by briefly stating future work venues in this same direction.

2 Background

Security risk assessment is rooted in the $Risk = Impact \times Likelihood$ equation that considers the impact of an attack on the (IS) environment and the probability of the event happening (Council, 2010). Yet, the two factors are not equally easy to compute.
An assessment of the environmental *impact* of an attack is in general relatively straightforward to obtain, as the risk assessor usually has (or should have) a clear understanding of the systems his or her analysis is targeting. On the other hand, it is much trickier to obtain a precise or at least realistic estimation of the *likelihood* of the event. Differently from the impact estimation, the likelihood estimation of an attack depends on data that is hardly available to the assessor. Security data is in fact seldom available to the public, and is in general in the hands of security vendors and agencies that may have incentives in exaggerating the threat estimation (Kaufmann, 2004). Therefore, the risk assessor has to rely over a *model of the attacker* that represents the defender's 'best guess' about how the attacker behaves.
The attacker model includes considerations over the attacker's capabilities, resources, and motivation. These quantities are hard to measure and predict; as an effect, threat modelling has traditionally moved toward a 'worst case scenario' interpretation of risk where the attacker can do anything, read anything, and

[1] `http://www.symantec.com/about/profile/universityresearch/sharing.jsp`

attack every vulnerability. This interpretation is the classic one introduced in the cryptography literature (Dolev and Yao, 1983), where *assuring* complete Confidentiality, Integrity and Availability of a secured transmission is critical to validate the cryptographic protocol.

The attacker *will* therefore exploit any possible vulnerability. Consequently, the attention has moved to a model of risk where the mere presence of a vulnerability is a sign of risk, and the number and type of vulnerabilities a measure of risk. For example, Attack Graphs (Sheyner et al., 2002; Wang et al., 2008) and Attack Surfaces (Manadhata and Wing, 2011) both aim at modelling the risk of the system under assessment in terms of 'vulnerability enumeration'. Attack Graphs model the graph of vulnerabilities the attacker may traverse to reach a particular target. In the graph, each node represents a vulnerability and the edge between two vulnerabilities is weighted with the *probability* that the attacker might successfully reach the next node in the graph. The estimation of this likelihood is an open problem, that is typically addressed by means of some simple metric such as how remotely the attack may happen or how complex the exploit must be. These assessments are given by the Common Vulnerability Scoring System (CVSS) (Mell, Scarfone, and Romanosky, 2007), the *standard-de-facto* risk metric for software vulnerabilities in the industry. Attack surfaces enumerate the different ways an attacker may enter the system, and attach to the assessment a risk metric to weight relative importance of the flaws, often relying on the CVSS score (Naaliel, Joao, and Henrique, 2014). For example, according to the frameworks mentioned above, the technically easier a vulnerability is, the more likely it is considered that the attacker will exploit it. This implies that vulnerabilities with the same level of 'difficulty' are equally likely to be attacked. In other words the attacker would always be equally interested in attacking all the easiest vulnerabilities; this interpretation has clear roots in the 'worst case scenario' modelling discussed above. Moreover, most vulnerabilities have very high 'CVSS likelihoods', meaning that according to this metric the almost totality of vulnerabilities has a probability of being exploited of one (Bozorgi et al., 2010). As a consequence, the compliance to security standards, laws, and state-of-the-art practices (Council, 2010; Naaliel, Joao, and Henrique, 2014; Quinn et al., 2010) can be widely suboptimal (Allodi and Massacci, 2014).

Previous research has shown that, from the attackers' standpoint, not all vulnerabilities are the same (Nayak et al., 2014) and that a small fraction of vulnerabilities are responsible for the majority of attacks in the wild (Allodi, 2015). Similarly, Bilge et al. have recently shown that attacks against 0-day vulnerabilities[2] happen seldom in the wild representing, overall, a relatively low risk (Bilge and Dumitras, 2012). Allodi et al. showed that, once controlling for possible confoundings, very few vulnerabilities are reportedly exploited in the wild (Allodi and Massacci, 2012) and proposed a methodology to evaluate the *risk reduction* that can be obtained by addressing specific types of vulnerabilities (Allodi and Massacci, 2014). Their results confirm that, indeed, most vulnerabilities should be treated with only very low priority as they represent a negligible risk when compared to the 'risky' ones.

These empirical results head in the direction of agent economic theory (Laffont and Martimort, 2009), where agents are typically effort-minimizing and 'work-averse'. On the contrary, the IT attacker is instead assumed to be all powerful and willing to go at great lengths to compromise one's system. Yet, economic incentives and costs may play a major role in the exploit creation process, but this is not reflected in the current view of the 'cyber attacker'.

3 The Work-Averse Attacker Model

The idea that an attacker may not be interested in exploiting 'all' vulnerabilities in a system emerges from a simple observation: in most cases, he/she needs to attack only one ('powerful' enough) vulnerability among the many that affect a particular software. Indeed, vulnerability discovery and exploitation are

[2] A 0-day vulnerability is a vulnerability for which an exploit in the wild exist *before* the disclosure date of the vulnerability. From the viewpoint of this paper, a 0-day vulnerability represents a case in which the attacker has *more* information about the vulnerable system than the defender has access to.

resource-consuming processes for the attacker (Miller, 2007), and can be economically expensive (Grier et al., 2012).

In a broader sense, the expected utility of an exploit for a vulnerability v at time t $E[U_{t,v}]$ comes from the revenue r the attacker can extract from the fraction $n(t,v) \in [0,N]$ of the N systems in the wild the vulnerability allows him to attack at time t. The revenue r an attacker can get from the system out of the exploitation of one vulnerability may depend on two factors:

1. The potential value of the attacked system.
2. The impact I of the vulnerability on the system. For example, a vulnerability granting full administrative access is likely to allow the attacker to extract more revenue from the attacked system.

We therefore model the extracted revenue per attacked system $r(I(v_i))$ as a function of the vulnerability impact. The cost c of the attack comprises the cost of developing/buying the exploit and the cost of delivering the attack by means, for example, of some attacking infrastructure (Allodi, Kotov, and Massacci, 2013; Grier et al., 2012).

The expected utility of an exploit for a vulnerability v at time t is therefore:

$$E[U_{t,v}] = n(t,v) \times r(I(v)) - c(v) \tag{1}$$

Note that $lim_{t \to \infty} n(t,v) = 0$ as users update their systems and the exploit for v loses efficacy in the wild. When the efficacy of the old exploit drops too low, the attacker will dedicate his/her resources (abandoning v)[3] to look for a new exploit v'.

Under the assumption that exploit development is costly and therefore an attacker is work averse, s/he will develop the exploit for a new vulnerability v' after some time $t+\delta > t$ if the expected value for v at $t+\delta$ is lower than the expected value for v' at $t+\delta$:

$$E[U_{t+\delta,v'}] - E[U_{t+\delta,v}] > 0 \tag{2}$$

The boundary condition to choose v' is therefore:

$$n(t+\delta,v') \times r(I(v')) - c(v') > E[U_{t+\delta,v}] \tag{3}$$

By generalising Eq. 3 it is possible to obtain the decision condition for the attacker over an arbitrary vulnerability v_j

$$n(t+\delta,v_j) \times n \times r(I(v_j)) - c(v_j) > \sum_{i=1}^{j-1} E[U_{t+\delta,v_i}] \tag{4}$$

At this point, the attacker will introduce a new exploit for v_j if:

$$c(v_j) < n(t+\delta,v_j) \times r(I(v_j)) - \sum_{i=1}^{j-1} E[U_{t+\delta,v_i}] \tag{5}$$

The cost for v_j is therefore bounded by the revenue that can be extracted from all pre-existing exploits the attacker may maintain. It is immediate to see that the more previous exploits have been developed, the higher the potential revenue from v_j must be in order to overcome the cost constraint. With $\sum_{i=1}^{j-1} E[U_{t+\delta,v_i}]$ growing with the number of available exploits, the upper-bound cost $c(v_j)$ for the new exploit tends to zero. The diminishing return seen in Eq. 5 has two main consequences:

1. The attacker is able to afford a diminishing amount of exploits in time.

[3] The vulnerability finding and exploit writing processes are very time consuming and require the allocation of plenty of resources (Allodi, Kotov, and Massacci, 2013; Grier et al., 2012; Miller, 2007). While an attacker can always re-use old technology (i.e. old exploits), maintaining a certain exploit operative requires maintenance costs in terms of both technological resources and time. When not looking for a new exploit, we do not put any constraint on how many exploits the attacker wants to use.

2. Assuming a direct relationship between exploit quality and cost of the exploit (Arora et al., 2010; Miller, 2007), the quality of the new exploits would tend to decrease with the amount of exploits available to the attacker.

The cost diverges from zero only when $n(t+\delta, v_j)$ is greater than $\sum_{i=1}^{j-1} n(t+\delta, v_i)$ because $\lceil n(t+\delta, v_j) + \sum_{i=1}^{j-1} n(t+\delta, v_i) \rceil \leq N$, so there is a cap on the total revenue that can be extracted. In other words, the attacker will build a new reliable exploit only when the overall revenue the attacker can extract from the old exploits drops because of too few vulnerable systems in the wild (i.e. because users at large upgraded their systems). This set-up leads us toward formulating two main hypotheses about the attacker. First, because of the cost constraint in Eq. 5,

Hypothesis 1 *The attacker will massively use only one exploit per software version.*

If Hyp. 1 holds, the introduction of a new exploit depends on the rate at which users update their software. Because update rates are generally low (Howe et al., 2012), we hypothesise the following:

Hypothesis 2 *The fraction of attacks driven by a particular vulnerability will decrease slowly in time.*

Update rates and software types. From Hyp. 1 and 2 we argue that the average user behaviour in updating a system determines the rate at which the efficacy of an exploit for a vulnerability $n(t, v_j)$ declines. However, not all software is updated at the same pace both on the vendor side (that is slower in developing the patches (Schryen, 2009) and the users' side (that may be more likely to apply available patches for a software type than for another (Howe et al., 2012)). Lately, some software (e.g. internet browsers) started adopting a 'quick development cycle' (Nguyen and Massacci, 2012) that quickly patches vulnerabilities and sends automatic updates to the users. The attacker behaviour may change with respect to the software type. For example, users may seldom update their Java plugin, whereas they run the latest version of the Internet Explorer browser.

Corollary to Hyp. 2 The attacker waits a longer period of time to introduce an exploit for software types under a slow update cycle than for others.

4 Data collection

To test our model's predictions we rely on data on attacks recorded in the wild by Symantec's sensors worldwide. Symantec shares the data with interested researchers through their Worldwide Intelligence Network Environment (WINE) initiative (Dumitras and Efstathopoulos, 2012). Our dataset was collected in July 2013 and is available for sharing at Symantec Research Labs under the reference *WINE-2012-008*. WINE consists of records of attacks Symantec sensors' detect in the wild. In particular, WINE is a representative, anonymized sample of the operational data Symantec collects from users that have opted in to share telemetry data, and comprises attack data from more than 1 Million hosts (Dumitras and Shou, 2011).

Attacks are identified by an internal unique *ID* that references a particular threat. Because not all threats require the exploitation of a software vulnerability (e.g. phishing attacks, social engineering, etc.), we map WINE's attack signature with the threat description publicly available at Symantec's Security Response dataset[4], from where we also extract the vulnerability's unique identifier (CVE-ID)[5].

By matching the CVE-ID with the vulnerability description reported in the National Vulnerability Database[6], we obtain information about the software and the last software version that is affected by the vulnerability. Because some software may be subject to different attack patterns than others, we

[4] `https://www.symantec.com/security_response/`

[5] http://cve.mitre.org

[6] `http://nvd.nist.gov`

Category	Sample of software names	No. Vulns	No. Software	No. Versions
Internet Explorer	MS internet explorer	30	1	8
PROD	personal_firewall_2, backup_exec	26	20	21
PLUGIN	flash_player, air	30	7	17
SERVER	hp-ux, isa_server	34	20	22
Tot		**120**	**48**	**68**

Table 1. Overview of the used dataset by software category. Internet Explorer has been singled out because it illustrates the attacker's work aversion both in the large (for the ensemble) and in the small (for individual attacks).

1st attack	2nd attack	Delta days	Affected machines	Volume of attacks
a	a	192	23544	58322
b	c	11	6	6
b	e	580	10	10
d	f	861	389	432
e	b	644	26	43

Table 2. Excerpt from our dataset. CVE-IDs are obfuscated as a, b, c, etc. Each <`1st attack, 2nd attack, delta`> tuple is unique in the dataset. The column `Affected machines` reports the number of unique machines receiving the second attack delta days after `1st attack`. The column `Volume of attacks` is constructed similarly but for the number of received attacks.

further categorise the vulnerabilities in our dataset in four categories: Internet Explorer, Server, Production Software and Plugin. We do not consider the case of vulnerabilities affecting the Windows operating system or some of its components: WINE assigns unique IDs on a per-system basis, and therefore identifies the system and not its user. When a user buys a new system or upgrades his/her Windows installation, his/her WINE ID may change. For this reason we can not reliably keep track of users throughout upgraded Windows versions. We use IE (Microsoft's Internet Explorer) as a representative of the browser's category. Table 1 reports the details of the remaining software categories we consider for our analysis. Our study comprises 120 vulnerabilities across 48 different software and 68 software versions. Our data spans from July 2009 to July 2012. The data collection took place in July 2013 but we decided to limit the data window because an update of the operating system (including moving to a difference Service Pack) would assign a different WINE ID to each user.

4.1 Data Preparation

To build our dataset, we first reconstruct the history of attacks received by every user in WINE. To evaluate the sequence of attacks against a certain software, we then collect all the pairs $< attack1, attack2 >$ of attacks that a user received, and keep track of the time delay (measured in days) between the two attacks. We then group the data by pairs of attacks and delta in time, and count how many users have been affected by that sequence and how many attacks of that type have been observed in the wild. Table 2 reports an excerpt from the dataset. Each row represents a succession of attacks. The first column and the second column report respectively the (censored) CVE-ID of the attacked vulnerability in the first and in the second attack. The third column reports the number of unique systems in the WINE platform affected at least once by that tuple; the fourth column reports the overall number of attacks detected; the fifth the distance between the two attacks expressed in days. Note that for anonymity reasons we aggregate the attacks against each unique machine in WINE into an ensemble of identical attacks. The columns reporting the software affected by the vulnerability, the latest affected software version and the software's category are here omitted for brevity.

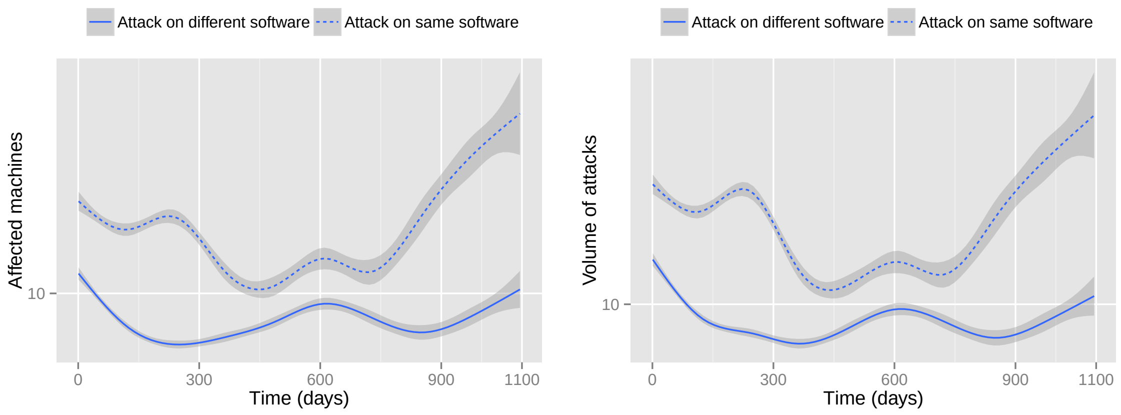

Figure 1. Regression of number of attacked machines (left) and volume of attacks (right) as a function of time. Attacks against the same software are represented by the dashed line; attacks against different software are represented by the solid line. Regression generated by fitting to a generalized additive model. Shaded areas represent 95% confidence intervals around the median.

To study attackers' attitude at creating new exploits, we only consider the tuples <`1st attack, 2nd attack`> respecting the following constraints:

1. The vulnerability exploited in the first attack was disclosed before or at most 120 days after the vulnerability for the second attack. This robustly large interval has been chosen according to how the vulnerability disclosure process works
2. The second exploited vulnerability is less than three years older than the first. We choose this time frame as it matches the length of the historic records we have for each WINE user, given the three-year interval covered by our sample. This also avoids comparisons between very old and new vulnerabilities that would add little to the discussion in terms of attackers' preferences.

In the first row of Table 2 the two subsequent attacks are against the same vulnerability. The tuple $< a, a >$ affected most machines and was the vector of a high number of attacks in the sample. For example, we find almost 60 thousand attacks against 23.5 thousand users that have received a second attack against a 192 days (6months) after the first. The second and third row report two instances where an attack on b has been followed by an attack against two other vulnerabilities. The third and fourth rows report other two combinations.

5 Analysis

We first give an overview of our dataset. Figure 1 shows a generalized regression (Hastie and Tibshirani, 1986) of attacked systems (left) and volume of attacks (right) as a function of time. The shaded areas represent the 95% confidence intervals around the fitted line. Subsequent attacks directed towards the same software are represented by the dashed line. Subsequent attacks against different software are represented by the solid line. We observe that the distribution of attacked machines (left) follows closely the distribution of recorded attacks (right). In this study we will consider only the number of affected machines, as this gives us a more direct measure of how many users are affected by a certain attack. We keep a closer analysis of volume of attacks for future work. We further observe that subsequent attacks against the same software are more frequent than subsequent attacks against different software. This is intuitive as the received attack depends on the software usage habits of the user. For example, a user that uses his/her system to navigate the Internet might be more prone in receiving attacks against Internet

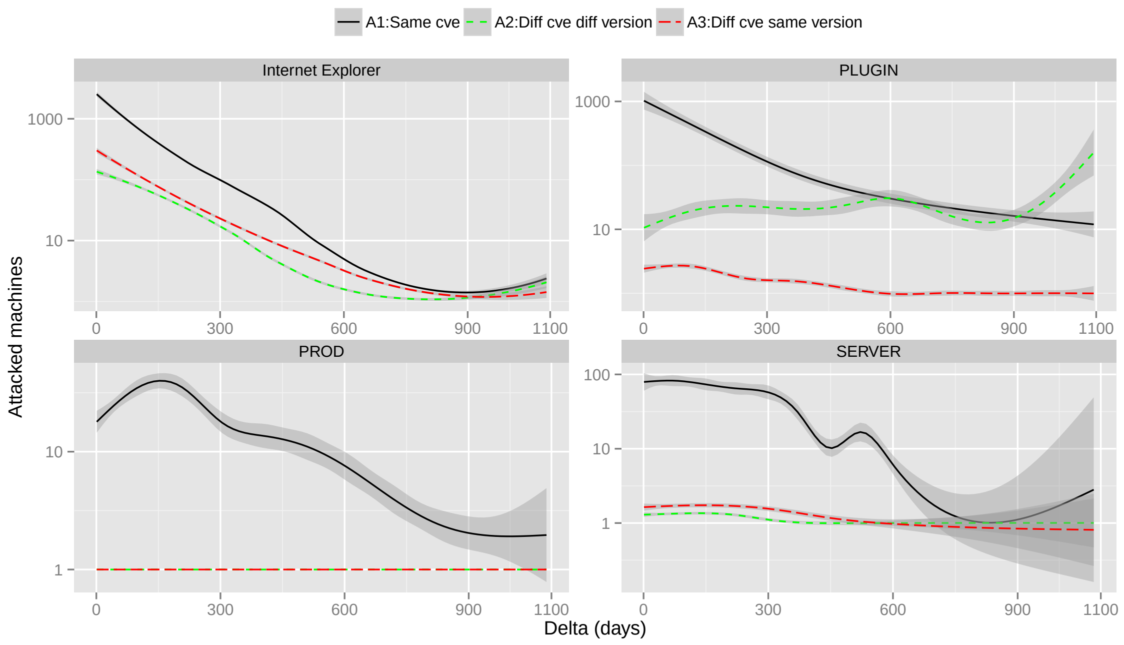

Figure 2. Targeted machines as a function of time for the three types of attack. A_1 is represented by a solid black line; A_2 by a long-dashed red line; A_3 by a dashed green line.

Explorer than against Microsoft Office. Because of this we will focus in this study on subsequent attacks against the same software. This will allow us to assess the attacker's attitude toward creating new exploits for the same software platform. We further observe that the fitted curves do not have a clear positive or negative slope as functions of time. This suggests that attacks are only weakly correlated with time, and other factors (such as users' patching attitudes, or just technological chances) may explain the trend.

Hypothesis 1. To check the veracity of Hyp. 1 we evaluate how many users receive two attacks, after a certain δt, of either of these types:

1. A_1=$A(cve = cve' | \delta t \;\&\; sw = sw')$: Against the same vulnerability and same software version.
2. A_2=$A(cve < cve' \& \; vers \neq vers' | \delta t \;\&\; sw = sw')$: Against a new vulnerability and a different software version.
3. A_3=$A(cve < cve' \& \; vers = vers' | \delta t \;\&\; sw = sw')$: Against a new vulnerability and same software version.

In accordance with Hyp. 1 the attacker should prefer to (a) attack the same vulnerability multiple times, and (b) create a new exploit when he/she wants to attack a new software version. Therefore, according to Hyp. 1 we expect the following ordering in the data to be generally true: $A_3 < A_2 < A_1$. An exception may be represented by SERVER vulnerabilities: SERVER environments are typically better maintained than 'consumer' environments, which may affect an attacker's attitude toward developing new exploits. For example, SERVER software is often protected by perimetric defences such as firewalls or IDSs. This may require the attacker to engineer different attacks for the same software version in order to escape the additional mitigating controls in place. For this reason we expect the difference between A_2 and A_3 to be narrower or reversed for the SERVER category.

Figure 2 reports a fitted regression of targeted machines as a function of time by software category. As expected, A_1 dominates in all software types. The predicted order is valid for PLUGIN and PROD. For PROD software we find no attacks against new vulnerabilities for different software versions, therefore

$A_2 = A_3 = 0$. This may be an effect of the typically low update rate of this type of software and relatively short timeframe considered in our dataset (3 years), or of a scarce attacker interest in this software type. Results for SERVER are mixed as discussed above: the difference between A_2 and A_3 is very narrow and A_3 is higher than A_2: attackers forge more exploits per SERVER software version than for other types of software.

Internet Explorer. Internet Explorer is an interesting case in itself. Here, contrary to our prediction, A_3 is higher than A_2. By further investigating the data, we find that the reversed trend is explained by one single outlier tuple: <*CVE-2010-0806,CVE-2009-3672*>. Both these CVEs refer to vulnerabilities affecting Internet Explorer version 7. The two vulnerabilities have been disclosed 98 days apart, 22 days short of our 120 days threshold. More interestingly, these two vulnerabilities are very similar, as they both affect a memory corruption bug in Internet Explorer 7 that allows for an heap-spray attack that may result in arbitrary code execution[7]. Two observations are particularly interesting to make:

1. Heap spray attacks are unreliable attacks that may result in a significant drop in exploitation success. This is reflected in the "Access Complexity=Medium" assessment assigned to both vulnerabilities by the CVSS v2 framework. In our model, this is reflected in a lower $n(v,t)$ value, as the unreliable exploit may affect less machines than those that are vulnerable.
2. The exploitation code found on Exploit-DB[8] is essentially the same for these two vulnerabilities. The code for CVE-2010-0806 is effectively a rearrangement of the code for CVE-2009-3672, with different variable names. In our model, this would indicate that the cost $c(v)$ to build an exploit for the second vulnerability is negligible, as most of the exploitation code can be re-used from the old vulnerability.

Having two independent but unreliable exploits that affect the same software version increases the chances of a successful attack, $n(v,t)$. Because the second exploit comes at a very low cost $c(v)$, the attacker chooses to exploit the second vulnerability as well as in this case the combination of the two exploits yields, by setting $c(v_2) = 0$:

$$c(v_1) < [n(t+\delta, v_1) + n(t+\delta, v_2)] \times R(I(v_1)) - \sum_{i \neq \{1,2\}} E[U_{t+\delta,v_i}] \tag{6}$$

Eq. 6 shows that, at the cost of one exploit, the attacker gets the combined fraction of successful attacks[9] of both vulnerabilities. Moreover, Internet Explorer is used by a significant fraction of Internet users[10], therefore $n(t+\delta, v_1) + n(t+\delta, v_2)$ may be particularly interesting for the attacker.

Although this vulnerability is an exception in the data, the existence of the second exploit for Internet Explorer 7 is coherent with our model and ultimately supports our thesis that an attacker would build an exploit only if the additional cost is balanced by an increased rate of successful attacks over his/her current capability.

Table 3 reports the results of the analysis for Hyp. 1 with the exclusion of the Internet Explorer outlier, as discussed above. Significance is given by a Wilcoxon paired test. All comparisons but SERVER accept the alternative that $A_2 < A_1$ and $A_3 < A_2$. Overall, we find strong statistical evidence supporting Hyp 1.

Hypothesis 2. We now check how the trends of attacks against a software change with time. Hyp. 2 states that the exploitation of the same vulnerability persists in time and decreases slowly at a pace

[7] CVE-2009-3672: `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-3672`
CVE-2010-0806: `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-0806`

[8] Exploit-DB is a public dataset for vulnerability proof-of-concept exploits.
CVE-2009-3672: `http://www.exploit-db.com/exploits/16547/`
CVE-2010-0806: `http://www.exploit-db.com/exploits/11683/`

[9] Note that because v_1 and v_2 are vulnerabilities of the same type, then $R(I(v_1)) = R(I(v_2))$.

[10] `http://www.w3counter.com/trends`

Category	Test	Significance
Internet Explorer	$A_2 < A_1$	***
Internet Explorer	$A_3 < A_2$	***
PROD	$A_2 < A_1$	***
PROD	$A_3 < A_2$	-
PLUGIN	$A_2 < A_1$	***
PLUGIN	$A_3 < A_2$	***
SERVER	$A_2 < A_1$	***
SERVER	$A_3 < A_2$	

Table 3. Results for Hypothesis 1. Significance (***) is reported for $p < 0.01$.

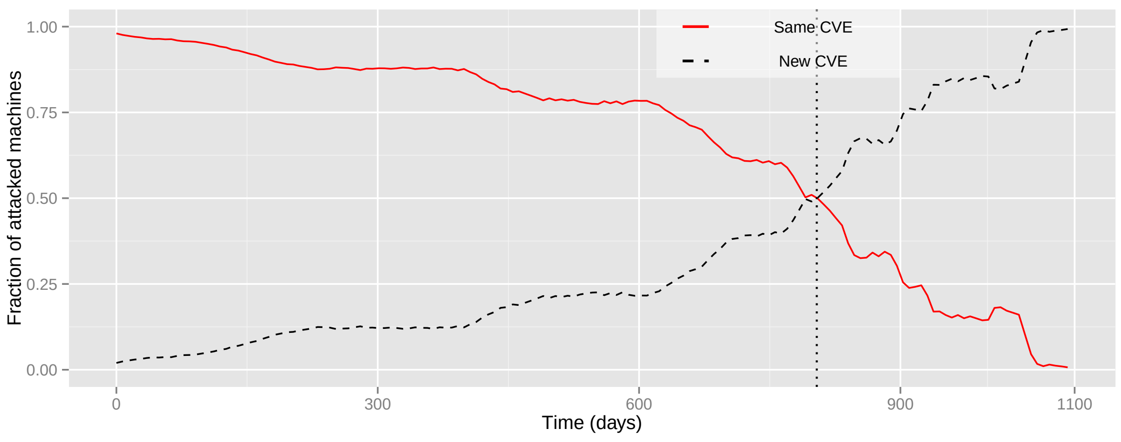

Figure 3. Fraction of systems receiving the same attack repeatedly in time (red, solid) compared to those receiving a second attack against a different vulnerability (black, dashed). The vertical line indicates the amount of days after the first attacks where it becomes more likely to receive an attack against a new vulnerability rather than against an old one.

depending on users' update behaviour. This is in contrast with other models in literature where new exploits arrive very quickly after the date of disclosure, and attacks increase following a steep curve (Arora et al., 2004).
Figure 3 reports the fraction of systems receiving, once an attack arrived, a subsequent attack against the same vulnerability (red, solid) as opposed to an attack against a different vulnerability (black, dashed). The x-axis reports the elapsed time since the first attack, in days. As hypothesised, the rate at which the same attack arrives decreases slowly with time and is still 20% after almost three years (1000 days). Notably, the event of receiving an attack against a different vulnerability becomes more likely than its counterpart only 800 days (or 2 years, see dotted vertical line in Figure 3) after the first attack happens. This is interesting in itself as it indicates that attackers use the same exploit for a long period of time before substituting it at scale with a new one.

5.1 Robustness check

The distribution reported in Figure 3 depends on users' patching attitudes. In particular, according to the model presented here, software that is patched more often should see a quicker arrival rate of new exploits in time. We expect that software that is more rarely updated by users receives attacks against new vulnerabilities with a larger delay than software that is updated more often.
To the best of our knowledge there is no available data on the average rate at which users update different software types. However, as previously discussed, we expect SERVER software to be patched regularly

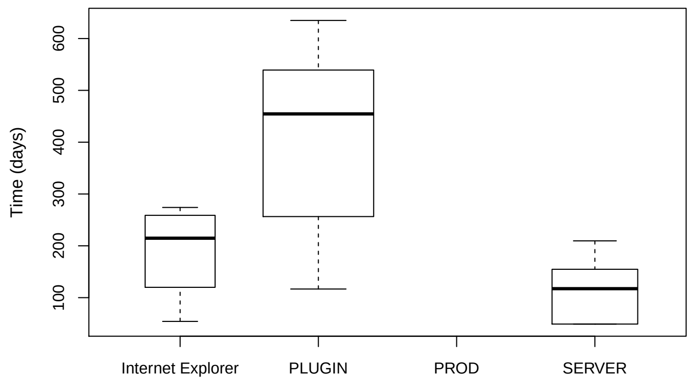

Figure 4. Distribution of average days between first exploit attempt and the appearance of an attack attempting to exploit a different vulnerability in the respective category.

(Quinn et al., 2010), and to be generally maintained better than consumer software. Therefore, we expect the arrival of new exploits to be quicker for SERVER vulnerabilities than for other software types. This would also be coherent with the results in Table 3, as for SERVER $A_3 \geq A_2$ (i.e. the attacker does not wait for a new version to build a new exploit). We expect Internet Explorer to be fairly often updated as Microsoft releases patches every month and automatically pushes it to the users via the Microsoft Update system. PLUGIN software is traditionally seldom updated by the users, as only very recently a few PLUGIN vendors started pushing update notifications. Still, we expect PLUGIN exploits to arrive on average later in time than for other categories. As discussed previously, we have no data on subsequent attacks against PROD software affecting different vulnerabilities.
Figure 4 reports the distribution of days for the appearance of a new attack for each software in the respective category. The delay for the appearance of a new exploit for PLUGIN software is the highest one ($p = 0.02$), with a median arrival delay of 454 days since first exploit. New exploits for Internet Explorer vulnerabilities arrive with a median delay of 214 days. SERVER attacks are the quickest to arrive, with a median delay of 117 days, but the difference with Internet Explorer is statistically significant for the alternative "SERVER exploits arrive faster than for Internet Explorer" at the 10% confidence level ($p = 0.08$).

6 Discussion and conclusion

In the present work we discussed the *Model of the Work-Averse Attacker* as a new model to understand cyber threats. Our proposal is attacker-centric and models the attacker as a resource-limited actor that has to choose which vulnerabilities to exploit. We here only address the general case where the attacker aims at the 'mass of systems' in the wild In the 'general threat' case, the cost constraints emerging from the model prevent the attacker from 'exploiting all vulnerabilities' as otherwise currently assumed in academia and industry alike. We supported our claims with evidence from attacks recorded in the wild. Evidence markedly points in the direction of the predictions our model makes. In particular, we find that:

1. An attacker massively deploys only one exploit per software version. The only exception we find is characterised by:
    - A very low cost to create an additional exploit, where it is sufficient to essentially copy and paste code from the old one, with little modifications, to obtain the new one.

- An increased chance of delivering a successful attack.

2. The attacker deploys new exploits slowly in time; after three years the same exploits still drive about 20% of the attacks.
3. The speed of arrival of new exploits only weakly correlates with time, but shows a strong dependency on software patching rates.

Our findings suggest that the rationale behind vulnerability exploitation could be leveraged by defenders to deploy more efficient security countermeasures. For example, it is well known that software updates correspond to an increased risk of service disruption (e.g. for incompatibility problems or updated/deprecated libraries). However, if most of the risk for a particular software version comes from a specific vulnerability, than countermeasures other than patching may be more cost-efficient. For example, maintaining network IDS signatures may be in this case a better option than updating the software, because one IDS signature could get rid of the great majority of risk that characterises that system while a software patch may 'overdo it' by fixing more vulnerabilities than necessary.
Of course, the attacker may react to changing defenders' behaviour: in the game-theoretic view of the problem, the defender always moves first and therefore the attacker can adapt his/her strategy to overcome the defenders'. This is an unavoidable problem in security that is common to any threat mitigation strategy. A more precise and data-grounded understanding of the attacker poses nonetheless a strategic advantage for the defender. For example, software diversification and code differentiation has already been proposed as a possible alternative to vulnerability mitigation (Chen, Kataria, and Krishnan, 2011; Homescu et al., 2013). By diversifying software the defender effectively decreases the fraction $n(t,v)$ of systems the attacker can compromise with one exploit. If the risk over a software version comes from only one vulnerability, then a possible counter-strategy to the attackers' adaptive behaviour is to first patch the high risk vulnerability, and then randomise the additional defences against the remaining vulnerabilities to minimize the attacker's chances of choosing the 'right' exploit to develop (as the attacker's multiple targets will likely choose a different set of vulnerabilities to patch). Diversifying defences may be in fact less onerous than re-compiling code bases (when possible) (Homescu et al., 2013) or maintaining extremely diverse operational environments (Chen, Kataria, and Krishnan, 2011).

Future work. Our model may as well have some foresight power in attributing a certain 'exploit likelihood' score to a vulnerability. Although the exploit selection process is a stochastic one, it could be possible to foresee which vulnerabilities may be more interesting to the attacker on the basis of what exploits are currently available in the wild, and what is the software affected by the vulnerability. Additional model factors like 'potential revenue' can be estimated by the impact of the vulnerability, and its exploitation complexity (as measured by the CVSS score) could be an indication of how reliable an exploit could be for the attacker (e.g. in the form of a discounting factor α for $n(t,v)$). We keep these pointers for future work.

Acknowledgements

The authors thank Matthew Elder at Symantec Corp. for his many useful comments. This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 285223 (SECONOMICS) and from the Italian PRIN Project TENACE. Our results can be reproduced by utilizing the reference data set *WINE-2012-008*, archived in the WINE infrastructure.

References

(Schneier 2005). URL: `https://www.schneier.com/blog/archives/2005/12/weakest_link_se.html`.

Allodi, L. (2015). "The Heavy Tails of Vulnerability Exploitation." In: *Proceedings of the 2015 Engineering Secure Software and Systems Conference (ESSoS'15).*

Allodi, L., V. Kotov, and F. Massacci (2013). "MalwareLab: Experimentation with Cybercrime attack tools." In: *Proceedings of the 2013 6th Workshop on Cybersecurity Security and Test.*

Allodi, L. and F. Massacci (2012). "A Preliminary Analysis of Vulnerability Scores for Attacks in Wild." In: *Proceedings of the 2012 ACM CCS Workshop on Building Analysis Datasets and Gathering Experience Returns for Security.*

— (2014). "Comparing vulnerability severity and exploits using case-control studies." *ACM Transaction on Information and System Security (TISSEC)* 17 (1).

Arora, A., R. Krishnan, A. Nandkumar, R. Telang, and Y. Yang (2004). "Impact of vulnerability disclosure and patch availability-an empirical analysis." In: *Proceedings of the 3rd Workshop on Economics and Information Security.*

Arora, A., R. Krishnan, R. Telang, and Y. Yang (2010). "An Empirical Analysis of Software Vendors; Patch Release Behavior: Impact of Vulnerability Disclosure." *Information Systems Research* 21 (1), 115–132. DOI: 10.1287/isre.1080.0226.

Baskerville, R., P. Spagnoletti, and J. Kim (2014). "Incident-centered information security: Managing a strategic balance between prevention and response." *Information & Management* 51 (1), 138 –151.

Bilge, L. and T. Dumitras (2012). "Before we knew it: an empirical study of zero-day attacks in the real world." In: *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS'12).* Raleigh, North Carolina, USA: ACM, pp. 833–844.

Bozorgi, M., L. K. Saul, S. Savage, and G. M. Voelker (2010). "Beyond Heuristics: Learning to Classify Vulnerabilities and Predict Exploits." In: *Proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining.*

Chen, P.-y., G. Kataria, and R. Krishnan (2011). "Correlated failures, diversification, and information security risk management." *MIS Quaterly-Management Information Systems* 35 (2), 397–422.

Council, P. (2010). *PCI DSS Requirements and Security Assessment Procedures, Version 2.0.* URL: https://www.pcisecuritystandards.org/documents/pci_dss_v2.pdf (visited on 10–2010).

Dolev, D. and A. Yao (1983). "On the security of public key protocols." *IEEE Transactions on Information Theory* 29 (2), 198 –208.

Dumitras, T. and P. Efstathopoulos (2012). "Ask WINE: are we safer today? evaluating operating system security through big data analysis." In: *Proceeding of the 2012 USENIX Workshop on Large-Scale Exploits and Emergent Threats.* LEET'12. San Jose, CA, pp. 11–11.

Dumitras, T. and D. Shou (2011). "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.* ACM, pp. 89–96.

Grier, C., L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M. Z. Rafique, M. A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, and G. M. Voelker (2012). "Manufacturing compromise: the emergence of exploit-as-a-service." In: *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS'12).* ACM, pp. 821–832.

Hastie, T. and R. Tibshirani (1986). "Generalized additive models." *Statistical science*, 297–310.

Homescu, A., S. Neisius, P. Larsen, S. Brunthaler, and M. Franz (2013). "Profile-guided automated software diversity." In: *2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).* IEEE, pp. 1–11.

Howard, M., J. Pincus, and J. M. Wing (2003). "Measuring Relative Attack Surfaces." In: *Proceedings of Workshop on Advanced Developments in Software and Systems Security.*

Howe, A., I. Ray, M. Roberts, M. Urbanska, and Z. Byrne (2012). "The Psychology of Security for the Home Computer User." In: *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, pp. 209–223.

Kaufmann, C. (2004). “Threat inflation and the failure of the marketplace of ideas: The selling of the Iraq war.” *International Security* 29 (1), 5–48.

Laffont, J.-J. and D. Martimort (2009). *The theory of incentives: the principal-agent model*. Princeton University Press.

Manadhata, P. K. and J. M. Wing (2011). “An Attack Surface Metric.” *IEEE Transactions on Software Engineering* 37, 371–386. ISSN: 0098-5589. DOI: http://doi.ieeecomputersociety.org/10.1109/TSE.2010.60.

Mell, P., K. Scarfone, and S. Romanosky (2007). *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. Tech. rep. FIRST, Available at http://www.first.org/cvss.

Miller, C. (2007). “The legitimate vulnerability market: Inside the secretive world of 0-day exploit sales.” In: *Proceedings of the 6th Workshop on Economics and Information Security*.

Naaliel, M., D. Joao, and M. Henrique (2014). “Security Benchmarks for Web Serving Systems.” In: *Proceedings of the 25th IEEE International Symposium on Software Reliability Engineering (ISSRE’14)*.

Nayak, K., D. Marino, P. Efstathopoulos, and T. Dumitraş (2014). “Some Vulnerabilities Are Different Than Others.” In: *Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses*. Springer, pp. 426–446.

Nguyen, V. H. and F. Massacci (2012). “An Independent Validation of Vulnerability Discovery Models.” In: *Proceeding of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS’12)*.

Quinn, S. D., K. A. Scarfone, M. Barrett, and C. S. Johnson (2010). *SP 800-117. Guide to Adopting and Using the Security Content Automation Protocol (SCAP) Version 1.0*. Tech. rep. National Institute of Standards & Technology.

Schryen, G. (2009). “A Comprehensive and Comparative Analysis of the Patching Behavior of Open Source and Closed Source Software Vendors.” In: *Proceedings of the 2009 Fifth International Conference on IT Security Incident Management and IT Forensics*. IMF ’09. Washington, DC, USA: IEEE Computer Society, pp. 153–168. ISBN: 978-0-7695-3807-5. DOI: 10.1109/IMF.2009.15. URL: http://dx.doi.org/10.1109/IMF.2009.15.

Sheyner, O., J. Haines, S. Jha, R. Lippmann, and J. M. Wing (2002). “Automated Generation and Analysis of Attack Graphs.” *Proceedings of the 29th IEEE Symposium on Security and Privacy* 0, 273.

Wang, L., T. Islam, T. Long, A. Singhal, and S. Jajodia (2008). “An Attack Graph-Based Probabilistic Security Metric.” In: *Proceedings of the 22nd IFIP WG 11.3 Working Conference on Data and Applications Security*. Vol. 5094. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 283–296.

Younis, A. A., Y. K. Malaiya, and I. Ray (2014). “Using Attack Surface Entry Points and Reachability Analysis to Assess the Risk of Software Vulnerability Exploitability.” In: *Proceeding of the 15th IEEE International Symposium on High Assurance Systems Engineering*, pp. 1–8. DOI: 10.1109/HASE.2014.10.