Patch Me If You Can: A Study on the Effects of Individual User Behavior on the End-Host Vulnerability State

Armin Sarabi[1], Ziyun Zhu[2], Chaowei Xiao[1],
Mingyan Liu[1], and Tudor Dumitraș[2]

[1] University of Michigan, Ann Arbor
{arsarabi,xiaocw,mingynan}@umich.edu
[2] University of Maryland, College Park
{zhuziyun,tdumitra}@umiacs.umd.edu

Abstract. In this paper we study the implications of end-user behavior in applying software updates and patches on information-security vulnerabilities. To this end we tap into a large data set of measurements conducted on more than 400,000 Windows machines over four client-side applications, and separate out the impact of user and vendor behavior on the vulnerability states of hosts. Our modeling of users and the empirical evaluation of this model over vulnerability states of hosts reveal a peculiar relationship between vendors and end-users: the users' promptness in applying software patches, and vendors' policies in facilitating the installation of updates, while both contributing to the hosts' security posture, are overshadowed by other characteristics such as the frequency of vulnerability disclosures and the vendors' swiftness in deploying patches.

1 Introduction

Software vulnerabilities represent a valuable resource for attackers. Exploits for these vulnerabilities can allow miscreants to control the vulnerable hosts remotely. Unpatched vulnerabilities also present a threat for enterprises, as an outward facing machine with an exploitable vulnerability can provide unauthorized access to the company's internal network [26]. Moreover, the emergence of exploit kits [14], makes it easy for attackers to compromise hosts in an automated fashion. To counter these threats, software vendors create and disseminate patches that users then install to remove vulnerabilities on their machines. Vendors have also increased the automation of their software updating mechanisms [9,13] in an attempt to accelerate the patching process to sidestep possible tardiness on the part of the end users.

It follows that the vulnerability state of any given end-host at any given time, reflected in the number of *known but unpatched vulnerabilities*, and *unpatched vulnerabilities with known exploits*, is the result of a combination of factors, including (1) the user's updating behavior, (2) the software products' patch release timeliness with respect to the disclosure of vulnerabilities, (3) the update

	Findings	Implications
+	The user behavior can be summarized using single parameter distributions.	Users' willingness to patch does not seem to depend on the type of improvements in new releases.
+	The geometric distribution provides a good fit, even for products with silent updates.	This simple model significantly simplifies the analysis of the relationship between user behavior and the vulnerability state of their machines.
+	Silent updates lead to shorter windows of vulnerability for end-hosts (as expected).	The product vendors can improve the vulnerability state by adopting a silent updating mechanism.
-	Even with silent updates, the majority of hosts have long windows of vulnerability.	The large number of security flaws in client-side applications limits the benefits of silent updates.
-	Many hosts have long windows of susceptibility to known exploits.	Exploit kits present a direct threat to these hosts.

Table 1: Summary of findings. +/- indicate positive and negative impacts.

mechanisms employed to deploy patches on hosts, and (4) the frequency at which vulnerabilities are disclosed and exploits are developed. While the latter three elements have been extensively studied in the literature—see e.g., [2–5, 7, 8, 18, 20, 22, 25] on vulnerability disclosure and patch releases, [11, 17, 21, 23, 29] on patch deployment, and [4, 6, 14, 24] on exploits—relatively less is known about the impact of individual user behavior. Prior work in this area has introduced several hypotheses on why users might delay patching vulnerabilities [15, 16, 28], and aggregated patching measurements for individual vulnerabilities over the general population and over selected groups of users [17].

In this paper, we present a broad field study of individual user behavior, including more than 400,000 users over a period of 3 years (01/2010 to 12/2012), and their updating activities concerning 1,822 vulnerabilities across 4 software products. The updating automation for these applications ranges from prompting users to install patched versions to silent updates, which require minimal user interaction. Our goal is to understand (1) how users behave on an individual level, and (2) how different updating behaviors relate to the vulnerability state of their machines, and how this relationship differs across products.

To achieve the above goal, we employ a combination of empirical analysis and mathematical modeling. In summary, our main contributions are as follows. We propose methods for quantifying the user updating behavior from field measurements of patch deployment. Furthermore, we conduct a systematic study of vulnerability patching, from the perspective of individual users (rather than individual vulnerabilities), and quantify the corresponding vulnerability state of the users' machines. Finally, building on insights from our measurements, we create and evaluate a parameterized model for individual patching behavior, and discuss its implications for end-host security. Table 1 summarizes our findings.

2 Data sets and their processing

We utilize a corpus of patch-deployment measurements collected by Nappa et al., on user hosts that include average users, as well as professionals, software

developers, and security analysts, and mostly consist of Windows XP/Vista/7 machines [17]. These measurements were conducted by observing the installation of subsequent versions of different applications, and are derived from the WINE data set [10]. The set of security flaws affecting each version are extracted from NVD [19], using the CVE-ID of the vulnerability, resulting in 1,822 vulnerabilities. We analyze users' patching behavior over 4 products: Google Chrome, Mozilla Firefox, Mozilla Thunderbird, and Adobe Flash Player, and only include hosts that have recorded more than 10 events for at least one application. This results in a data set consisting of 11,017,973 events over 426,031 unique hosts, 99.3% of which are between 01/2010 and 12/2012.

Although an open vulnerability indicates that the application could be exploited, few vulnerabilities are actually exploited in the wild. We collect exploit data from (1) public descriptions of Symantec's anti-virus signatures [27] and (2) metadata about exploit kits from Contagiodump [12]. Combining both sources of exploit information results in exploit release dates for 21 CVEs. The median time between vulnerability disclosure and an exploit kit targeting it is 17 days.

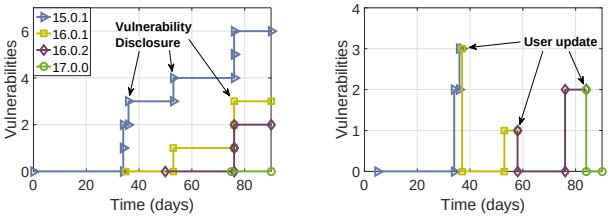
For Firefox, Flash Player, and Thunderbird, we manually scrape release history logs, either provided on the vendor's website, or collected by a third party, to find out when each version is released to the public. We have collected the results along with the source for each entry in a single document [3].

2.1 Curated Data

Host state Each update event corresponds to a *(machine ID, product, version, timestamp)* tuple, indicating the installation of a software on the host. However, the WINE database provides no information on when the product has been removed, or if the user has installed multiple product lines in parallel (e.g. Firefox 3.6, and 4.0). We utilize the following heuristic to update the state of a machine after each event. Assume that an event at time t signals the installation of version v belonging to product line ℓ, and we have detected the presence of versions $S_{t^-} = \{(\ell_1, v_1), \ldots, (\ell_n, v_n)\}$ on the machine prior to the event. For each ℓ_i in S_{t^-}, if there are no observations for the same line within 6 months of the current event, we remove the (ℓ_i, v_i) pair from S_{t^-}. We then add the (ℓ, v) pair, or update the corresponding pair in S_{t^-} if the same product line is already installed on the host, to obtain the state S_t after the event. We then take the union of vulnerabilities that affect each version in S_t from NVD, as the set of vulnerabilities present on the host. The subset of vulnerabilities that have already been disclosed, or exploited, represent the machine's security posture.

Version release date For Firefox, Flash Player, and Thunderbird, we can obtain the official release dates for each version by scraping version release notes from the vendor, or release histories collected by a third party. For Chrome, we tap into the patch measurement data to estimate release dates for each version. In previous work, Nappa et al. [17] identify the release date automatically, by selecting the first date when the version appears in WINE. However, we found

[3] We will release this list to the community upon paper publication.



(a) (b)

Fig. 1: The number of vulnerabilities in successive Firefox versions (left) and following a user's update events (right). Each color represent a single version.

that this approach can be unreliable in some cases. The binary that corresponds to a new version might appear in the wild half a year before it is made available on the release channel. We observe that on a release date there is usually a high volume of patching events. We thus first rank the dates by the count of patching events, and then identify the patch release date as the earliest day among the 10 dates with top ranks. We compared the results from this method with the release dates for Firefox and we found that they match for all the versions.

Purpose of updates To determine if users are influenced by the purpose of the updates, we identify four types of software releases: introducing new features I_{feats}, fixing bugs I_{bugs}, patching security vulnerabilities I_{vulns}, or introducing a new product line I_{majVer}. Using these four categories, we manually label the versions for Firefox and Flash Player. Since the release notes are not available for every build and they switched to silent updates on 2012-08-28 and 2012-03-28, respectively, the number of versions we labeled is 30 and 39, respectively.

User updates To study the frequency of irregular user behavior, we first obtain the number of events that result in the presence of more than one product line on a host. For Chrome, Flash Player, Firefox, and Thunderbird, 0.9%, 4.9%, 1.2% and 0.3% of events lead to the installation of more than one product line. For Flash Player, we further analyze the number of vulnerabilities associated with each product line. On average, in the presence of multiple product lines, 79.5% of vulnerabilities come from the lowest product version installed on the machine. Therefore, we take the lowest application version on the machine as its current state, and only consider a new event as a version upgrade if it updates the lowest installed version. Note that for evaluating whether a machine is prone to a known vulnerability, we will still use the complete set of installed versions.

Finally, for each state transition that updates the lowest installed version, we need to extract the user's delay in applying the update. We first take the timestamps for the current and previous events (denoted by T_u^k and T_u^{k-1}), and extract the first time an update was released for the previously installed version (denoted by T_r). The user's delay is then $S_u^k := T_u^k - \max(T_u^{k-1}, T_r)$. This means that we measure the users' delay from the day an update is available for the installed version, or the product installation date, whichever comes last;

Product	Users	> 0.05	> 0.01
Chrome	167592	87.8%	97.6%
Firefox	21174	74.6%	93.0%
Flash Player	7722	98.2%	99.9%
Thunderbird	1857	86.5%	97.5%

Table 2: Chi-squared test results over user update delays. We cannot reject the hypothesis that these sequences are drawn from a geometric distribution.

the latter takes effect when the user installs an outdated version. Note that successive versions do not necessarily follow a chronological order, as multiple product lines are often developed in parallel. For each release, we take the next version in the same line to be the update for that release. For end-of-life releases, we pick the first version in the subsequent line as the next logical update.

Figure 1a depicts a sample scenario for 4 successive releases of Firefox, released at times $t = 0, 35, 50, 75$ ($t = 0$ corresponds to "2012-09-11"). Firefox v15.0.1 is prone to 6 vulnerabilities, all of which are undisclosed at the time of release. However, these vulnerabilities are made public at times $t = 34, 36, 53, 76$, and patched in subsequent versions. Figure 1b illustrates a sample user in our data set who installs these versions at $t = 5, 37, 58, 84$, respectively. Note that with each update, the user inherits the set of vulnerabilities in the new release. An update is made available for the first version at time $t = 35$, and the user initiates a software update at time $T_u^1 = 37$, therefore the user's delay for the first update event is $S_u^1 = 2$ *days*. Similarly, $S_u^2 = 8$ *days*, and $S_u^3 = 9$ *days*.

3 Analysis of user behavior and its security implications

3.1 Modeling a user's patching delay

We assume that the user's update delays are drawn from a probability distribution specific to the $(user, product)$ pair. In previous work, the survival function for number of hosts without a security patch has been modeled as an exponential decay process [21, 23]. We note that a geometric distribution for a user's delay in applying a software update leads to the same model for the survival function. We independently test this assumption by performing a chi-squared goodness-of-fit test between each sequence and a geometric distribution whose parameter is calculated using a maximum likelihood estimate. The output of the test is a p-value: small p-values reject the null hypothesis "The samples are drawn from a geometric distribution". Table 2 summarizes our results, for each product we have included the number of users tested, and the percentage with p-values higher than significance levels of 5% and 1%. For the test, we ignore users with fewer than 20 update events. Our results show that for the majority of users the geometric distribution is a good fit.

The above results suggest that the users' response to new product releases are fairly "simple-minded", in the sense that they can be well-modeled using a one-

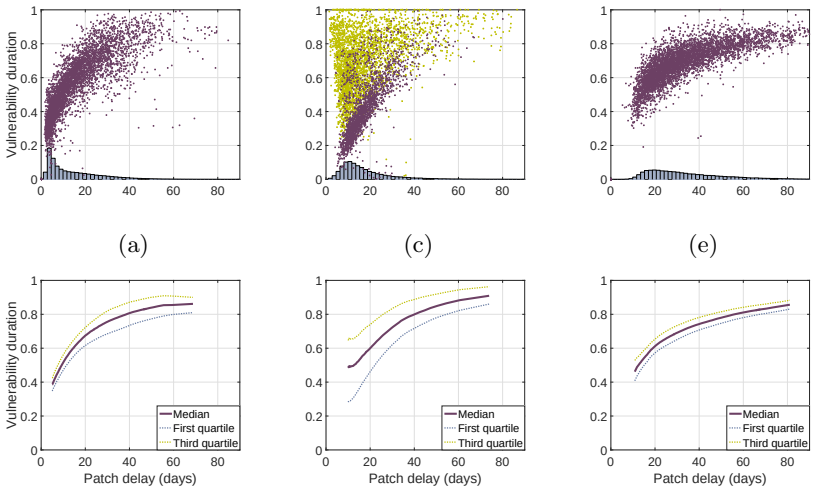


Fig. 2: Scatter plots of normalized vulnerability duration vs. average user delay in days (top), and the mean, and first and third quartiles for different user types (bottom). Each point in the scatter plots corresponds to a single user. In 2c the yellow/red dots are users active in 2010/only active starting 2011.

parameter distribution. In what follows, we examine the relationship between patch delays and vulnerability states. Note that due to this single-parameter characterization, the average patching delay is sufficient for summarizing user behavior, and we shall only rely on sorting users by this measure.

3.2 Vulnerability state

We take the fraction of time that a host remains susceptible to at least one known vulnerability as an indicator of its security posture or *vulnerability state*. Figures 2a, 2c, and 2e display scatter plots of this measure for Chrome, Firefox, and Flash Player, respectively. For each figure we have randomly selected 5000 users, where each point represents one user. A point's x and y coordinates correspond to the average patch delay of that host, and its measured vulnerability state. The histogram at the bottom of each plot shows the distribution of users with respect to their average patch time; generated for users with an observation interval of at least one year, resulting in 140,588 sample points for Chrome, 64,016 for Firefox, 55,042 for Flash Player. Note that the majority of hosts are observed for intervals smaller than 3 years, and we have omitted hosts with less than 10 update events (see Section 2). Therefore, our study does not capture users with average patch delays greater than roughly 100 days; longer observation windows are required to accurately assess the behavior of such users.

Vulnerability state as a function of average patch delay We further group users with similar behavior by sorting them according to their estimated patch delay, and create bins consisting of 500 users. We calculate the median vulnerability duration, and the first and third quartiles in each bin; the results are illustrated in Figures 2b, 2d, and 2f. We observe that a user with equal delays in each product experiences similar vulnerability duration. At 20 days, the user will remain vulnerable for 60% of the time, at 40 days this increases to 80%.

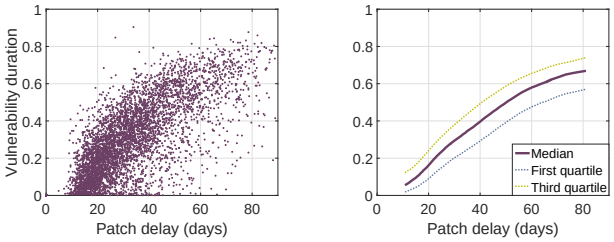
Across the three products, Chrome users clearly are more likely to have a lower patch delay (as shown in the histograms), likely the effect of silent update, whereas Flash users are the most tardy. However, given the same average delay, the amount of vulnerabilities a user faces is very consistent across all products.

Outliers In Figure 2c we see high variability in vulnerability durations for users with similar patch times. Upon further inspection, we discovered two vulnerabilities for Firefox, `CVE-2010-0654` and `CVE-2010-1585`, that were published on "2010-02-18" and "2010-04-28", but first patched on "2010-07-20" and "2011-03-01", respectively. As a result, users that have been observed during 2010, have remained vulnerable for most of that year, regardless of their behavior. In Figure 2c, we have used a lighter color to display hosts that have been observed at any time during 2010. The rest of the hosts (those that have opted in after 2010), exhibit similar variability to Chrome and Flash Player users.

Comparison across products We further calculated the average vulnerability duration, over all users of each application. Note that these values are affected by the following properties of each product: (i) the distribution of different user types (the histograms in Figure 2), and (ii) the expected vulnerability duration for each user type (Figures 2b, 2d, and 2f). For Chrome, Firefox, Flash Player, and Thunderbird, the average host was susceptible to at least one vulnerability for 53.5%, 59.9%, 68.7%, and 55.7% of days. It follows that the improvement provided by different updating mechanisms in these applications is marginal.

A host's vulnerability state is influenced by two conditions. First, for a single vulnerability, the patch should be applied before the vulnerability is publicly disclosed. Nevertheless, even if the user misses the disclosure date, the damage can be minimized by prompt patching. However, when taking into account successive vulnerabilities, if the user does not apply the patch before the next vulnerability is disclosed, the clock is reset, and now they will have to apply a new patch to secure their machine. Quantitatively, for Chrome and Firefox, our data set includes 124, and 114 vulnerability disclosures between 2010 and 2012, resulting in an average of approximately 10 days between successive disclosures. However, our estimated results show that the average patch times for users of Chrome and Firefox, is 9.9 and 15.6 days, respectively, meaning that users often cannot patch a vulnerability before the next one is discovered. For Chrome, adopting silent updates does not seem to provide the necessary margin to see any significant effect on the vulnerability duration of hosts.

Breakdown of the vulnerability window Note that the vulnerability of a machine can be caused due to the vendor's failure to release a patch before a vulnerability is disclosed, or the user's negligence in installing the patch. We found



(a) (b)

Fig. 3: Scatter plot (left) and mean, and first and third quartiles (right) for exploited vulnerabilities of Flash Player.

that, summed over all users, for Chrome, Flash Player, Firefox, and Thunderbird, 59.3%, 61.6%, 47.9% and 55.7% of days where a machine was susceptible to a known vulnerability was caused by user negligence.

3.3 Susceptibility to vulnerability exploits

Being prone to known vulnerabilities does not necessarily translate into an imminent threat, as the machine can only be breached through a real exploit. We perform a similar study on the percentage of days that a host remains susceptible to an exploitable vulnerability. Figures 3a and 3b display the scatter plot and vulnerability trends for 15 exploits of Flash Player. We did not have a sufficient amount of exploits for Chrome and Firefox, we were only able to find one known exploit for Chrome, and 2 for Firefox. Comparing these plots to 2a-2f, we observe the same correlation between average patch times and vulnerability states. However, for similar patching delays, we generally see lower risk for known exploits. This is due to the small number of exploited vulnerabilities for Flash. Nevertheless, we observe that many hosts are susceptible to exploits more than 50% of the time, highlighting the threat exploit kits present to end-hosts.

3.4 Factors that impact user behavior

Version-specific factors Suppose we take an interval of size $2T$ and look at the total number of updating events that occur within the first and second half of the interval, across all users and for a specific subset of releases of a given product, and denote these by N_1 and N_2, respectively. We can then define a population-wide *willingness to update* as $W = 1 - (N_2/N_1)^{1/T}$. When looking at a period immediately following a release, if more users consider a new patch important and apply the patch promptly, then W tends to be high. Conversely, if more users hesitate to install the new patch, then more patching events occur at the second half of observation period resulting in a lower W.

For Firefox and Flash Player, we extract four features I_{vulns}, I_{bugs}, I_{feats}, and I_{majVer}, as described in Section 2. For each feature, we split the versions

Aplication	I_{vulns}	I_{bugs}	I_{feats}	I_{majVer}
Flash Player	0.860	0.416	0.736	0.419
Firefox	0.109	0.226	0.126	0.027

Table 3: p-values from T-test on version-specific factors.

Country	Chrome	Firefox	Flash	Thunderbird
All	9.9	15.6	29.7	15.2
AU	10.6	16.3	30.1	15.1
CA	10.4	15.6	30.7	14.6
DE	10.9	15.3	24.9	14.7
FR	10.4	16.2	28.8	14.4
IT	8.8	15.9	26.1	13.5
JP	13.0	14.2	26.6	16.3
NL	10.4	15.2	28.5	14.7
PL	8.2	13.8	26.9	14.2
UK	9.2	15.7	28.3	13.9
US	10.5	15.5	32.1	15.4

Table 4: Average patch times by country.

into two groups: those that have the feature and those that do not, and measure W within each group. We then conduct a statistical hypothesis test to determine if the updates from one group are deployed faster than the ones from the other group. Specifically, we perform a T-test between two groups with the null hypothesis "There is no difference in the mean of W". Low p-values from the T-test indicate that the factor affects the user's behavior. Here, we choose the significance level $\alpha = 0.05$. As we perform multiple hypothesis tests on the same data set, increasing the likelihood that we will find a significant result by pure chance, we apply the Bonferroni correction [1], by dividing α by the number of hypotheses tested. The adjusted α is 0.0125. The results are shown in Table 3. I_{majVer} for Firefox is the only factor with p-value below 0.05. However, this is not statistically significant after applying Bonferroni correction. This indicates that changes in versions have no statistically significant effect on user behavior.

Note that our results do not necessarily reflect users' indifference to these categorizations. In a recent user study [15], 80% of users state that update categories can influence their decision on applying a software update, and two-thirds react positively to a prototype that tags each update with one of five categories. Our results indicate that for the examined products, this information may not be readily available (we had to manually tag the release notes ourselves), which in turn causes users to behave independently of the update's intent.

Different countries and patch delivery Table 4 shows the average time to patch for the top 10 countries with the largest numbers of users. Note that Chrome uses silent updates, and therefore has the lowest patch times. Firefox and Thunderbird versions prior to 15.0, and 16.0 (released 2012-08-28, and 2012-10-09, respectively) download updates in the background and prompt users for installation. Flash Player versions prior to 11.2 (released 2012-03-28) prompt users to download and install updates, and consequently exhibit the longest patch times. All three products switch to silent updates after the indicated dates; however these changes do not apply to the majority of our samples. For all four products, patching behavior is remarkably consistent, suggesting that cultural differences among these countries do not play a significant role in user behavior.

4 Related work

Rescorla [23] studied a 2002 OpenSSL vulnerability and observed two waves of patching: one in response to the vulnerability disclosure and one after the release of the Slapper worm exploit. Ramos [21] analyzed several remotely-exploitable vulnerabilities and reported a slow decay rate in some cases and some vulnerabilities that did not decay at all. Yilek et al. [29] scanned OpenSSL servers affected by a 2008 key generation vulnerability in Debian Linux and found a high patch rate in the first 30 days, followed by patching waves for the next six months. Durumeric et al. [11] showed that more than 50% of servers affected by the recent Heartbleed vulnerability in OpenSSL remained vulnerable after three months. Zhang et al. [30] showed that, even after patching OpenSSL, most websites remained vulnerable because they had not revoked certificates that may have been compromised owing to Heartbleed. The rate of updating is considerably higher for systems that employ automated updates [9, 13]. Gkantsidis et al. [13] concluded that 80% of Windows Update users receive patches within 24 hours after their release. Dübendorfer et al. [9] suggested that Google Chrome's silent update mechanism is able to update 97% of active browser instances within 21 days. Nappa et al. [17], measured vulnerability decay in 10 client-side applications and identified security threats presented by multiple installations of the same program and by shared libraries distributed with several applications. Alhazmi et al. [2] examined five different vulnerability discovery models, fitting the models using data from three operating systems.

On factors that may affect vulnerability patching and user behavior, Schneider et al. [16] proposed several hypotheses, including an under-appreciation of risks and a fear of destabilizing other software. Vaniea et al. [28] suggested that negative experiences with past updates affect the users' willingness to deploy patches. Mathur et al. [15] study 30 users' updating practices, and design and evaluate a prototype updating interface based on their feedback.

5 Conclusions

In this paper we have conducted an in-depth analysis of the dynamics between vendors and consumers when it comes to software security. To the best of our knowledge, this is the first study on how individual behavior can influence the security state of a user's machine over long periods, where the continuous discovery of vulnerabilities, patch deployment by vendors, and the installation of patches create windows of opportunities for malicious entities to exploit open vulnerabilities on the machine. We have shown that frequent updating, and steps taken by vendors to speed up the installation of patches, provide marginal benefits when the rate at which new vulnerabilities are introduced into the product's code is high. Consequently, developers' should exercise due diligence when shipping new products to end-users, as the detrimental effects of releasing vulnerable applications to the public often cannot be eliminated by prompt patch deployment.

Our results also represent a first step toward understanding the *deployment-specific barriers* for updating software. We observe that user behavior can be

modeled well using a simple and elegant mathematical model. We do not observe clusters of users with respect to the patching delay or the vulnerability state. Moreover, users do not make patching decisions depending on the type of improvements introduced with each new release (possibly due to how this information is presented), and the willingness to patch does not vary significantly across different countries. However, users seem to exhibit different behavior for different products, suggesting that vendors may be able to influence the users' patching delays. For example, Figure 2 suggests that the vulnerability duration for Flash Player exhibits a lower variability than for Chrome and Firefox, despite the lack of a silent updating mechanism. This consistency may result from the fact that users are compelled to upgrade when sites remove backward compatibility for older Flash versions. A deeper understanding of these barriers could enable improvements in the software updating process.

Although we have shown that users' behavior can effectively be explained using a simple model, we are not able to build similar profiles for vendors. This is partly due to lack of a large data set on software vulnerability cycles. The set of unique vulnerability disclosures and patch deployments concerning the products under examination was too small to carry out a comprehensive study on product behavior. Such an analysis could close the loop when assessing the security posture of an end-user, by predicting the host's vulnerability state across different products, or for new products entering the market. Finally, leveraging additional data sources that can reveal the whole extent of user behavior, such as extending the study to other operating systems, and measuring periods of time where the system or a specific application are not used (this would lead to an overestimation of the vulnerability window in our current analysis) are other possible directions for future work.

References

1. Abdi, H.: Bonferroni and Šidák corrections for multiple comparisons. Sage (2007)
2. Alhazmi, O.H., Malaiya, Y.K.: Modeling the vulnerability discovery process. In: International Symposium on Software Reliability Engineering (2005)
3. Alhazmi, O.H., Malaiya, Y.K., Ray, I.: Measuring, analyzing and predicting security vulnerabilities in software systems. Computers & Security 26(3), 219–228 (2007)
4. Arbaugh, W.A., Fithen, W.L., McHugh, J.: Windows of vulnerability: A case study analysis. IEEE Computer 33(12), 52–59 (2000)
5. Arora, A., Krishnan, R., Nandkumar, A., Telang, R., Yang, Y.: Impact of vulnerability disclosure and patch availability - An empirical analysis. In: Workshop on the Economics of Information Security (2004)
6. Bilge, L., Dumitraş, T.: Before we knew it: An empirical study of zero-day attacks in the real world. In: ACM Conference on Computer and Communications Security (2012)
7. Cavusoglu, H.C.H., Raghunathan, S.: Emerging issues in responsible vulnerability disclosure. In: Workshop on Information Technology and Systems (2004)
8. Clark, S., Collis, M., Blaze, M., Smith, J.M.: Moving targets: Security and rapid-release in Firefox. In: ACM SIGSAC Conference on Computer and Communications Security (2014)

9. Dübendorfer, T., Frei, S.: Web browser security update effectiveness. In: International Workshop on Critical Information Infrastructures Security (2009)
10. Dumitraș, T., Shou, D.: Toward a standard benchmark for computer security research: The Worldwide Intelligence Network Environment (WINE). In: Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (2011)
11. Durumeric, Z., Kasten, J., Adrian, D., Halderman, J.A., Bailey, M., et al.: The matter of Heartbleed. In: Internet Measurement Conference (2014)
12. Exploit kits. `http://contagiodump.blogspot.com`
13. Gkantsidis, C., Karagiannis, T., Rodriguez, P., Vojnovic, M.: Planet scale software updates. In: ACM SIGCOMM Computer Communication Review (2006)
14. Grier, C., Ballard, L., Caballero, J., Chachra, N., Dietrich, C.J., et al.: Manufacturing compromise: The emergence of exploit-as-a-service. In: ACM Conference on Computer and Communications Security (2012)
15. Mathur, A., Engel, J., Sobti, S., Chang, V., Chetty, M.: "They keep coming back like zombies": Improving software updating interfaces. In: Symposium on Usable Privacy and Security (2016)
16. Mulligan, D.K., Schneider, F.B.: Doctrine for cybersecurity. Daedalus, Journal of the American Academy of Arts and Sciences 140(4), 70–92 (2011)
17. Nappa, A., Johnson, R., Bilge, L., Caballero, J., Dumitraș, T.: The attack of the clones: A study of the impact of shared code on vulnerability patching. In: IEEE Symposium on Security and Privacy (2015)
18. Neuhaus, S., Zimmermann, T., Holler, C., Zeller, A.: Predicting vulnerable software components. In: ACM Conference on Computer and Communications Security (2007)
19. NIST: National Vulnerability Database. `https://nvd.nist.gov`
20. Ozment, A., Schechter, S.E.: Milk or wine: Does software security improve with age? In: USENIX Security Symposium (2006)
21. Ramos, T.: The laws of vulnerabilities. In: RSA Conference (2006)
22. Rescorla, E.: Is finding security holes a good idea? In: IEEE Security and Privacy (2005)
23. Rescorla, E.: Security holes... who cares. In: USENIX Security Symposium (2003)
24. Sabottke, C., Suciu, O., Dumitraș, T.: Vulnerability disclosure in the age of social media: Exploiting Twitter for predicting real-world exploits. In: USENIX Security Symposium (2015)
25. Shahzad, M., Shafiq, M.Z., Liu, A.X.: A large scale exploratory analysis of software vulnerability life cycles. In: International Conference on Software Engineering (2012)
26. Shankland, S.: Heartbleed bug undoes web encryption, reveals Yahoo passwords. `http://www.cnet.com/news/heartbleed-bug-undoes-web-encryption-reveals-user-passwords` (2014)
27. Symantec Corporation: Symantec threat explorer. `http://www.symantec.com/security_response/threatexplorer/azlisting.jsp` (2012)
28. Vaniea, K., Rader, E.J., Wash, R.: Betrayed by updates: How negative experiences affect future security. In: ACM Conference on Human Factors in Computing (2014)
29. Yilek, S., Rescorla, E., Shacham, H., Enright, B., Savage, S.: When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In: Internet Measurement Conference (2009)
30. Zhang, L., Choffnes, D., Dumitraș, T., Levin, D., Mislove, A., et al.: Analysis of SSL certificate reissues and revocations in the wake of Heartbleed. In: Internet Measurement Conference (2014)