Proceedings

LASER 2013

Learning from Authoritative Security Experiment Results

LASER 2013

Proceedings

LASER 2013
Learning from Authoritative Security Experiment Results

Arlington, Virginia, USA
16–17 October 2013

ISBN 978-1-931971-06-5

Editorial production by Christoph Schuba

Table of Contents

Technical Program

Organizing Committee

Laura Tinnel, SRI International (General Chair)
Greg Shannon, Carnegie Mellon University/CERT, (PC Co-Chair)
Tadayoshi Kohno, University of Washington, (PC Co-Chair)
Christoph Schuba, Oracle Corp. (Proceedings)
Carrie Gates, CA Technologies, (Sponsorships)
David Balenson, SRI International (Treasurer and Local Arrangements)
Edward Talbot, Consultant (Publicity & Scholarships)

Program Committee

Greg Shannon, Carnegie Mellon University/CERT, (PC Co-Chair)
Tadayoshi Kohno, University of Washington, (PC Co-Chair)
David Balenson, SRI International
Matt Bishop, University of California, Davis
Joseph Bonneau, Google Inc.
Pete Dinsmore, Johns Hopkins University/APL
Debin Gao, Singapore Management University
Carrie Gates, CA Technologies
Alefiya Hussain, University of Southern California/ISI
Carl Landwehr, George Washington University
Sean Peisert, University of California, Davis and LBNL
Angela Sasse, University College London
Christoph Schuba, Oracle Corp.
Ed Talbot, Consultant
Laura S. Tinnel, SRI International
Kami Vaniea, Michigan State University
Charles Wright, Portland State University

Contributing Sponsors

SRI International

Is It the Typeset or the Type of Statistics? Disfluent Font and Self-Disclosure

Rebecca Balebako, Eyal Pe'er, Laura Brandimarte, Lorrie Faith Cranor, Alessandro Acquisti
Carnegie Mellon University

Abstract

- **Background.** The security and privacy communities have become increasingly interested in results from behavioral economics and psychology to help frame decisions so that users can make better privacy and security choices. One such result in the literature suggests that cognitive disfluency (presenting questions in a hard-to-read font) reduces self-disclosure.
- **Aim.** To examine the replicability and reliability of the effect of disfluency on self-disclosure, in order to test whether such approaches might be used to promote safer security and privacy behaviors.
- **Method.** We conducted a series of survey studies on human subjects with two conditions - disfluent and fluent font. The surveys were completed online (390 participants throughout the United States), on tablets (93 students) and with pen and paper (three studies with 89, 61, and 59 students). The pen and paper studies replicated the original study exactly. We ran an independent samples t-test to check for significant differences between the averages of desirable responses across the two conditions.
- **Results.** In all but one case, participants did not show lower self-disclosure rates under disfluent conditions using an independent samples t-test. We re-analyzed the original data and our data using the same statistical test (paired t-test) as used in the original paper, and only the data from the original published studies supported the hypothesis.
- **Conclusions.**We argue that the effect of disfluency on disclosure originally reported in the literature might result from the choice of statistical analysis, and that disfluency does not reliably or consistently affect self-disclosure. Thus, disfluency may not be relied on for interface designers trying to improve security or privacy decision making.

1 Introduction

Humans play a crucial role in security and privacy decisions. Technically secure systems can be vulnerable to attacks when users make suboptimal choices, such as choosing or re-using easy-to-guess passwords or falling prey to phishing or spear-phishing attacks [19, 7]. Similarly, privacy decisions are often left to the sole responsibility of the user, who is —at times quite unrealistically— expected to make optimal choices regarding what information to reveal or share based on her preferences and the trade-offs involved in information disclosure [1]. Realizing the complexity of such a task, security and privacy researchers have been trying to find ways to assist individuals' decision-making by designing interventions or interfaces informed by fundamental ideas from fields such as psychology and behavioral decision research.

One such idea is that of *cognitive fluency.* Disfluent conditions are those that induce "metacognitive difficulty" [4]. For instance, disfluency is induced during text processing tasks by using a hard-to-read font or a second language. Based on the literature on cognitive fluency, people perceive a task as more difficult when it is disfluent than when it is fluent, and process it analytically, as opposed to relying on quick judgments and heuristics [4, 17]. It has been argued that people make better decisions and learn better in disfluent conditions than in fluent ones [9, 16].

A potential implication of these findings could be that users could be assisted in making better security and privacy decisions simply by changing the font used to provide information relevant to that decision. The argument is that users would slow down to process the font, and in doing so, would slow down to truly contemplate their decision. This would lead to fewer errors in judgment when making a security decision (such as whether to open an attachment) or a privacy choice (such as whether to reveal sensitive information).

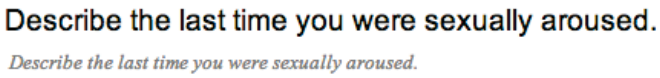

Figure 1: Examples of question in fluent font (top) and disfluent font (bottom)

2 Problem Being Solved

In a series of experiments, we investigated the role of disfluency in the decision to disclose sensitive information. While this is a privacy decision, it should be noted that the decision to reveal privacy-sensitive information could also lead to security issues. For example, when a user reveals information to challenge questions in authentication solutions, or when a businessperson reveals corporate information that could lead to a data breach.

Our intention was to replicate, and then build off of a seminal 2009 paper by Alter & Oppenheimer reporting that fluent conditions encouraged disclosure, while disfluent conditions discouraged it [3]. In particular, the manuscript found that disfluency reduces disclosure when it is measured using the Social Desirability Scale (SDS) [8]: participants admitted to fewer undesirable behaviors when the questions were harder to read. Our attempts to replicate this work showed that disfluency did not consistently impact self-disclosure. We were not able to find an explanatory variable or a method of testing that could show that disfluency reduces disclosure when compared to a fluent condition. Instead, we found that one unusual choice of statistical tests may explain the original effect reported by the authors. We conclude that disfluency may not be relied upon to influence users to disclose less information, and that this holds true for different measures of disclosure and different formats for the surveys.

3 Background and related work

A large body of research has explored how people make disclosure decisions. One of the rising themes from the literature is that people's personal preferences for self-disclosure are not always consistent, and that various factors affect people's decisions to disclose personal information [14]. People seem to rely on contextual cues, such as a survey's look and feel or implicit social norms, when disclosing intimate details about themselves [11]. A recent study has shown that when people perceive higher control over who can access and use their online personal information, they become more willing to disclose it even if that implies higher objective risks of privacy intrusions or security breaches, as compared to a condition where they perceive less control but are actually exposed to lower objective risks from their disclosure [6]. Also, people respond more honestly and with higher rates of disclosure to an online, versus a paper-and-pencil version, of the same questionnaire [21], and are more inclined to divulge information online than when they communicate face-to-face [10, 22].

Following this line of research, a recent manuscript [3] showed how cognitive disfluency (presenting questions in a hard-to-read font) suppressed people's propensity to disclose personal and sensitive information about themselves. When questions were printed in a disfluent font (50% gray, italicized 10-point Times New Roman font), participants exhibited lower rates of self-disclosure, compared to a condition where the questions were printed in a clear font (black, 12-point Times New Roman font). In a first study (Study 1a), 33 undergraduate participants showed higher percentages of socially desirable responses (indicating lower self-disclosure) when the 33 items in the Crowne-Marlow Social Desirability Scale (SDS) [8] were printed in disfluent vs. clear fonts. The second study (Study 1b) replicated these results using the 10-item version of the SDS. The third study (Study 2) showed that disfluency increased participants' tendency to generate thoughts associated with risks. The fourth study (Study 3) showed that the effect of disfluency on self-disclosure was mediated by negative emotions. The last study (Study 4) was a field experiment that showed that when an online disclosure web site (Grouphug) changed its design, making it easier to read, self-disclosure went up. Our efforts to replicate focused on Study 1a and Study 1b, which used the SDS measure.

4 Approach

The purpose of all our studies was to test the following hypothesis:

H: Disfluency reduces self-disclosure

We report several failed attempts to replicate one of the findings reported in [3] - namely, the results associated with Study 1a and Study 1b in the original paper. We focused on these two studies because they a) demonstrated the (alleged) effect of disfluency and disclosure and b) did so without any additional mediators (as Studies 2 and 3 did) and c) were replicable (Study 4 could not have been, for all practical purposed replicated reliably). Our original objective was actually to validate, using the disfluency manipulation, several self-disclosure

measures, which we planned to use for a different research project. We predicted that, as disfluency affected SDS scores, it should also affect other measures of self-disclosure.

5 Method

To test hypothesis **H**, we ran survey studies on human subjects. Our study was approved by Carnegie Mellon's Institutional Review Board. We designed our study so that the only differences between the two conditions was whether a disfluent or fluent font was used. Participants were assigned randomly to each condition. There were no significant differences between conditions in each study for age or gender. All information about the study, including consent forms, information about the researchers, and instructions, were identical in the different conditions for each study.

After the initial failure to replicate the results presented in one of the studies in [3], we attempted several variations of the study. The original authors of the manuscript were extremely prompt and helpful, and provided us with the original experimental material employed in their experiments. We ran our surveys online, in person using tablets, and in person using pen and paper. We describe the materials used and the subjects tested for each survey below.

The first study was completed online using all four measures of self-disclosure, as described below. To explore whether the fact that the study was conducted online could explain the null results of Study 1, we conducted several follow-up studies (Study 2, 3, and 4) focusing specifically on the SDS measure used in [3]. The second study was done in-person using tablets. The third and forth studies were done using pen-and-paper surveys. Specifically, the third study was done using material that we independently developed, while the forth study used the original research materials used by Alter and Oppenheimer. The results of our studies are shown in Table 5.2. Finally, a follow-up study was also online and was a replication of one measure in the first study—namely, the SDS.

The survey questions are included in Appendix A, and samples of the pen and paper versions are provided in Appendix B. All surveys were anonymous; users were not asked to provide any identifying information.

Across the studies, participants were asked to complete several measures of self-disclosure. All of these measures were either shown in a clear, regular, font (Times new-roman, 12 points, black) or a disfluent font (Times new-roman, 10 points, grey). Examples of each are shown in Figure 1).

5.1 Manipulation Check

Based on previous findings [3], we predicted that the disfluent font would make the survey harder to read and thus reduce participants' rates of self-disclosure. In the first online study, participants in the disfluent condition indeed rated the survey as less easy to read than those in the clear font condition ($M_{disfluent}$=5.38, SD=1.78, vs. M_{clear}=6.38, SD=.91, t (383) = 6.96, p ¡ .01 where 1="not at all" and 7="very much" easy to read).

5.2 Measures of Self-Disclosure

We used four measures of self-disclosure in our first online study. We describe these measures below. In the pen-and-paper studies and tablet studies, we focused solely on SDS, in an effort to replicate the original study presented in [3].

5.2.1 Social Desirability Scale

The Social Desirability Scale, developed by Crowne and Marlowe in 1960 [8], includes questions about socially desirable behaviors, as well as undesirable behaviors. Many of the socially desirable behaviors reflect an improbable but idealized behavior, while the undesirable behaviors may be more realistic. The original version of the scale consisted of 33 yes or no questions. This scale has been used both as a measure of how respondents self-report their own behavior (along with their need for approval) and as a measure of their actual behavior [12]. Researchers have developed short forms of the SDS scale and tested their reliability and homogeneity [18, 20, 13]. We also found high reliability for this scale in our study (Cronbach's alpha = .85 in Study 1). Participants' score on the SDS is calculated by summing up the number of undesirable responses (i.e., responding 'yes' to a socially undesirable behavior, such as "I like to gossip" or responding 'no' to a socially desirable behavior, such as "I'm always a good listener"). This sum is then divided by the number of items, resulting in a proportion of undesirable responses. This score could be considered a proxy of self-disclosure, as the more people are willing to admit to undesirable responses, the higher their self-disclosure is. Thus, a higher score on the SDS means higher self-disclosure. We believe SDS may be an imperfect measure of self-disclosure, as it may conflate actual conformance to social norms with disclosure. Therefore, we specifically used three additional measures to examine disfluency and self-disclosure.

5.2.2 Unethical Behaviors

We used 14 questions about unethical behaviors adapted from [11]. The questions had been tested for privacy sen-

Study	Measure	Disfluent		Control		Independent t-test		Paired t-test	
		Mean	SD	Mean	SD	t(df)	p	t(df)	p
A&O 1a	SDS 33-items [a]	40.92%	13.73	34.94%	12.72	1.26 (31)	0.22	2.26 (32)	0.03
A&O 1b	SDS 10-items [a]	42.11%	16.53	34.71%	18.75	1.26 (34)	0.22	2.6 (9)	0.03
1: Online	SDS 33-items [b]	45.97%	19.62	45.75%	17.68	-.12 (388)	0.91	.24 (32)	0.81
	Unethical behaviors [c]	29.82%	15.86	31.89%	16.47	1.27 (388)	0.21	1.88 (14)	0.08
	$15 gift card [d]	3.23	5.73	3.37	5.65	.25 (388)	0.81	-	-
	$10 gift card [d]	2.27	3.65	2.19	4	-.22 (388)	0.83	-	-
	Sensitive questions (1)[e]	1.22	0.45	1.30	0.54	2.09 (388)	0.04	1.6 (4)	0.19
	Sensitive questions (2)[e f]	1.36	0.47	1.17	0.46	-2.16 (112)	0.03	-2.3 (4)	0.08
2: Tablets	SDS 33-items [b]	43.62%	13.60	41.35%	13.5	-.81 (91)	0.42	2.17 (32)	0.04
3: P&P1	SDS 33-items [b]	46.46%	16.70	48.28%	16.5	.49 (78)	0.63	-.89 (32)	0.38
4: P&P2	SDS 33-items [b]	43.81%	16.02	42.61%	13.3	.31 (57)	0.76	.5 (32)	0.62
	SDS 10-items [b]	25.16%	12.10	25.67%	11.7	-.17 (59)	0.87	.71 (9)	0.49

[a] Results from Alter & Oppenheimer (2009) [3]

[b] Percentage of responses agreeing (or disagreeing) with socially desirable (or undesirable) behaviors. Higher scores indicate lower self-disclosure.

[c] Percentage of responses admitting to unethical behaviors. Higher scores indicate higher self-disclosure.

[d] Mean difference (in dollars) participants were willing to pay for an anonymous vs. identified gift card of the same value ($10 or $15). A positive difference indicates that the participant was willing to pay more for the anonymous card, and, larger (positive) differences indicate less self-disclosure.

[e] Two independent judges rated the depth of disclosure in participants' responses to the open-ended questions on a scale between 0 (non-responses) to 4 (highly revealing responses). The scores presented are averages of the five sensitive questions. Higher scores indicate higher self-disclosure.

[f] Results of the follow-up study to Study 1.

Table 1: Comparing disfluent and control conditions for all studies and measures

sitivity. These questions included financial ("Have you had credit card debt above $100"), illegal ("Have you tried LSD, ecstasy or similar drugs"), and sexual themes ("Have you masturbated at work, school, or in a public restroom"). Participants were given the options to respond with "yes," "no," or "Prefer not to answer." We used affirmative answers ("yes") as a measure of willingness to disclose and averaged the rate of self-disclosure across the 14 questions. Notice that these answers rely on self-reporting, and thus do not attempt to measure participants' actual level of unethical behaviors, but only their willingness to disclose such behaviors. Assuming that a participants who either committed or did not commit a certain unethical behavior is equally likely to be randomly assigned to the control or disfluent condition, different scores between the conditions would suggest different levels of self-disclosure.

5.2.3 Sensitive Questions

The next dependent variable we considered consisted of five open-ended questions that pertain to sensitive issues [15]. For instance, this set included questions such as: "What is your most common sexual fantasy?" Participants were asked to respond using several sentences. Following Moon [15], two independent judges rated the depth of disclosure in participants' responses to the open-ended questions on a scale from 0 (non-responses) to 4 (highly revealing responses) for amount and detail of disclosure. Inter-rater reliability ranged from .86 to .91 for all questions. Thus, the scores for all five questions were averaged into one score for disclosure depth on the sensitive questions in overall.

5.2.4 Gift Cards

Previous literature has used willingness to keep or exchange gift cards of different value and with different privacy protection features as a measure of informational privacy concerns [2]. Using a similar approach, we asked four pairs of questions (8 total) about willingness to buy gift cards that were either described as anonymous (not requiring any personal information to use) or identified (requiring email, name and address to use). We calculated, for each participant, the difference in their willingness to pay for an identified vs. anonymous gift card of a certain value ($10 and $15). A positive difference would indicate that the participant was willing to pay more for the anonymous card, and larger (positive) differences would suggest a higher preference toward the anonymous card (and, thus, less disclosure).

6 Analysis

As most research that compares the effectiveness of a treatment on a reliable scale between a treatment and control conditions does, we calculated the percentage of desirable responses per participant, and ran an independent samples t-test to check for significant differences between the averages of desirable responses across the two conditions (these are reported in 5.2). Such a test is typically used to determine whether the difference in the mean between two unrelated groups is significant (between-subjects design), under the assumption that the samples are drawn from normally distributed populations with equal variances (but adjustments for unequal variances are trivial with modern statistical software). The significance of the results is highly reliant on the sample size.

7 Results

In this section we describe the data and the results from the four studies we completed. For each study, we describe the participant selection criteria, the number of participants, and the gender balance of participants. We also describe the questionnaires and the results for each study.

7.1 Study 1: Online

Study 1 consisted in an online survey that included 390 participants from Amazon Mechanical Turk [1] (64.6% were female; mean age was 35.34, SD=13). Berinsky found that studies using this platform often had samples more representative of the United States population than most in-person studies. Also, they were able to replicate several studies using this platform [5] Therefore, we are reassured that our results can be generalized to the US population and their online disclosure decisions. We included several attention check questions (e.g. "Have you ever had a fatal heart attack while watching television?"), which we used to eliminate participants who gave the wrong answer.

The four measures (SDS, unethical behaviors, sensitive questions, and willingness to buy gift cards) were presented to participants in random order, and so were the questions composing each measure. The surveys were anonymous, and participants were paid 80 cents. About half of the participants received all of these questions in a clear font, while the others received them in a disfluent font. For each of these measures, we computed an overall average score for each of the conditions and compared these using an independent samples t-test. As

[1] http://aws.amazon.com/mturk/

can be seen in 5.2, only the open-ended sensitive questions showed a statistically significant difference in the expected direction, while for all other measures no significant differences were found between the disfluent and clear font conditions.

In a follow-up study, conducted six months later, we tried to replicate the effect of disfluency on these sensitive questions. In this replication effort, we recruited 114 additional participants from Amazon Mechanical Turk and used only the sensitive question measure (as opposed to all four measures in Study 1). We found statistically significant results in the *opposite* direction: disfluency increased disclosure, as can be seen in 5.2.

7.2 Study 2: Tablets

We hypothesized that the null results from Study 1 were due to the fact that the study was conducted online. Therefore, we decided to run a study in person using tablets. Furthermore, in an effort to replicate the original work, we used a similar population: undergraduate students on a college campus. We asked 93 students (58.1% females, average age 19.5, SD=1.7) to complete the 33-items SDS using 8-inch tablets. The tablets were configured in a way that a participant could not change the zoom of the screen (which was, theoretically, possible in the previous study, and may have potentially tampered with our manipulation of disfluency). In these studies, run at the university campus center, a research assistant asked students to take the survey immediately using the provided tablet. Participants received a candy bar for their time. We found, as shown as the second study in 5.2, no significant difference between the disfluent and control conditions.

7.3 Study 3: Pen and Paper 1

Study 2 was done electronically, not using pen and paper, as in the original study. We thus decided to run our third study using paper-and-pencil questionnaires (as in [3]). We asked 80 undergraduate students (58.8% female, average age 19.6, SD = 1.7) to complete the 33-items SDS questionnaire either in the clear font or disfluent font conditions. Once again, we ran the study at the university campus center. Students were recruited by a research assistant, took the paper survey immediately, and were given a candy bar for their participation. Again, we found no statistically significant difference between the conditions (see 5.2).

7.4 Study 4: Pen and Paper 2

We then suspected that perhaps we were not manipulating disfluency as in the original study [3]. We contacted the authors, requesting the original research materials, which they promptly supplied. Their questionnaire did not use a table format as our did, and was more compact. Otherwise, the font, grey, and italics were the same. The disfluent versions of the 33-item materials for Study 3 and 4 are shown in Appendix A to enable comparison.

We ran our fourth study on two additional samples: one using the long, 33-item version (N=59, 69.5% females, average age 19.3, SD=1.8), and one using the short, 10-item version (N=61, 57.4% female, average age 19.3, SD=2.7). Unfortunately, as shown in 5.2, we again found no statistically significant difference between the disfluent and clear conditions in either of these samples.

7.5 Results Summary

We ran four different studies, one with 390 on-line participants, and three studies with between 59 and 93 participants. We compare these to the original studies which used 31 and 34 participants. Our smaller studies of undergraduate students more closely resembled the original study's participants, which were also undergraduates. Across all our studies we found disfluency to not have a statistically significant effect in eight of the ten instances. In only one case (the sensitive questions in Study 1), disfluency had a significant effect in the expected direction (participants in the disfluent condition disclosed less). However, in another case (the follow-up experiment to Study 1 using sensitive questions only), the effect was statistically significant, but in the opposite direction (participants in the disfluent condition disclosed more). Thus, in summary, it can be seen that the effect of disfluency on self-disclosure is not reliable or consistent. We find no evidence that disfluency impacts self-disclosure.

8 Re-Analysis and Discussion

Collectively, these failed attempts at replicating the effect of disfluency on self-disclosure puzzled us. We therefore decided to share our results with Alter and Oppenheimer and asked for their expert opinion on the matter. In response, they openly and promptly shared their data from the original tests, and commented that they had used a different statistical test to analyze their data.

A closer examination of the Alter and Oppenheimer's original paper suggested that they had used a did not use the independent means t-test, we we did (described above). Namely, the number for the degrees of freedom reported in their studies does not correspond to an independent t-test analysis (32 in Study 1a and 9 in Study 1b, in which Ns where 33 and 36 respectively). As confirmed through personal communications with the original authors, the tests used in the original paper were t-tests for

paired (and not independent) samples. Paired t-tests are typically used for within-subjects designs. For example, they can be used to show before and after results for a subject taking a treatment, or two different treatments on the same subject. In the case of Alter and Oppenheimer's studies, however, the researchers calculated a mean score for each item of the SDS in each of the conditions, resulting in a dataset containing 33 (or 10, in the short version) paired mean scores. Each of these mean item scores in each condition were treated as a single unit of observation, and subjected to a paired-samples t-test to examine the differences in the items' means between the conditions.

When treating individual questions as units of observation, one does not take into account the number of subjects that answered each question to determine significance. More importantly, the independence assumption on which the test is based is harder to justify, since the various questions are answered by the same participant and are, therefore, likely to be correlated. Additionally, the paired samples approach might provide an underestimation of the variance in the population, as within-subjects variance tends to be smaller than between-subjects variance.

After receiving from the authors of the original paper the original raw data for Studies 1a and 1b, we re-analyzed it using independent samples t-tests instead of the paired t-test that was reported in the original paper. We discovered that the results were not statically significant (see 5.2). Finally, we re-analyzed our own data, in all of our studies and for all of the measures that used multiple questions (i.e., excluding the gift-cards question), using a paired t-test approach. In our studies, a paired t-test did not yield significant results, except for one case (Study 2; see 5.2). A meta-analysis on the results of all independent t-tests showed the average test value to be of Z = 1.02, and not statistically significant (p = .31). A similar meta-analysis on the results of all of the paired t-tested showed a similar, non-significant, result (Z = .93, p = .36). To summarize, these re-analyses showed that the null hypothesis which states that disfluency does not affect self-disclosure could not be rejected. In other words, no consistent or reliable effect could be found for disfluency on self-disclosure, using either the independent samples t-test or the paired t-test.

9 Conclusions

Two conclusions may be drawn from the results of our studies. First, and most importantly, we did not find evidence that disfluency consistently impacts self-disclosure, or at least not in a manner that our studies, based on [3], could precisely estimate. The inconsistency of this effect was observed on a number of measures (including SDS, unethical behaviors and sensitive personal questions), in a variety of administration forms (online surveys, in person, and pen-and-paper) and in different samples (students and non-students). Thus, if an interface or system designer wished to use fluent or disfluent fonts in an attempt to encourage higher or lower disclosure by users, she will find that disfluency is not a reliable solution. Security and privacy designers should be aware that disfluency might slow-down their users, but would not cause them to reveal less.

Second, the original reported effect of disfluency on self-disclosure might have been due to the use of a very specific statistical test, whose appropriateness in this case could be questionable. Specifically, the original analysis was conducted as an 'item-level' analysis instead of the more widely used 'subject-level' analysis. We are not fully convinced that a paired samples t-test is appropriate for this type of analysis. Our failures in replicating the disfluency results could therefore serve to raise a discussion in the community as to what is the more valid statistical approach to be used in similar instances.

10 Acknowledgments

We thank Adam Alter and Daniel Oppenheimer for sharing their data and analyses with us, and for their prompt responses and cooperation with our queries. We also thank Howard Seltman for his expert statistical advice. This work was funded by the National Science Foundation under Grants CNS-1012763 (Nudging Users Towards Privacy) and a Google Focused Research Award on Privacy Nudges.

References

[1] ACQUISTI, A. Privacy in Electronic Commerce and the Economics of Immediate Gratification. In *Proceedings of the 5th ACM conference on Electronic commerce* (2004), ACM, pp. 21–29.

[2] ACQUISTI, A., JOHN, L., AND LOEWENSTEIN, G. What is privacy worth. In *Workshop on Information Systems and Economics (WISE)* (2009).

[3] ALTER, A. L., AND OPPENHEIMER, D. M. Suppressing secrecy through metacognitive ease cognitive fluency encourages self-disclosure. *Psychological science 20*, 11 (2009), 1414–1420.

[4] ALTER, A. L., OPPENHEIMER, D. M., EPLEY, N., AND EYRE, R. N. Overcoming intuition: Metacognitive difficulty activates analytic reasoning. *Journal of Experimental Psychology-General 136*, 4 (2007), 569–576.

[5] BERINSKY, A. J., HUBER, G. A., AND LENZ, G. S. Using mechanical turk as a subject recruitment tool for experimental research. *Submitted for review* (2011).

[6] BRANDIMARTE, L., ACQUISTI, A., AND LOEWENSTEIN, G. Misplaced confidences: Privacy and the control paradox. *Social Psychological and Personality Science* (2012).

[7] CRANOR, L. F. A framework for reasoning about the human in the loop. In *Proceedings of the 1st Conference on Usability, Psychology, and Security* (2008), vol. 1, USENIX Association, pp. 1–1.

[8] CROWNE, D. P., AND MARLOWE, D. A new scale of social desirability independent of psychopathology. *Journal of consulting psychology 24*, 4 (1960), 349.

[9] DIEMAND-YAUMAN, C., OPPENHEIMER, D. M., AND VAUGHAN, E. B. Fortune favors the (): Effects of disfluency on educational outcomes. *Cognition 118*, 1 (2011), 111–115.

[10] HARPER, V., AND HARPER, E. J. Understanding student self-disclosure typology through blogging. *The Qualitative Report 11*, 2 (2006), 251–261.

[11] JOHN, L. K., ACQUISTI, A., AND LOEWENSTEIN, G. Strangers on a plane: Context-dependent willingness to divulge sensitive information. *Journal of Consumer Research 37*, 5 (2011), pp. 858–873.

[12] JOHNSON, T. P., AND FENDRICH, M. A validation of the crowne-marlowe social desirability scale. In *57th Annual Meeting of the American Association for Public Opinion Research* (2002).

[13] LEITE, W. L., AND BERETVAS, S. N. Validation of scores on the marlowe-crowne social desirability scale and the balanced inventory of desirable responding. *Educational and Psychological Measurement 65*, 1 (2005), 140–154.

[14] MOON, Y. Intimate exchanges: Using computers to elicit self-disclosure from consumers. *Journal of Consumer Research 26*, 4 (2000), 323–339.

[15] MOON, Y. Intimate exchanges: Using computers to elicit self-disclosure from consumers. *Journal of Consumer Research 26*, 4 (2000), pp. 323–339.

[16] MOREWEDGE, C. K., AND KAHNEMAN, D. Associative processes in intuitive judgment. *Trends in Cognitive Sciences 14*, 10 (2010), 435 – 440.

[17] OPPENHEIMER, D. M. The secret life of fluency. *Trends in Cognitive Sciences 12*, 6 (2008), 237 – 241.

[18] REYNOLDS, W. M. Development of reliable and valid short forms of the marlowe-crowne social desirability scale. *Journal of clinical psychology 38*, 1 (1982), 119–125.

[19] SMITH, S. Humans in the loop: human-computer interaction and security. *Security Privacy, IEEE 1*, 3 (2003), 75–79.

[20] STRAHAN, R., AND GERBASI, K. C. Short, homogeneous versions of the marlowe-crowne social desirability scale. *Journal of clinical psychology* (1972).

[21] TOURANGEAU, R. Survey research and societal change. *Annu. Rev. Psychol. 55* (2004), 775–801.

[22] WHITTY, M. T., AND JOINSON, A. N. Truth, lies and trust on the internet. *(2009). Truth, lies and trust on the Internet. vi, 176 pp. New York, NY, US: Routledge/Taylor & Francis Group* (2009).

A Pen and Paper Screenshots

The following screenshots show the disfluent (Times new-roman, 10 points, grey) version of the two pen and paper studies: Study 3 (top) and Study 4 (bottom). These screenshots demonstrate that the disfluent versions were light and difficult to read.

Following are 33 statements to which some people agree and some do not.
Please mark whether you agree or do not agree with each of the following statements.

	Agree	*Disagree*
1. Before voting, I thoroughly investigate the qualifications of all the candidates		
2. I never hesitate to go out of my way to help someone in trouble		
3. It is sometimes hard for me to go on with my work if I am not encouraged		
4. I have never intensely disliked anyone		
5. On occasion I have had doubts about my ability to succeed in life		
6. I sometimes feel resentful when I don't get my way		
7. I am always careful about my manner of dress		
8. My table manners at home are as good as when I eat out in a restaurant		
9. If I could get into a movie without paying and be sure I was not seen I would probably do it		
10. Of a few occasions, I have given up doing something because I thought too little of my ability		
11. I like to gossip at times		
12. There have been times when I felt like rebelling against people in authority even though I knew they were right		
13. No matter who I'm talking to, I'm always a good listener		
14. I can remember "playing sick" to get out of something		
15. There have been occasions when I took advantage of someone		
16. I'm always willing to admit it when I make a mistake		
17. I always try to practice what I preach		
18. I don't find it particularly difficult to get along with loud mouthed, obnoxious people		
19. I sometimes try to get even rather than forgive and forget		
20. When I don't know something I don't at all mind admitting it		
21. I am always courteous, even to people who are disagreeable		
22. At times I have really insisted on having things my own way		
23. There have been occasions when i felt like smashing things		
24. I would never think of letting someone else be punished for my wrong-doings		

	Agree	*Disagree*
25. I never resent being asked to return a favor		
26. I have never been irked when people expressed ideas very different from my own		
27. I never make a long trip without checking the safety of my car		
28. There have been times when I was quite jealous of the good fortunes of others		
29. I have almost never felt the urge to tell someone off		
30. I am sometimes irritated by people who ask favors of me		
31. I have never felt that I was punished without cause		
32. I sometimes think when people have a misfortune they only got what they deserved		
33. I have never deliberately said something that hurt someone's feelings		

Please state your gender: □ Male □ Female
What is your age? _____
Are you a(n) □ undergraduate or □ graduate student?

Thank you very much!

Listed below are a number of statements concerning personal attitudes and traits. Read each item and decide whether the statement is True or False as it pertains to you personally.

1. Before voting I thoroughly investigate the qualifications of all the candidates.
2. I never hesitate to go out of my way to help someone in trouble.
3. It is sometimes hard for me to go on with my work, if I am not encouraged.
4. I have never intensely disliked anyone.
5. On occasion I have had doubts about my ability to succeed in life.
6. I sometimes feel resentful when I don't get my way.
7. I am always careful about my manner of dress.
8. My table manners at home are as good as when I eat out in a restaurant.
9. If I could get into a movie without paying and be sure I was not seen, I would probably do it.
10. On a few occasions, I have given up doing something because I thought too little of my ability.
11. I like to gossip at times.
12. There have been times when I felt like rebelling against people in authority even though I knew they were right.
13. No matter who I'm talking to, I'm always a good listener.
14. I can remember "playing sick" to get out of something.
15. There have been occasions when I took advantage of someone.
16. I'm always willing to admit it when I make a mistake.
17. I always try to practice what I preach.
18. I don't find it particularly difficult to get along with loud-mouthed, obnoxious people.
19. I sometimes try to get even rather than forgive and fdrget.
20. When I don't know something I don't at all mind admitting it.
21. I am always courteous, even to people who are disagreeable.
22. At times I have really insisted on having things my own way.
23. There have been occasions when I felt like smashing things.
24. I would never think of letting someone else be punished for my wrongdoings.
25. I never resent being asked to return a favor.
26. I have never been irked when people expressed ideas very different from my own.
27. I never make a long trip without checking the safety of my car.
28. There have been times when I was quite jealous of the good fortune of others.
29. I have almost never felt the urge to tell someone off.
30. I am sometimes irritated by people who ask favors of me.
31. I have never felt that I was punished without cause.
32. I sometimes think when people have a mistortune they only got what they deserved.
33. I have never deliberately said something that hurt someone's feelings.

Please fill out the following items:

Age:_____ Sex: _____ Ethnicity/Race: _____

How frustrated do you feel at the moment (rated from 1 to 10, where 1 is not at all frustrated, and 10 is very frustrated)? _____

Experimental Security Analyses of Non-Networked Compact Fluorescent Lamps: A Case Study of Home Automation Security

Temitope Oluwafemi[1], Sidhant Gupta[2], Shwetak Patel[1, 2], Tadayoshi Kohno[2]
[1]*Elec. Eng.*, [2]*Comp. Sci. & Eng.*
University of Washington
Seattle, WA
{oluwat, sidhant, shwetak, kohno}@uw.edu

ABSTRACT

Background. With a projected rise in the procurement of home automation systems, we experimentally investigate security risks that homeowners might be exposed to by compact fluorescent lamps (CFL), where the lamps themselves do not have network capabilities but are controlled by compromised Internet-enabled home automation systems.

Aim. This work seeks to investigate the feasibility of causing physical harm—such as through the explosion of CFLs—to home occupants through an exploited home automation system.

Method. We set up a model of a compromised automated home; placing emphasis on a connected Z-Wave enabled light dimmer. Four distinct electrical signals were then applied to two different brands of CFLs connected to a Z-Wave enabled light dimmer until they popped[1] or gave way[2].

Results. Three of ten CFLs on which we conducted our experiments popped, although not to the degree of explosions we expected. The seven remaining CFLs gave way with varying times to failure indicating process and design variations. We did find that it was possible to produce fluctuations at an appropriate frequency to induce seizures. We were also able to remotely compromise a home automation controller over the Internet. Due to timing constraints, however, we were only able to compromise the light bulbs via an adversary-controlled device using open-zwave libraries, and not via the compromised controller.

Conclusions. Our results demonstrated that it will be hard for an attacker to use the described methods to harm homeowners, although we do demonstrate the possibility of attacks, particularly if the homeowner suffers from epilepsy. However, and more importantly, our work demonstrates that non-networked devices—such as light bulbs—might be connected to networked devices and hence can be attacked by remote adversaries.

[1] We define popped as the visual or auditory observance of a spark in the CFL.

[2] The term "give way" refers to the normal failure of a CFL without a spark.

General Terms

Experimentation, Measurement.

Keywords

Home automation systems, cyber-physical systems, computer security, cyber-physical security, compact fluorescent lamps, CFLs

1. INTRODUCTION

To date, few experimental computer security research efforts have focused on computer systems that interact directly with the physical world—the so-called cyber-physical systems. There are, of course, some exceptions, e.g., various software attacks have been successfully demonstrated on cars, printers, robots and pacemakers with physical consequences as shown in [1, 3, 4, 10 and 11]. However, we argue that the field of experimental computer security research for cyber-physical systems is still in its infancy. This is partly due to the fact that cyber-physical systems are just emerging on the commercial market, but the greater challenge has been how to conduct research in this space. Significant, important issues can and do arise when attempting to experimentally evaluate the security of a cyber-physical system—issues that are not normally encountered, at least not in the same form—when experimenting with conventional non-cyber-physical systems. For example, is it possible to reconstruct the environment for the cyber-physical system in a laboratory setting in sufficient detail in order to ensure experimental validity? And is it possible to conduct the experiments in a way that does not jeopardize anyone's safety?

In this work we describe experiments that we conducted with an emerging class of cyber-physical systems: home automation systems. Many home automation systems already exist in the market, and recent worldwide market forecasts by Berg Insight claim that revenues generated through the sales and purchases of home automation units will grow at a compound annual rate of 33% from $2.3 billion USD in 2010 to nearly $9.5 billion USD in 2015 [19]. Home automation systems are often Internet-connected, and indeed—as an example of such connectivity—the number of cellular connections used by home automation units are expected to grow worldwide at a compound rate of 85.6% from 0.25 million in 2010 to 5.5 million connections in 2015 [19]. Home automation systems allow homeowners to control appliances—e.g., lights or ovens—from another device (such as a laptop)

within the home, or even over the Internet from a mobile device.

We began by obtaining two mainstream home automation systems and subjected them to a number of experiments. We describe briefly the totality of our work since we believe that it is important to understand the full context for our research, but foreshadow here that the bulk of this paper is focused on our experimentation with a seemingly unlikely target: light bulbs. Returning to the full context, we experimentally found that the home automation systems we acquired are vulnerable to remote attacks. We experimentally verified that an attacker—even from someplace outside a home, i.e., over the Internet—could violate the sanctity of the home by, for example, turning on or off home automation-connected devices (like light dimmers and HVAC systems) and even unlocking a home's front door or disabling a networked alarm system. We also found that an attacker could learn which devices are in a home and connected to the home automation network, thereby violating the homeowner's privacy. We also found that an attacker could control switches and dimmers in the home. While we identified and experimentally demonstrated these vulnerabilities with the home automation systems that we purchased, we note that others have made similar observations before, e.g., [5 and 8].

One of the capabilities mentioned above—that an Internet-connected attacker can remotely control switches and dimmers—may not sound significant at first. But herein lies what we believe is a fundamentally interesting property: there *is* the potential for an attacker to affect a device plugged into an outlet by maliciously controlling the outlet in certain ways. Certainly an attacker could use this capability to turn something connected to the outlet on and off or alter the brightness of a light bulb using a dimmer. While such actions might initially seem to only create nuisances for home occupants, after further contemplation, we began to speculate on whether an attacker could also use this simple capability to enact significantly more *physical* damage to the home environment. Concretely, one question we asked was: would it be possible for an attacker to make a light bulb connected to the network-controlled outlet explode?

Modern lighting solutions, such as CFLs and LED lamps, are designed to be efficient and thus increasingly make use of sophisticated electronic circuitry when compared to traditional incandescent light bulbs. We hypothesized that by altering the supply voltage characteristics to such devices, they could be made to operate beyond safe specifications of the electronic circuitry. We argue that knowing whether it would be possible to explode a light bulb remotely would be valuable for the computer security community. If possible, then defenses would need to be created before home automation systems become more ubiquitous and the risks increase.

Of course, we could not enter into an investigation of "can we experimentally explode light bulbs" lightly. In fact, we nearly did not proceed with this line of investigation because we did not know how to proceed both safely and in a cost effective manner. For example, how would we contain an explosion, should one occur? And how would we handle the leak of chemicals, should the physical damage to a light bulb cause some of its internal chemicals to leak. Fortunately, after significant research into possible options, we did derive a method. We use a glove box, which provides a controlled and well-ventilated environment to help contain and clean up hazardous materials. Another thing we learned to be cautious about during our experiments: the potential to induce seizures by fluctuating power to a light bulb.

While we did end up making light bulbs "pop", the "pops" were not nearly as significant as the worst-case explosions that we had feared. We also found that an attacker can cause the light bulbs to oscillate at a frequency known to cause seizures. From a security perspective—and the perspective of the homeowner—these results provide valuable insight into how secure these systems and connected devices are and the scale of attacks that can be mounted against them. The results indicate that it will be harder for an attacker to use these exact techniques to harm homeowners, such as exposing the occupants of the home to the mercury content of CFLs. However, on a more serious note, the results clearly demonstrate that it *is* possible for a remote attacker to compromise something as simple as a light bulb—a technology that, by design, has no network connectivity itself. We view this observation as an important contribution of the paper, with the other main contributions being the experimental methodologies we discuss. This observation is an important contribution because it provides proof of plausibility that—in the future—other devices *without network connections* might be found vulnerable to *network-based* compromise in the future.

Stepping back, we observe that cyber-physical systems are becoming increasingly prevalent. As such, we expect to see increasing interest in experimentally evaluating the security properties of such cyber-physical systems. But, if these systems are vulnerable to security compromises, then the experiments—if successful—have the potential to cause harm to the experimental environment, and possibly even to the experimenter. Hence, we believe that the foundations we lay in this paper may be of value for future cyber-physical systems researchers.

In the following section, we define the problem we aim to solve. Section 3 gives a brief background on home automation systems, CFLs and related work. Section 4 presents a detailed analysis of the security vulnerabilities discovered in the home automation systems we examined. Section 5 presents an overview of the method describing our work with CFLs. We will conclude this paper by

examining the results from our experiments and discuss potential future directions for all stakeholders.

2. PROBLEM BEING SOLVED

The purpose of our research is to gauge the level of impact that a remote attacker might have on the inhabitants of an automated home, particularly in regards to manipulating appliances, like CFLs, that do not have network capabilities. We specifically sought to explore the possibility of causing physical harm through the application of known electric signals to CFLs controlled through wireless light dimmers.

Our hypotheses is based on the incorrect use of non-dimmable CFLs with wirelessly controlled light dimmers. Since most CFLs cannot be dimmed using a traditional triac-based dimmer[3], manufacturers may not test against such specifications (using a CFL in a dimmer) and/or guard against these situations. There are newer CFLs that can be dimmed and these bulbs sense the dim level and internally regulate the power to the bulb. Our focus in this paper is on standard CFLs. Clearly one could simply use a non-dimming appliance module, but our assumption is a person might use a dimmable module without knowing the consequences. Consequently, there is a possibility of a current spike in non-dimmable CFLs used with dimmers that can result in fires. Hence, we hypothesize that an attacker can mount an attack to cause an explosion with the possibility of starting a fire and/or releasing harmful mercury contents of CFLs when connected to remotely compromised and controlled light dimmers.

To investigate the plausibility of our hypothesis, we describe experiments using open-zwave libraries to control Z-Wave enabled light dimmers with connected CFLs. We conduct these experiments in a glove box to provide a shield from shattered glass and to contain the mercury content of CFLs, in the event of an explosion.

As with most computer security vulnerability efforts, the results of this research can inform the design of future home automation systems and/or light bulbs (and other devices that might connect to home automation systems). We believe that now is the time to perform such research, before these systems become ubiquitous and the risks of any (possibly unknown) vulnerability increases.

3. BACKGROUND AND RELATED WORK

3.1 Home Automation

Most home automation systems consist of a primary controller that controls a variety of connected secondary nodes which include but are not limited to, door locks, alarm systems, HVAC and sprinkler systems, light modules (dimmers), and energy monitoring nodes as shown in Figure 1. Although some home automation systems use WiFi for communications between secondary nodes, data is usually sent through a low power and low data rate wireless communication standard such as Z-Wave or ZigBee. It is also usually the case that most primary controllers are equipped with both WiFi and Z-Wave or ZigBee for added connectivity to the Internet.

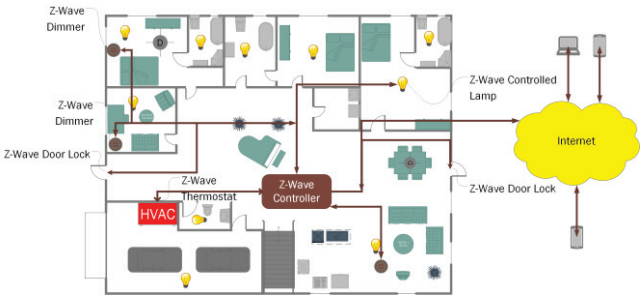

Figure 1: Home automation model.[4]

3.2 Compact Fluorescent Lamps

CFLs are fast becoming the standard for electric bulbs as countries around the world are beginning to phase out incandescent bulbs due to their power inefficiency. CFLs provide about seventy-five percent savings in energy when compared to incandescent bulbs. Newer CFLs are mostly integrated with electronic ballasts, while older models use large and heavy magnetic ballasts [15].

Most common CFLs integrated with electronic ballasts have more complex circuitry and active electronic components than incandescent light bulbs. Standard CFLs are also not supposed to be used along with dimmers as the current drawn by the lamps increases by a magnitude of about five times their normal operation [6]. There have been instances of fires caused by using CFLs with dimmers [18 and 20]. Furthermore, most CFLs contain about 3-5mg of mercury, which is harmful and constitutes environmental waste.

Given the composition of CFLs and their mode of operation, they appear to be appealing targets for a potential attacker with an intent to physically harm occupants of automated homes.

Moreover with automated homes, attackers have the ability to remotely control CFLs connected to dimmers and light switches by sending arbitrary signals, as we demonstrate in Sections 4 and 5. We again stress that such adversarial capabilities are possible *even though* the CFLs themselves do *not* have any built-in network capabilities.

3.3 Related Work

As mentioned, others have investigated the security and privacy of other classes of cyber-physical systems. For example, Checkoway and others in [1] were able to demonstrate the remote compromise of automobiles, providing attackers with the ability to remotely disable brake systems and eavesdrop on in-vehicle conversions. They were able to highlight consumer safety concerns and

[3] See Section 3.2.

[4] The house is assumed to be retrofitted with a variety of automated appliances, sensors, actuators and controllers.

motivate car manufacturers to develop more secure and robust defenses against such attacks.

Halperin et al. [11] and Gollakota et al. [10] demonstrated how an implantable cardiac defibrillator could be remotely compromised using software radios; Denning and others in [4] demonstrated some cyber-physical exploits in robots used in homes.

Printers were also shown to be vulnerable by Cui et al. in [3] through the remote modification of their firmware and resulting compromise to cause possible fire outbreaks, in addition to providing exfiltration capabilities of privately printed documents.

In the context of the home, and the ever increasing array of connected appliances with sentimental value, Denning et al. [5] investigated and highlighted various entry points for the tech savvy criminal to infiltrate the home. Fouladi et al. [8] also demonstrated exploits taking advantage of vulnerabilities in the Z-Wave protocol stack. Similar to these, [12 and 21] also identified some flaws and mounted some attacks in both Z-Wave and ZigBee implementations respectively,

All of these findings stress the need for more emphasis to be placed on the security and privacy of cyber-physical systems. This is due to the fact that these systems, unlike most traditional computing systems have the capability to effect changes in the physical world. We argue that home automation systems are as critical as the aforementioned cyber-physical systems, as these classes of systems are in direct and prolonged contact with humans in the comfort of their homes.

4. REMOTE COMPROMISE AND CAPABILITIES

We now describe several successful attempts at remotely compromising both of the home automation systems that we purchased. The vulnerabilities we uncover are a result of not following standard security best practices, so the vulnerabilities themselves are not novel contributions. However, we include these vulnerabilities because they underscore an important point: that future home automation systems may be vulnerable to compromise, and that it is important to follow-through with understanding the implications of those compromises and explore opportunities for defense-in-depth so that, if compromised, the damages can be mitigated. As noted, others have also evaluated the security of home automation systems, e.g., [5, 8, 12 and 21].

We have notified the relevant manufacturers about the vulnerabilities so that the vulnerabilities can be patched. Since the vulnerabilities are not novel, and since we have no reason to believe that other home automation systems are more secure, we have chosen not to mention product makes and models in this paper.

4.1 Experimental Setup

We chose two brands of Z-Wave enabled home automation systems for consistency. We will refer to the first of the two products as product A while the second will be referred to as product B. Product A requires an external Z-Wave module to be connected to it and exposes a web interface which allows the homeowner to connect to the system over the Internet. Once connected, the homeowner can control lights, door locks, thermostats, and other connected devices.

Product B on the other hand, used a built-in Z-Wave module. Product B also exposes a web interface for monitoring web cam feeds and the alarm system. Remote control of Z-Wave enabled appliances is also possible through the provided web interface.

Remote connectivity to these systems is achieved through port forwarding on the homeowner's router. Both systems have premium services to provide this feature.

In addition to these controllers, we had Z-Wave enabled door locks, thermostats, light dimmers and binary switches all connected to these controllers to closely simulate the use of these appliances in the home. Moreover, it was important to analyze how these nodes are affected in the event of a security breach.

4.2 XSS Vulnerability

Through extensive investigations, we found that we were able to embed persistent JavaScript tags in the logs page of product A. This was possible because product A kept a log of all login attempts, including the username, without properly parsing and sanitizing the username input. Hence, in place of a valid username, an attacker can enter JavaScript code that will be included in the logs of the system. The consequence of this is, whenever the homeowner views the log page, persistent JavaScript code executes and the attacker can do whatever he or she wishes. Moreover, the attacker can mount a covert attack by erasing the logs afterwards.

We wrote some JavaScript code to exploit this vulnerability. The embedded JavaScript code, when executed, will create a new user with arbitrary credentials and escalated privileges. We ensured the covertness of our attack by embedding the core functionality of our exploit in an *iframe* not visible to the homeowner. We also cleared the logs to erase any trace of a newly added user. For security reasons, we chose not to publish this exploit and informed the manufacturers of product A.

Extensive work has been done to exploit XSS vulnerabilities as illustrated in [13 and 14].

4.3 Insecure HTTP

Using plain HTTP on all pages was also a prevalent problem we noticed in product A. Every communication that we observed with the unit is sent in the clear whether the homeowner accesses the controller on his or her home network or over the Internet. An attacker can eavesdrop on

credentials including usernames, passwords and other valuable information. Because it is easy for an attacker to intercept wireless communications, if a user logs into product A over the Internet such as via the wireless Internet at a coffee shop, then an attacker at the same location may be able to intercept those wireless communications, learn the user's credentials, and then use those credentials for him or herself in the future.

4.4 Miscellaneous Attack Vectors

Product A also had a VNC server enabled by default with a password of "admin" running on a fixed high-numbered port. This service was however only enabled for LAN access and could not be remotely accessed. Nevertheless, a local attacker could gain access to this service.

Furthermore, product A allows developers to design various plugins, scripts or applications to enhance functionality of their units. While this provides room for innovation on many fronts, there are apparent security and privacy risks associated with this model. The question of how well vetted these scripts, plugins or apps are before being distributed to their respective application stores, begs to be asked. Can a developer with malicious intents distribute packages on a large scale through app stores? With product A, we suspect this to be possible since there does not appear to be a vetting process to ensure that apps do not infringe on the security (digital and physical) and privacy of homes and individuals using their product; we did not, however, experimentally attempt to distribute a malicious app.

As for product B, it stored a very simple and predictable authentication cookie on the user's browser which was not associated with any session id or expiration time frame. As a result, by adding this cookie to the browser, we were able to by-pass the authentication page and had direct access to the control panel. Hence, the only hurdle left for an attacker to gain access to the control panel of the system is obtaining the IP address of product B or the IP address of the homeowner's router and the specific port that product B is bound to. The latter option depends on port forwarding being enabled on the router of the homeowner.

4.5 Implications of Vulnerabilities

We experimentally verified that—after compromising products A and B—an adversary would be able to control other Z-Wave connected devices in the home. For example, we experimentally verified that an attacker could lock and unlock a Z-Wave door lock that we purchased. We also found that an attacker could turn on and off power-hungry devices, such as HVAC systems and home appliances, if connected to a Z-Wave switch. There are clear negative consequences to such capabilities, ranging from allowing an attacker easy access to a house (a broken door or scratches inside a door lock would provide evidence of forced entry, whereas a door unlocked via a remote exploit may not provide such evidence) to allowing an attacker to control power-hungry devices in the home (and potentially impacting the homeowner financially).

We do not consider the above capabilities any more. For the rest of this paper, we focus on what an attacker might be able to do to a perhaps surprising target—a non-dimmable CFL. The CFL has no network connectivity itself. For this study, we assume that the homeowner has physically plugged a standard CFL into a Z-Wave-connected dimmer. We note that such CFLs should not be plugged into dimmers and hence our analyses explicitly take the bulbs outside their intended operating environment. We do not know how many homeowners will plug CFLs into dimmers, though there have been instances of such incorrect usage as evidenced by [20]. A key question that we ask ourselves is whether it would be possible to use vulnerabilities in a home automation system to attack a device that, by itself, does *not* have any network connectivity—the light bulb.

5. APPROACH AND METHOD

We now describe our approach to experimentally analyze the range of attacks that homeowners may be exposed to via CFLs and compromised home automation systems.

5.1 Safety Measures

We needed to make sure we were working in a safe environment since we anticipated a chance of glass shattering or a more severe scenario in which we would have been exposed to the mercury content of CFLs. We initially decided to design an enclosure from Plexiglas as shown in Figure 2. Our initial assessment was that this would be effective in protecting against shattered CFLs, but that it would not adequately protect against mercury vapor; hence, we did not use this enclosure for our experiments. We also considered using gas masks to ensure our safety, but cleanup of residual mercury vapor is a non-trivial task since the vapor could be persistent.

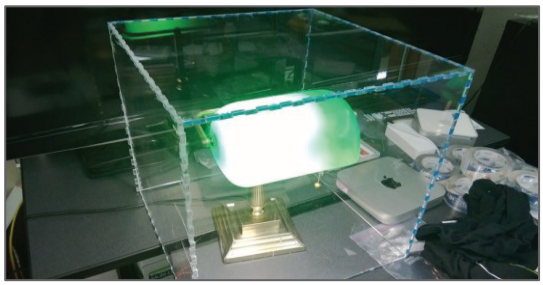

Figure 2: Plexiglas structure.

Being computer scientists and electrical engineers, we did not immediately know how to proceed and contemplated not being able to conduct our experiments. However, upon further research, and following EPA's recommended guidelines for cleaning up a broken CFL [2], we settled on using a properly ventilated glove box, shown in Figure 3. The glove box ensured that mercury vapor and shattered glass, if any, would be well contained and properly cleaned up. Another challenge was to figure out how to supply the CFL with electricity in the glove box while ensuring that it remained airtight. We improvised by drilling conduits within rubber stoppers shown in Figure 4 and carefully

sealed it up with adhesives to ensure that there would be no vapor leakage.

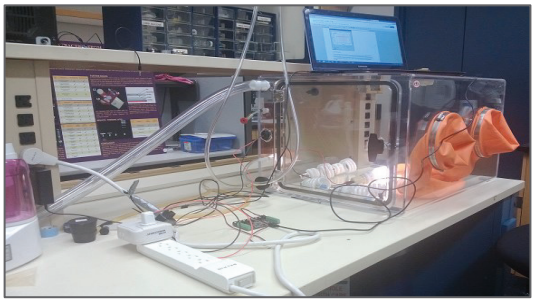

Figure 3: Glove box used to contain shattered glass and mercury.

Due to fire safety concerns, we had to be physically present when conducting our experiments. We did not have the luxury of many computer science experiments where tasks could be left to run with results viewed at a convenient time.

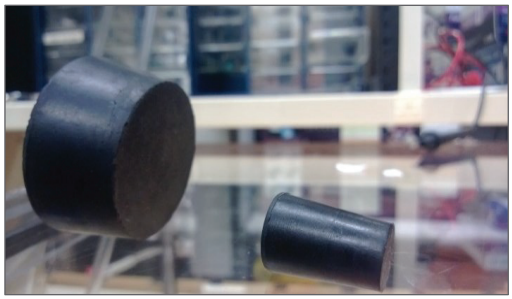

Figure 4: Rubber Stoppers.

5.2 CFL Current Monitor

In conducting our experiments, it was necessary to know how much current (RMS) was flowing through the CFL because this information would help us keep track of operation anomalies and help us recognize patterns of failure in the CFL. We acquired a Phidgets current sensor and an interface kit shown in Figure 5, with the capability of providing 125 samples per second. We also designed a graphical user interface shown in Figure 6 to aid visualization.

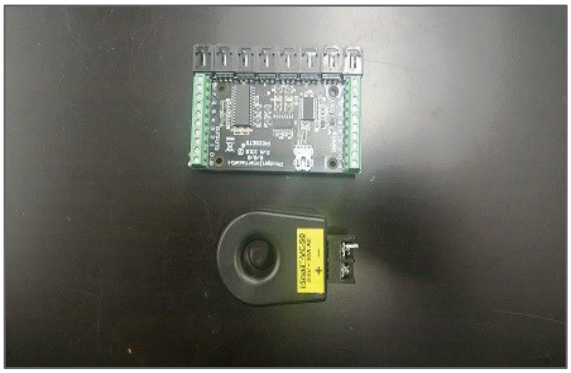

Figure 5: Phidgets Interface Kit and Current Detector.

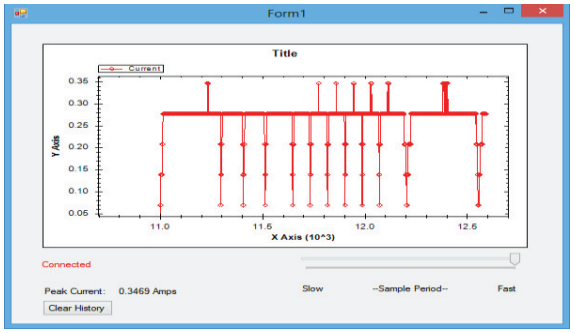

Figure 6: Real-time Plotting Utility Reporting Current Consumption.

5.3 AC Box

Similar to the need to measure current flowing through the CFL, we also logged the voltage waveform driving the CFL. Unlike current, where we log the RMS, the entire voltage waveform was recorded since shape of the waveform can change drastically depending on the load and dim level rather than the amplitude.

To safely measure the voltage, we galvanically isolated the measurement equipment from the AC-line using a step-down transformer (Triad Magnetics part F12-090-C2-B) with an approximate coil ratio of 1:16 under full load (a 75 ohm resistor was placed across the secondary terminals to load the output). This allowed us to safely connect a bench oscilloscope to the secondary of the AC transformer. Figure 11 shows one instance of the recorded waveform using this approach.

5.4 Signal Generation

For the purpose of our experiments, we assume a naive homeowner has upgraded his home with a home automation system and has connected a CFL to a dimmer with remote control capabilities. We also assume the attacker has compromised the system through one of the aforementioned vulnerabilities and is intent on physically harming the occupants of the home by causing CFLs connected to dimmers to explode. Our experiments are designed to gauge what, if anything, an attacker might be able to accomplish. As additional background knowledge: lights fluctuating at certain frequencies can be dangerous to people with photosensitive epilepsy; CFLs contain mercury; and an exploded light bulb could result in shattered glass or possibly a fire outbreak [18 and 20].

To study this threat experimentally, we utilized an Aeon Z-stick® static update controller which uses the Z-Wave protocol for low data rate communications as shown in Figure 7. Additionally, we utilized open-zwave libraries to remotely control Z-Wave-enabled light dimmers, and connected to the dimmers were two different groups of CFL brands. We then generated four distinct signals and extensively tested them out on the CFLs until they either gave way or produced an anticipated result like a dramatic pop. Since the only parameter we could alter from a remote perspective was the Z-Wave dimmer level and considering

how time-intensive the experiments were, we were unable to experiment with a large number of signal types. We therefore chose the four signal types that we thought would cause the CFL to operate outside normal operating conditions.

Figure 7: Aeon Z-stick®.

Figure 8 shows periodic triangular pulses that were applied to the Z-Wave dimmer. With this mode of operation, a peak voltage level was chosen (as described below) and the voltage applied to the CFL was varied from zero to the chosen peak level and back to zero at a refresh rate from at least every second to about every 60 milliseconds. While the timing in addition to the signal, were chosen to closely simulate an individual physically varying the brightness of the CFL, we had upper bounds on the refresh rate due to the low data rate constraint of the Z-Wave protocol. The peak dimmer level shown in Figure 8 is arbitrary and can be set to any value between 0 and 100 (the range 0 and 100 correspond to the levels allowed by the dimmer). The peak voltage was chosen by observing the voltage level at which the CFL became unstable, i.e., at the onset of visual fluctuations. The level at which the CFL became unstable was largely affected by process and design variations.

From our experience, instability usually kicked in when the dimmer level was set to about 20% of the maximum brightness of the lamp. The reason why the peak voltage selection was important is that we observed through repeated experimentation that the CFL was more likely to fail at a faster, however inconsistent rate, when the selected voltage level induced visual fluctuations in the lamp. We again stress, however, the limited sample size of our experiments.

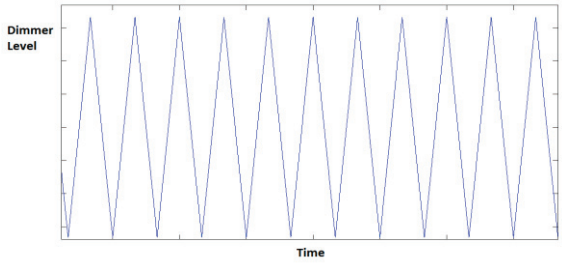

Figure 8: Periodic triangular pulses applied to Z-Wave Dimmer. Plot of Z-Wave dimmer level versus time.

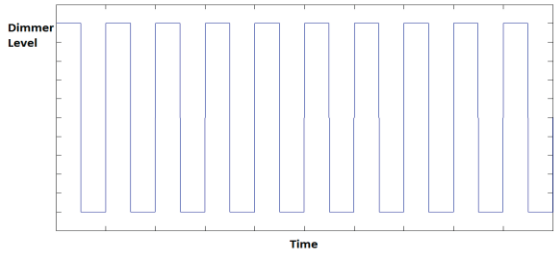

Figure 9: Periodic rectangular pulses applied to CFL. Plot of Z-Wave dimmer level versus time.

For the second signal, we toggled the applied voltage level between a peak voltage of our choice and zero at a refresh rate from at least every second to about every 60 milliseconds as shown in Figure 9. Again, the peak dimmer level shown in Figure 9 is arbitrary and can be set at any level between 0 and 100. In this case, we selected the peak voltage to be maximum. We selected this waveform as a simple variant of the periodic triangular pulses, though we acknowledge that other wave forms are possible too. Our original intentions with this signal was to cause the CFL to pop, but we soon realized that this signal might cause the light to flash at a seizure-inducing frequency (see results section).

For the third signal, we wanted to gauge whether a random signal might be effective at damaging the bulbs. Hence, we decided to add randomness by generating Gaussian distributed random numbers and decided against a certain threshold to either increase or decrease the applied voltage. An example plot of the applied random signal is shown below in Figure 10.

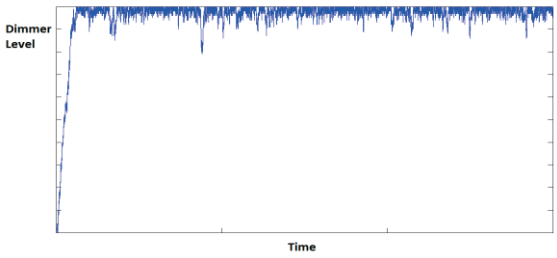

Figure 10: Random Gaussian distributed signal applied to CFL. Plot of Z-Wave dimmer level versus time.

Finally, we combined the triangular pulses shown in Figure 8 with some randomness from a Gaussian distributed random number generator similar to the signal shown in Figure 10. We also set the peak voltage as defined for the triangular pulses described earlier.

Table 1 has labels “Signal A”, “Signal B”, “Signal C” and “Signal D” attached respectively to the applied periodic triangular and rectangular pulses, the random Gaussian distributed signal and the periodic triangular-random Gaussian signal combo.

Table 1: Summary of Applied Signals.

Label	Characteristics
Signal A	Periodic Triangular Pulses
Signal B	Periodic Rectangular Pulses
Signal C	Random Gaussian Distributed Signal
Signal D	Periodic Triangular Pulses + Random Gaussian Distributed Signal

The effect that these applied signals have on the CFL are shown in Figures 11-13. The dimmer generates a pulse width modulated signal whose width is controlled by the applied dimmer level. Figure 11 shows the voltage plot across the terminals of the CFL when the dimmer level is set to 8, while Figures 12 and 13 show voltage plots across the terminals of the CFL with the dimmer set to levels 50 and 100 respectively.

Figure 11: Plot of voltage across the terminals of the CFL with dimmer level set to 8.

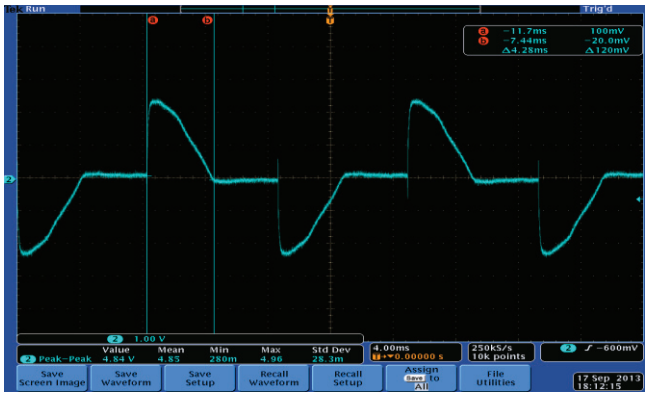

Figure 12: Plot of voltage across the terminals of the CFL with dimmer level set to 50.

Figure 13: Plot of voltage across the terminals of the CFL with dimmer level set to 100.

In real time, there is a progressive increase of the pulse width from the minimum to the maximum (Figure 13) when Signal A is applied and the peak dimmer level is set to 100. Once the voltage across the CFL reaches its maximum width, it shrinks and flattens out to zero. This is repeated until the CFL pops or gives way.

Similarly, the pulse width of the voltage plot across the terminals of the CFL changes between two values—maximum and minimum pulse widths—when Signal B is applied and the peak dimmer level is set to 100. For Signal C, the pulse width randomly increases or decreases depending on the set threshold, dimmer level and previous CFL voltage. Finally, Signal D is simply a combination of the effects Signals A and C have on the CFL.

6. RESULTS

6.1 Data and Analysis

Our experiments yielded a wide variety of results, including inconsistent times to popping the CFLs. We conducted several preliminary experiments to determine the most effective and safest way (from our perspective as researchers) to get the CFLs to fail. Table 2 shows some of the results we obtained through the application of the signals defined in Table 1. Through repeated experimentation, we found out that Signal A was the most effective in causing CFLs to fail, Signal B had a side effect of possibly triggering seizures, and Signal C had to be combined with Signal A for it to be as effective. We acknowledge that our sample sizes are small, however—an artifact of the resource intensiveness of conducting experiments with this class of cyber-physical systems.

We conjecture that the inconsistent times to failure is largely attributed to process and design variations among similar and different CFL brands. Even though the lifespan of the devices were ultimately reduced, the time to failure varied to a large extent. It is also important to note that we did not conduct this particular set of experiments over the Internet, but limited the scope to a local control of the Z-Wave controller using open-zwave libraries. We hope to experimentally evaluate an end-to-end attack as an

extension to the work in the future. We got some results that we believe the community would be interested in, as a component within the electronic ballasts of some of our test CFLs, specifically a bipolar junction transistor (BJT) dramatically burnt out with a "pop". This left some charring on the device as shown in the Figure 14 identified by the circular ring.

Table 2: Time to failure for CFLs. *Popped after direct connection to electricity without the dimmer.

Lamp Tag	Brand	Time (Hours)	Signal Type(s) Applied	CFL Status
#1	Walmart Great Value	0.3	Signal A	Gave way
#2	Walmart Great Value	0.8	Signal A	Gave way
#3	Walmart Great Value	7	Signal A	Gave way
#4	**GE**	**7**	**Signals A, B and C applied in no particular order**	**Popped**
#5	Walmart Great Value	3	Signal A	Gave way
#6	GE	0.6	Signal A	Gave way
#7	Walmart Great Value	4	Signal A	Gave way
#8	**GE**	**0.7**	**Signal A**	***Popped**
#9	Walmart Great Value	Over 6	Signal C	Settled in a state consisting of visual fluctuations.
#10	**GE**	**1.5**	**Signal D**	**Popped**

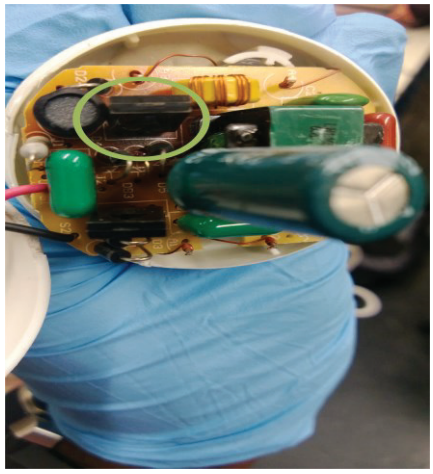

Figure 14: Charred CFL.

The recorded results shown in the Table 2 do not include several preliminary test runs we had, to determine the feasibility of inducing failures in CFLs.

For the recorded set of experiments, we initially started out by applying Signal A to the Walmart Great Value brand, which only resulted in the CFLs giving way at, however, inconsistent times. We also experimented with signals A, B and C by randomly applying them to the same CFL (lamp 4) in no particular order. This resulted in the first pop we observed after seven hours of experimentation.

After applying Signal A to lamp 8 (highlighted in orange in Table 2) for about 42 minutes, we noticed it was beginning to fail. To confirm its failure, we connected lamp 8 directly to the power source without the dimmer and heard a pop, indicating that a component (BJT) had given way in its ballast. The current spike that resulted from connecting the CFL directly to the power source is shown below in Figure 15. Depending on the kind of lighting fixture or shade around the light bulb, the heat generated from the failing bulb may pose a fire hazard. CFLs failing in this manner have been reported to cause major fire damage based on past recall reports [16 and 17]. No fires were ignited in our experiments, however.

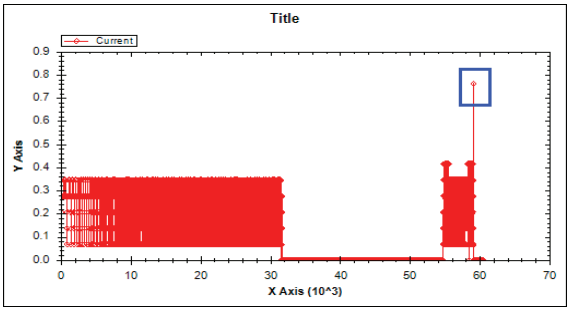

Figure 15: Resultant spike from current surge in CFL.

Taking this result into consideration, we tweaked Signal C by combining its mode of operation with Signal A to yield Signal D, The purpose of Signal D was to randomly cause a spike in the current flowing through the CFL at various

points during the experiment. This was necessary, as applying Signal C solely to the CFL was not yielding the desired result of either popping or giving way. As indicated in the previous section, we purposely set the peak voltage for Signals A and C to be reasonably low, to ensure that the CFL was in an unstable, flickering state. The voltage level that the CFL was set to varied from one lamp to the other and was due to process and design variations. The result of the Signal D was intermittent spikes in current from sporadically setting the dimmer to its maximum at times determined by the result of a Gaussian distributed random variable. We achieved the same result we got applying Signal A to the CFL with Signal D as evidenced with lamp 10.

Even though Signal B was intended to cause the CFL to pop, it may have a side effect of possibly causing a seizure; we did not experiment with this extensively after learning that the bulb was oscillating at a dangerous frequency [7]. Moreover, we did not initially anticipate that Signal B might be at a seizure-inducing frequency, but began to investigate that frequency after experiencing some discomfort from applying this signal to the CFL. For safety reasons, we did not run this experiment extensively, and when we did we took safety precautions (see Section 7 for details).

In summarizing our results, we set out to experimentally cause two different brands of CFLs to pop remotely by applying the signals shown in Figures 8-10 through a Z-Wave enabled light dimmer. Our results indicate that we were able to cause a reduced life-span, though inconsistent failure times, in the CFLs. More interestingly, we were able to cause some CFLs of the GE brand to pop with the BJT burning out. In our limited experiments, none of the pops caused serious damage to the external environment. Lastly, although we set out to pop CFLs using Signal B shown in Figure 9, we noticed a side-effect of possibly triggering seizures at the operated frequency of oscillation.

7. DISCUSSION

We stress that our demonstrated CFL attacks are not end-to-end. We demonstrated the ability for an attacker to remotely compromise and control two home automation system controllers, and from there we did confirm the ability of an attacker to do simple device manipulations, like unlock doors and turn on or off appliances. And we explored the feasibility of an attacker, connected to a wireless home automation network, to control a network-connected dimmer and thereby affect the CFLs plugged into the dimmer. However, we did not mount our attacks against the CFLs over the Internet to an uninstrumented home-automation ecosystem. A fundamental limitation was timing—using our current compromises to the home automation controllers, we were unable to send packets to the dimmer fast enough. Nevertheless, we argue that our current results are important because there are ways in which an adversary might be able to obtain internal access to a home automation system's internal wireless network. For example, more sophisticated code-injection attacks could be found against home automation controllers (e.g., full code injection rather than JavaScript injection). A nearby attacker might also attempt to attack the home automation system's wireless protocols directly, and thereby gain direct wireless access to the dimmers. An attacker might also produce Trojan home automation hardware, and unsuspecting users may connect that Trojan hardware to their home automation systems' wireless networks. The fundamental conclusion, therefore—that a network-based attacker might be able to affect a device that, by itself, is not designed to be networked (the CFLs)—remains true.

During the course of our experiments, we found out that there was no convenient and cost-effective way to detect mercury spillage. As a result, we are yet to experimentally verify the amount of mercury vapor, if any, leaked as a result of our experiments. Our glove box and ventilation system was, however, borrowed from a wet lab and was designed to deal with such vapors, whether detectable or not. Additionally, due to process and design variations, the failure times for the CFLs were very inconsistent. In certain cases, we were able to either get the CFL to fail with or without a pop in as little as eighteen minutes or as long as over seven hours. This is reflected in Table 2. We did not experiment with placing the CFLs next to lamp stands or accessories.

As mentioned, due to fire safety concerns, we had to be physically present when conducting our experiments. We did not have the luxury of most computer science experiments where tests could be left to run with results viewed at a convenient time. Also for safety, we needed to shield ourselves from staring directly at the fast switching Signal B shown in Figure 9, as it is in a frequency range that may induce a seizure in an observer [7]. While Signal B was not as effective as Signals A, C and D in terms of causing CFLs to pop, it caused discomfort to the observer; we implemented the safety precautions after experiencing this discomfort and realizing that the light was pulsing at a potential seizure-inducing frequency. Specifically, we covered the glove box with opaque black plastic bags to shield us from staring directly into the lamp. Future security research on cyber-physical systems must identify potential safety risks proactively, rather than reactively; proactive identification in all cases, however, may be fundamentally challenging if not impossible.

Sample size for cyber-physical systems research is another issue that the research community must address in the future. Some studies—such as past work on automobiles [1]—experimented with only two artifacts. We experimented with more light bulbs, but—given our limited resources—not nearly as many as we would have liked. For safety, our experiments required manual supervision, as noted above. This need for manual supervision is comparatively rare in computer science, and differentiates cyber-physical systems research from some other classes of

computer security research. Each experiment took significant time, further contributing to the small sample size. However, we acknowledge that our sample size is small and encourage future follow-on work to repeat our experiments with larger sample sizes, more signal variety, and more bulb types.

With all of these findings, it is necessary to take a step back to examine the consequences from the perspective of industry stakeholders, homeowners and also researchers.

From the perspective of industry stakeholders, it is important to stress the need for the design of more robust and secure home automation systems. This should encompass every party in the ecosystem ranging from those involved in the physical layer design to application developers who may unintentionally introduce vulnerabilities into the system. For instance, product A created a scenario like this, as the web interface was prone to XSS attacks as discussed.

For homeowners, there is an apparent trade-off between the convenience factor that home automation systems provide and security and privacy of the home. To what extent are homeowners willing to compromise security and privacy of the home for the ability to remotely control physical actuators around the home? Should homeowners be worried about inherent security flaws in the design of home automation systems and as such give the industry some time to mature and overcome these issues?

For researchers, a lot more needs to be done in this field to ensure that industry partners develop robust and secure home automation systems. Furthermore, with more heavy-duty home appliances increasingly connected to the Internet, detailed analysis of added connectivity benefits and resulting costs to security and privacy have to be carried out.

8. CONCLUSION

While home automation systems undoubtedly provide immense benefits in terms of convenience, more work needs to be done to ensure robust and secure designs of these systems. Furthermore, there is a need for all stakeholders involved—ranging from industry and research partners to homeowners—to fine-tune our understanding of whatever flaws these systems possess. We hope our work will further catalyze interest in discovering and fixing vulnerabilities in home automation systems, and their surrounding ecosystems, and enlighten end users to be cautious with their adoption and mode of use.

Of particular interest, we believe, is the fact that devices *not* designed for network connectivity (e.g., light bulbs) may be connected to other devices that *do* have network connectivity. Such connections may expose the former devices to risks that the designers of those devices never anticipated. The designers of the latter (networked) devices (like dimmers or entire home automation systems) may not know which other devices will connect to them in a home deployment, and hence providing sufficient protection mechanisms on the latter devices may be challenging. We encourage further research and design on secure home automation systems.

9. ACKNOWLEDGMENTS

We thank Professor Karl F. Böhringer for providing the lab space and necessary apparatus to conduct our experiments. We thank Dr. Carrie Gates for shepherding this paper and Karl Koscher for his help in conducting the experiments. We thank the numerous anonymous reviewers for their valuable feedback and recommendations. This work was supported by the Intel Science and Technology Center for Pervasive Computing.

10. REFERENCES

[1] S. Checkoway, D. McCoy, D. Anderson, B. Kantor, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analysis of Automotive Attack Surfaces," in Proceedings of the USENIX Security Symposium, San Francisco, CA, August 2011.

[2] Cleaning Up a Broken CFL, (N.D.), from U.S. Environmental Protection Agency. Retrieved June 26, 2013 from the U.S. Environmental Protection Agency: http://www2.epa.gov/cfl/cleaning-broken-cfl#instructions

[3] Cui, A., Stolfo, S. "Print Me If You Dare: Firmware Modification Attacks and the Rise of Printer Malware," in The 28th Chaos Communication Congress, December 27, 2011.

[4] Denning, T., Matuszek, C., Koscher, K., Smith, J. R., and Kohno, T. A spotlight on security and privacy risks with future household robots: attacks and lessons. In Ubicomp '09: Proceedings of the 11th international conference on Ubiquitous computing (2009), pp. 105-114

[5] Denning, T., Kohno, T., and Levy, H. M. Computer security and the modern home. Commun. ACM, 56(1):94–103, Jan. 2013.

[6] Elliot, R. Should There be a Ban on Incandescent Lamps?, February 22, 2007 from Elliott Sound Products. Retrieved June 26, 2013 from Elliott Sound Products: http://sound.westhost.com/articles/incandescent.htm#dim

[7] Photosensitivity and Seizures, (N.D.), from the Epilepsy Foundation. Retrieved June 26, 2013 from the Epilepsy Foundation: http://www.epilepsyfoundation.org/aboutepilepsy/seizures/photosensitivity/

[8] Fouladi, B., Ghanoun, S. Security Evaluation of the Z-Wave Wireless Protocol. In Black hat USA (2013).

[9] OpenZwave. (n.d.). Retrieved June 2013, from openzwave Google code site: https://code.google.com/p/open-zwave/

[10] Gollakota, S., Hassaneih, H., Ransford, B., Katabi, D., Fu, K. They can hear your heartbeats: non-invasive security for implantable medical devices: Proceedings of the ACM SIGCOMM 2011 conference, August 15-19, 3011, Toronto, Ontario, Canada.

[11] Halperin, D., Heydt-Benjamin, T.S., Ransford,B., Clark, S. S., Defend, B., Morgan, W., FU, K., Kohno, T., and Maisel, W. H. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *IEEE Symposium on Security and Privacy* (2008).

[12] Kennedy, D., & Simon, R. (2011). Pentesting over Power lines. Defcon 2011

[13] A. Kiėzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, Automatic creation of SQL injection and cross-site scripting attacks, in ICSE'09, Proceedings of the 30th International Conference on Software Engineering, Vancouver, BC, Canada, May 20-22, 2009.

[14] Klein, A. Cross Site Scripting Explained, June 2002 from Sanctum Security Group. Retrieved June 29, 2013 from Stanford University: http://crypto.stanford.edu/cs155/papers/CSS.pdf

[15] Learn About CFLs, (N.D.), from Energy Star. Retrieved June 26, 2013 from the Energy Star: http://www.energystar.gov/index.cfm?c=cfls.pr_cfls_about

[16] OFPC Safety Alert, Compact Fluorescent Light Bulbs (CFL's) February 9, 2011 from the New York State Division of Homeland Security and Emergency Services. Retrieved July 6, 2013 from New York State New York State Division of Homeland Security and Emergency Services: http://www.dhses.ny.gov/ofpc/news/press/documents/2011_safety_alert_cfl_actual.pdf

[17] Recalls, October 5, 2010, from the United States Consumer Product Safety Commission. Retrieved July 6, 2013 from the United States Consumer Product Safety Commission: http://www.cpsc.gov/en/Recalls/2011/Trisonic-Compact-Fluorescent-Light-Bulbs-Recalled-Due-To-Fire-Hazard/

[18] Rocznik, K., Consumerwatch: CFL Bulb Safety, March 25, 2013, from CTV News. Retrieved March 25, 2013 from CTV News: http://winnipeg.ctvnews.ca/consumerwatch-cfl-bulb-safety-1.1210152

[19] Smart Homes and Home Automation, July 2011, from Berg Insight. Retrieved July 1, 2012 from Berg Insight: http://www.berginsight.com/ReportPDF/ProductSheet/bi-sh1-ps.pdf

[20] Spradlin K., Blaze Underscores need for CFL bulb Education, April 30, 2008, from Cumberland Times-News. Retrieved June 26, 2013 from Cumberland Times-News: http://times-news.com/archive/x1540421978

[21] Wright, J. (2011). Practical ZigBee Exploitation Framework. Toorcon 2011.

Web Adoption: An Attempt Toward Classifying Risky Internet Web Browsing Behavior

Alexander D. Kent
Los Alamos National Laboratory

Lorie M. Liebrock
New Mexico Institute of Mining and Technology

Joshua Neil
Los Alamos National Laboratory

Abstract

- **Background.** This paper explores associations of computer compromise events in relationship to web browsing activity over a population of computers.
- **Aim.** Our hypothesis was that computers are more likely to be compromised in comparison to other computers when the computer regularly browses to web sites prior to other computers visiting the same site (early adopters) or browses to unique web sites that no other computer visited (unique adopters) in a given time period.
- **Method.** Web proxy data and associated computer-specific compromise events covering 24,000+ computers in a contiguous 6 month time period were used to group computers in various adopter categories and compare potential compromise events between the groups.
- **Results.** We found distinction in web surfing behavior, in some cases differentiating the chance of compromise from 2.5-fold to over 418-fold between certain adopter categories. However, the study also showed no additional value in predicting compromise using these more complex adopter categories when compared to using simple unique web activity counts. As additional contributions, we have characterized several large, real-work cyber defense relevant data sets and introduced a method for simplifying web URLS (client web requests) that reduces unwanted uniqueness from dynamic content while preserving key characteristics.
- **Conclusions.** We found that a count of unique web visits over time has the same level of predictive power for potential compromise as does the more complicated web adopter model. Both models have better than chance levels of prediction but also reinforce the idea that many factors beyond elements of web browsing activity are associated with computer compromise events. Nonetheless, our adopter model may still have value in objective computer risk determination based on web browsing behavior.

1 Introduction

Using a web browser application on a computer to query and fetch information from the Internet is a primary and regular activity for many computers within an organization. Unfortunately, web browsing is also a primary vector for a computer to become compromised by malicious entities. According to Symantec's 2011 Internet Threat Report, web-based attacks have increased by 36% compared to 2010 with over 4,500 new attacks every day; additionally, 39% of all email-based attacks used a web link within the email as the malicious vector [26]. Given this concerning situation, methods for differentiating web browsing behavior across an organization that is benign versus that which increases the risk of computer compromise is of particular relevance and use.

The focus of the work presented in this paper is on improving the security of all computers as a comprehensive system within an organization. We assume that compromises, originating from the Internet through a variety of mechanisms, occur at a low but continuous rate for all significantly-sized organizations. In addition, we assume that the contemporary organization's goal for cyber security is to rapidly detect and then reduce the frequency and damage caused by these Internet-originating compromise events. Note that this assumption is different than the traditional philosophy of the cyber Maginot Line with maximization of the perimeter and assumption that any breach is failure. The work we present in this paper is about learning from ongoing compromise-oriented activity across a large, coherent population of computers to manage and minimize future malicious activity to the aggregate population. It is not about how to improve the protection of individual computers in isolation.

Our work demonstrates that risk of compromise is not

uniform across a set of computers (representing users). We find that, indeed, there are patterns of risk when computers exhibit certain web browsing behavior over time. We also find that a simple quantity of web activity (the number of unique web locations visited) may have an equivalent association and provide the same predictive capability as the more complicated adopter models we explore below.

In this paper we begin with an overview of existing related and relevant research. We then present a model for describing web access behavior across a population of computers. Next we define a variety of data sources used for validating this model including some useful characteristics in large-scale web browsing traffic and relevant compromise indicating data sets. We also propose data reduction and normalization processes to help allow better comparison and analysis. These data sets are then used to explore the model. We conclude the paper with a discussion of applications for our models, shortcoming of our approach, and future directions for using the data and outcomes of this study.

2 Related Work

A variety of work has been done to characterize web browsing behavior and activity. Many focus on various means of content and search classification. Kumar et al. provide a large-scale study of web browsing content classification using a one week data sample collected from browser toolbar searches [18]. A variety of others look at content classification, general browsing patterns and content valuation [1, 2, 19]. Another area of research has been on re-visitation of web content and understanding how often and perhaps why users are returning to the same or similar content [14, 5, 27]. All of this research focuses on single users and not necessarily information spread between users. Most of the research, to varying levels, found significant re-visitation of web locations and content by users. No research was found that explicitly examines a single large-scale organization over many months in terms of web browsing behavior quantification. We note that our behavior modeling is focused on the activity external to the computer and not traditional on-computer behavior modeling seen in existing research [11].

Additionally, a variety of research exists in web-based compromises, methods, and understanding. Provos et al. provide a well cited, comprehensive overview of recent web attack methods and the significant volumes of malware seen on the web [24]. Their results on over 4.5 million URLs showed 10% as malicious. A slightly older study by Mushchuck et al. show similar results over a smaller set of URLs; 13.4% of web downloaded executable content being malicious and 5.9% of dynamic (script) web content being malicious [22]. Another study by Provos et al. shows that 1.3% of Google searches returned at least one URL result that was malicious [23]. Note that this malicious content is not hosted by Google's site but is instead on the sites presented by Google in the research results.

The work by Moore et al. provides some interesting and relevant time frames for how long web sites serving malicious content exist before take-down events occur [21]. Their study shows that phishing-oriented web sites existed for an average of 58 hours before take-down but had long-lived sites as well, producing a lognormal distribution and a median of just 20 hours. They also imply the difficulties of just blocking and blacklisting web locations as a solution to defending against malicious web content given the variable nature and ease at which web locations are changed.

Ma et al. propose a machine learning approach to determining malicious web content solely through the URLs and a variety of associated, non-content attributes (WHOIS and similar data) [20]. Using a sample of approximately 30,000 known benign and malicious websites (URLs) from several sources, they showed a 95-99% accuracy in determining malicious content. Invernizzi et al. demonstrate an efficient approach for determining and finding malicious web content throughout the Internet using attributes from existing malicious web locations to help narrow the search [15].

Hein et al. provide an overview of contemporary attacks against web browsers and mitigation strategies [13]. Their paper describes several mechanisms of how exploits are injected into web browser clients without direct user knowledge and both infrastructure and browser improvement mitigation techniques. They end the paper by proposing a crowd-sourced trust model that allows web browsers to determine potential harm of content based on others' prior experience. A variety of novel approaches to the traditional detection of malware delivered through the web continue to be developed [4, 17]. Grier et al. show an interesting ability to determine the root (actual) source of malicious content that attempted to compromise a web browser as a function of their proposed browser [12]. Davis et. al demonstrate the use of time series data to present web access volumes before and after publicized cyber incidents to determine the effects of incidents on activity by the public to the web site [7].

Few validated methods for objectively determining risk with a computer based on activity or behavior exist. In contrast, existing research focuses on the behavior of computer attackers themselves and not on the recipient's perspective [6]. Unfortunately, quantifiable methods of validation are lacking in much of the related, existing research [28]. An objective, data-driven approach to de-

termining risky behavior is of particular importance to large-scale cyber defense [10]. We believe this approach is a key aspect of the model presented and discussed below.

3 Approach

In this section, we introduce our model for describing web access behavior across a population of computers. It assumes a source of historical web access logs representing the population of computers. Internet-accessing web proxy logs are a common source of such data for large organizations.

Our web adopter model (WAM) hypothesis is inspired by Rogers' sociological model of technology early adopters and the repetitive mechanisms through which new technologies expand gradually to a larger group of adopters [25]. This technology adoption model can be succinctly described as follows: When a new technology becomes available, a subset of the populace (risk-taking and technology-centric individuals) quickly adopt the technology. With increasing propagation, a larger set of individuals follow in adoption, but with an assumed lower risk (since any problems were worked out by the earlier adopters). This process and the role individuals play within the process is generally static from one related technology to the next in terms of propagation distribution, speed, and path.

When applied to Internet web browsing behavior, we find that indeed there are well-defined patterns when computers (representing users) adopt specific web locations over time. We find this behavior, in combination with the count of unique web locations accessed by the computer, does have statistical power for predicting the risk of compromise. In particular, when we look at specific subclasses of compromises, there is a strong association between web adopter behavior and probability of compromise.

WAM specifically distinguishes three classes of web adopter behavior that we show associate well to risky and non-risky behavior. The first adopter type we refer to as unique adopter (UA) behavior. The UA behavior applies to computer access events to Internet web locations that are unique accesses within the computer population and time frame considered. The second type we call early adopter (EA) behavior, which are those computers accessing web locations in a well-defined time period before other computers within the population also visit the web location. The final type we define as mainstream adopter (MA) behavior. MA behavior occurs when a computer accesses a web location that is common in the computer population and it would be impossible to distinguish EA behavior; for example, much of the computer population visits `http://google.com`. These behavior classifications are applied to each unique web access that a computer makes over a time period and drives overall labeling of the computer's browsing behavior.

We now describe WAM more formally.

Given a set of all unique web locations W (Section 4.1 defines web locations), over a time period T, each unique web location χ is associated with an unevenly spaced time series (USTS) [9] of access events. The USTS has N elements over time period T beginning at T_{start} and ending at T_{end}.This USTS is a sequence of value pairs of the first access time by a computer and the identifier (name) of the computer (t,C):[1]

$$W_\chi = (t_1,C_1),...,(t_N,C_N)$$
$$\text{s.t. } T_{start} \le t_1 \le t_2 \le ... \le t_N \le T_{end}.$$

Note that any subsequent accesses by an individual computer C to the same web location χ are not within the USTS W_χ; only the first access by that computer to the web location are included.

We define a single element set containing computer C as an UA set, $\{(C_1,...,C_n)\}$, to web location χ when C was the only computer to access the location in the time period T:

$$UA_\chi = \{C : W_\chi = \{(t,C)\}\}.$$

Similarly, we define a set of computers $\{C_1,...,C_n\}$ as an EA set to web location χ when set members are the first to access χ in the time period T; at least three computers accessed χ; and the EA set accessed χ at least 24 hours prior to at least one additional computer accessing χ (not in the EA set):

$$EA_\chi = \{C_i : (t_i,C_i) \in W_\chi,$$
$$|W_\chi| \ge 3,$$
$$\exists (t_k,C_k) \in W_\chi \text{ s.t. } t_i + 24 \text{ hours } \le t_k\}.$$

The 24 hour time separation between early adopters and non-early adopters (visitors) is based on observation of the data set. Figure 1 shows the distribution of time between the early adopter(s) and subsequent adopters. Note the well-defined time steps between the set of early adopter(s) and the subsequent computers that access the web location. Specifically, we find that the time separation generally falls on well-defined time boundaries of 1 day (24 hour) increments. Our speculation is that this reflects human behavior and the propagation of information about the web location from the early adopter population to the subsequent adopters. It provides a useful boundary between early adopters and the rest of the adopting population of a web location. In addition, it

[1] Most cyber security relevant data sets are at one second resolution and events within the same second are randomly ordered within that second, otherwise the sequence is strictly ordered in time.

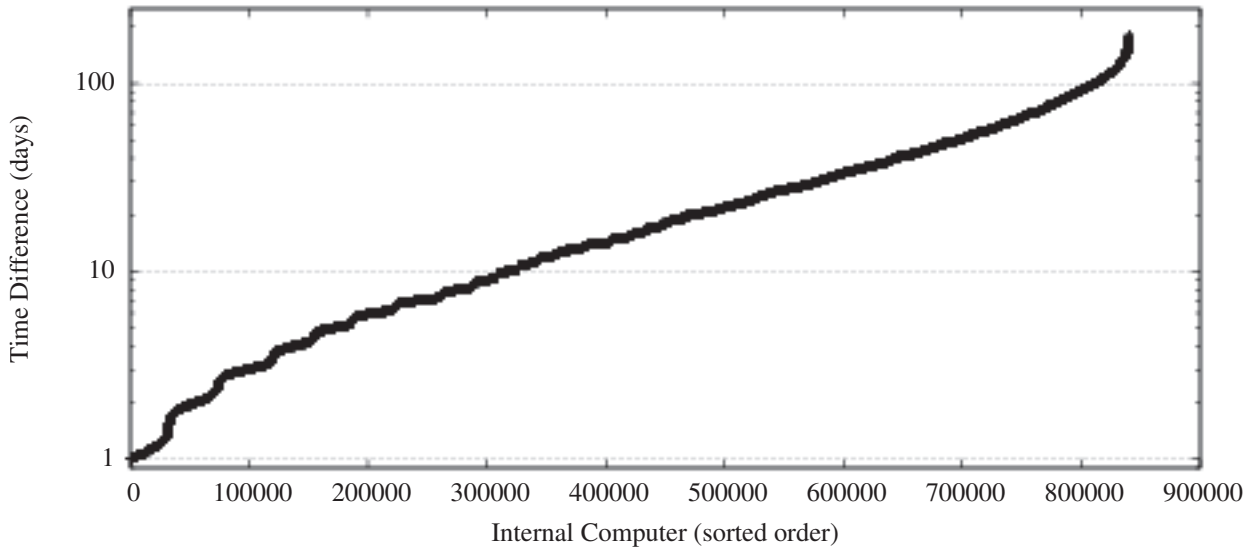

Figure 1: The empirical distribution of the time difference in days between the set of EA computers EA_χ and the additional adopters (computers who accessed it) of all web locations χ in our web data set. Note the step at well-defined boundaries of day intervals. Note the graph is log scale on the y-axis.

was important to allow for multiple early adopters, as can obviously occur. A good example of multiple early adopters would occur with a phishing email containing a web link that multiple receiving users then clicked soon after receiving it.

The top 1 percentile of web locations by number of unique accesses represents over 90% of all successful web access events, as detailed in Section 4.1. While the top percentile was chosen subjectively, it provides a useful differentiator for particularly common locations like `google.com` versus the rest of the web locations. We define ℓ, the set of lengths for all W_χ (number of unique accesses for all χ):

$$\mathscr{L} = \{|W_\chi| \forall \chi\}.$$

Let $p_{99}(\mathscr{L})$ be the 99th percentile of $\mathscr{L}$. We now define Λ as the set of web locations χ that are in the 99th percentile or above in terms of unique access length:

$$\Lambda = \{\chi : |W_\chi| \geq p_{99}(\mathscr{L})\}.$$

Finally, we define a set of computer $\{C_1, ..., C_n\}$ as a MA set to web location χ when the web location is in the set of most accessed web locations Λ:

$$MA_\chi = \{C : (t, C) \in W_\chi, \chi \in \Lambda\}$$

When applied to Internet web browsing behavior, we find that indeed there are well-defined patterns of web location adopter behavior for specific unique requests over time. In addition, we find web adopter behavior does have an association to riskier behavior relating to compromise in our data sets. In particular, when we look at specific subclasses of indicators of compromise (IOC),[2] there is a strong association between adopter behavior and probability of compromise. The results of the model in conjunction with IOC events are discussed in Section 5.

4 Data

Several sources of data were collected and analyzed for the purposes of validating and analyzing WAM. The data sets collected are from Los Alamos National Laboratory's (LANL's) organizational user networks over a period of 6 months during 2011 and uses activity data collected from 5 primary sources:

- *Web locations:* 6.4 billion outbound web proxy log entries representing Internet web requests from 24,292 computers.
- *Antivirus:* 306,135 antivirus log entries from approximately 18,000 Microsoft Windows-based computers.[3]

[2] An IOC is often also referred to as an intrusion detection or compromise *signature* in much of the related research.

[3] The count is approximate due to computers only reporting *events* and an exact inventory of computers with the antivirus reporting agent was not available. The additional 6242 or so computers seen in the web proxy logs are non-Windows computers or Windows computers with custom configurations that do not report antivirus data.

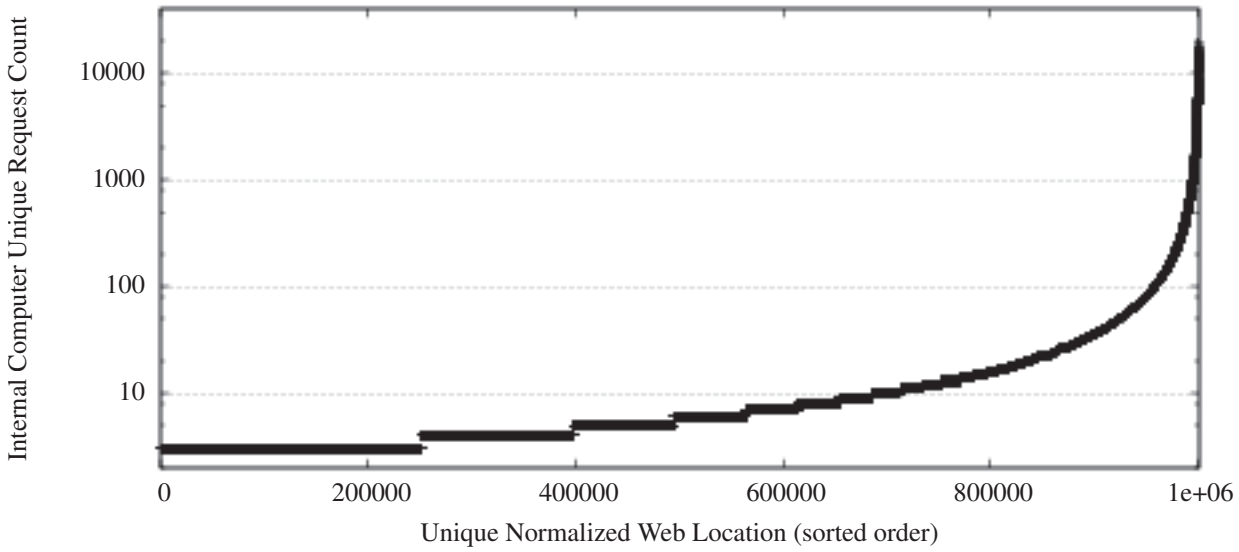

Figure 2: The empirical distribution by number of unique client computer accesses to unique normalized web locations over the 6 month time period. Only data for the 1,000,664 web locations that had three or more unique client visits is shown. The top 1% of web locations are still included in this representation. Note the graph is log scale on the y-axis.

- *IR:* 1256 Internet-intrusion incident response analyst tickets from LANL's incident response capability.
- *Phishing* 44,550 normalized web locations known for serving phishing-based malware derived from public Internet sources.
- *Proximity:* 19 normalized web locations derived from time proximity across two or more computers with existing and time-relevant IOC events.

More specific dates of the data analyzed are not disclosed to reduce the likelihood that an adversary could use the information presented in this paper for inappropriate purposes.

4.1 Web Location Access Events

For the first data set, the outbound HTTP proxy logs were analyzed. Using only successful GET and POST requests (and excluding CONNECT and others), this accounts for 6.4 billion individual web page request records over the 6 months from the 24,292 LANL computers. However, many of these requests are unnecessarily specific and redundant so we therefore normalize them.

A normalized HTTP web request has a somewhat complex definition as used in this paper. Most simply stated, it is a client computer's first successful request for a given file extension or file type from the base domain of the source server. This simplification allows us to condense the web request events that are overly unique due to load balancing servers or dynamically generated client-specific paths and file names; and to reduce the variety of file types possible (e.g. `x-pdf` and `pdf` resolve to the same type), but still retain some distinction of different file types coming from a web domain.

More specifically, the normalized web request uniform request locator (URL) is substantially shortened to include just the base 2-tuple of the server's domain (or 3-tuple in the case of two letter country code suffix or first two octets of the IP address if no domain). The URL's path is then replaced with just the file extension or MIME type if there is no file extension. Given these changes, the normalized web request becomes: `http://aaa.com/pdf` or `http://aaa.com.au/html` (we call these "web locations" and define them individually as χ). When reduced, this accounts for 3,942,541 normalized unique web locations (site and type pairs).

Borders et al. provide an intriguing, though more complex, method of dealing with the dynamic URL's generated by dynamic web content to enable site comparison and analysis [3]. Unfortunately, their method also requires elements of content beyond just the URL. Their intended use was also quite different than ours.

Even with this normalization, the breadth of uniqueness of the web sites is noteworthy: 2,389,586 (60.6%) of the normalized sites during 6 months are unique (only one client computer accesses it); 552,291 have two different client's requesting them; and 1,000,664 with three or more. We believe this high uniqueness is primarily the result of two factors: load balancing of server con-

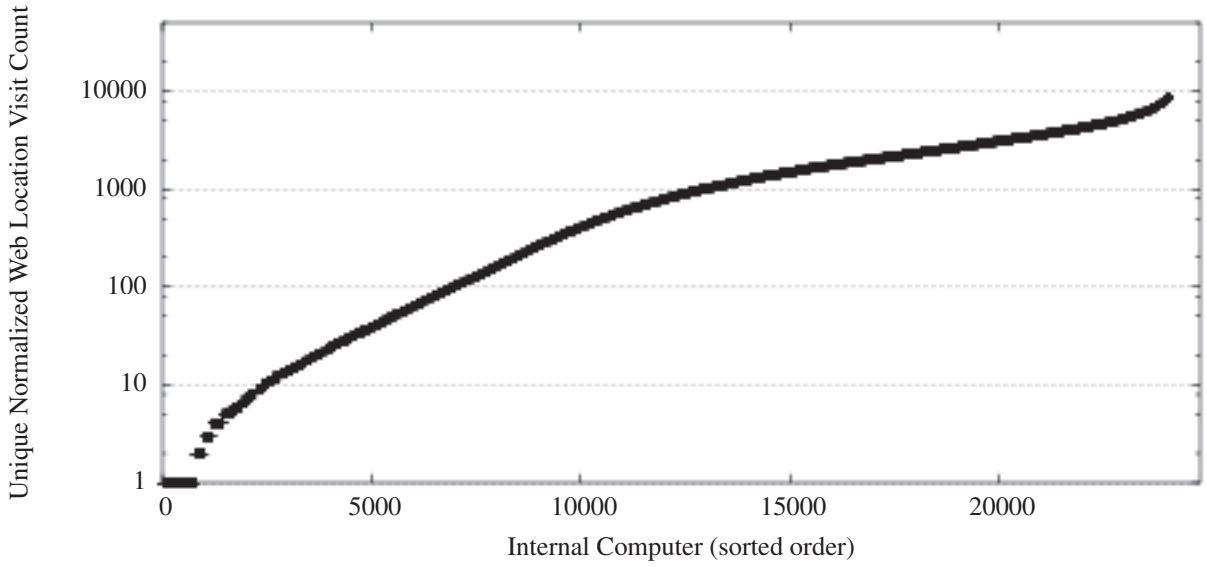

Figure 3: The empirical distribution of the 24,292 internal computers and the number of unique normalized web locations each has accessed over the 6 month time period. Note the graph is log scale on the y-axis.

tent and dynamic locations generated by dynamic content. Our normalization process attempts to combine location-related content, while still allowing some distinction through file type differences. Figure 2 shows the empirical distribution of clients visiting normalized web locations with 3 or more clients.

On the other end of the distribution, a vast majority of requests are made to a relative minority of unique locations on the Internet. For example, requesting the web location `http://google.com/html` occurs from 19,800 of the 24,292 total computers during the 6 months.[4] We find that the top 1 percent of web locations (39,424 web locations) in the month's traffic have 108 or more requests from distinct client computers. We make the assumption that this top 1% represents the most popular information on the Internet for LANL computers. In fact, we find that this set of web locations represents over 90% of the total web access traffic in the 6 months. While compromise is not impossible from these top locations, it is improbable and when it does occur, it is quickly and publicly mitigated. Thus for the purposes of this work, we exclude the data from these top 39,424 normalized web locations. Normalization and removal of these web locations reduces our data set to 3,903,117 unique web locations.

Figure 3 shows the empirical distribution of computers to the number of unique normalized web locations each accessed over the 6 months. Observe the extremely high number of unique web requests that exist for some web locations and the value in removing these few extremes for comparison purposes across the rest of the web locations. The average number of web locations that a computer accessed in the 6 months was 1565 with a standard deviation of 2043. The minimum was 1 web location and the maximum was 36,896. The median number of web locations accessed was 91.

4.2 Compromise Events

As previously stated, compromise data over the 6 month time period comes from two sources: the individual antivirus logging of approximately 18,000 Microsoft Windows-based computers and the incident response (IR) tickets for intrusions from LANL's incident response capability. During the 6 month time period, 848 computers, in 306,135 individual (and often repetitive) events, reported having the local antivirus engine detect malware. From analysts, 1256 unique IOC ticket events involving 401 computers were recorded during the time period.

In terms of compromise and intrusion detection, most methods of detection are an *existence* IOC that does not contain information regarding source or method of compromise. For example: an antivirus engine can detect that a piece of malware is on a computer as part of a nightly file check, but would not indicate how the malware came to be on the computer. Likewise, incident response tickets will indicate that a computer has been found to contain malware (perhaps it was unexpectedly

[4] In terms of all requests (not just the first one by a given computer), `http://google.com/html` represents 7,124,661 or 1.5% of all web traffic over the 6 months.

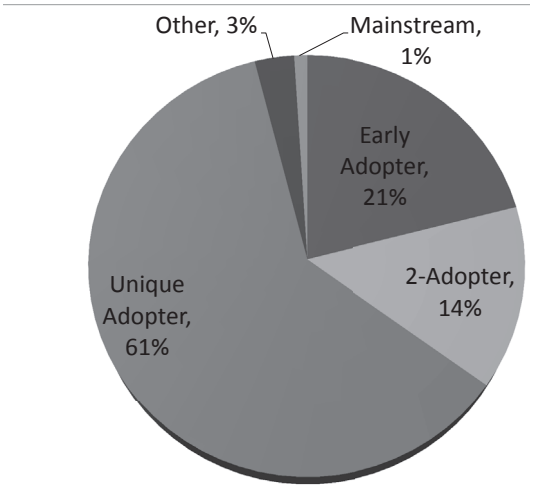

Figure 4: Chart comparing the unique normalized web locations in the 6 months (T) by type. Unique adopters are those web locations with exactly one client computer visiting during the 6 months where $|W_\chi| = 1$. 2-adopters are those locations with exactly two unique client computers, $|W_\chi| = 2$. Early adopters are web locations χ are those locations where $|EA_\chi| \geq 1$. Mainstream are web locations χ that account for the top 1% of unique visits by count where $|MA_\chi| \geq 1$. Finally, those sites that have 3 or more visitors but do not meet the conditions as an EA or MA containing location are labeled as Other.

beaconing to the Internet), but again does not indicate how the malware came to be resident on the computer. IOCs also contain an inherent level of error and more accurately they are indicators of *potential* compromise with often high and unquantified levels of false indication.

False positives likely do exist at some level within the two IOC data sets due to the inaccuracies of antivirus engines and IR investigations that did not associate to an actual intrusion. In addition, an unquantifiable number of malware events likely also exist within the 6 months that are not represented within these two data sets (false negatives). Even with these two limitations, we feel that this compromise data is particularly valuable and unique as a data source given its association with actual cyber intrusion events, as shown in the following results.

4.3 Internet-Published Phishing Websites

Publicly available malware-serving website lists for phishing attacks were used to generate another data set of potential IOC events for relational purposes. 12,863,857 and 1,447,310 unique, malware-serving URLs were retrieved from `malwaredomainlist.com` and `phishtank.com`, respectively on September 5, 2012. These URLs were then reduced to 44,550 unique normalized web locations using the same method described above for web proxy data. None of these phishing-based malware web locations matched the top 1% of web sites previously described (mainstream). The substantial reduction in URLs points to the significant reuse of base sites for serving malware. Obviously, such reduction does introduce the risk of false positive matching in this data set, though for the purposes of our study we believe it does not significantly impact the results. Of the 44,550 web locations, 4411 were seen within the 6 months of LANL's web request data. Of course, accessing a web location known to contain phishing-related malware does not mean guaranteed compromise but it is definitely risky behavior and we consider it to be a *potential* IOC.

4.4 Time Proximity to Existing IOC Events

The final IOC-related data set uses the web location USTS sets to determine IOC events of interest. This approach combines the time line of successful website accesses (or downloads) for each client computer with the time line of IOC events that also occurred for the computer. It then considers suspect any website accesses within a 24 hour time period before or after an IOC event. If two or more computers consider a website access suspect due to time proximity with an IOC, the website access is considered a *potential IOC*. In our data set, when the website access is seen by 3 or more computers in association with an IOC, we find no false positives–the website location has been found to always be a source of malicious activity. Additional details and the results of this IOC detection method can be found in [16].

5 Results

Using WAM, as described in Section 3 and the data sources described in Section 4, we now discuss results.

To provide a sense of how the adopter types are distributed, Figure 4 shows the ratios of each adopter type by web locations in the normalized web access data set over the 6 months. As expected, the largest volume of web locations are associated with UA behavior followed by web locations that have distinct EA computers. Note that the 2-adopters in the figure are those web locations χ where $|W_\chi| = 2$ and can have neither UA or EA associated computers, per the definitions. Similarly, "Other" applies to those web locations that do not have EA or MA sets, e.g. there are multiple computers accessing it but none initially more then 24 hours apart or enough to make it mainstream.

To apply WAM, we take the set of web locations χ that each computer C accessed from the normalized web

access data set (W) over the 6 months, defined as:

$$W_C = \{\chi : \chi \in W, C \in W_\chi\}.$$

For each web location χ in W_C, we apply an label based on computer C's membership in one of the various sets UA_χ, EA_χ, or MA_χ. For each computer C, all web locations χ in W accessed during T (the 6 months for our data set) are potentially labeled as UA, EA, or MA respectively.[5] The ratios of each of these labels to the total web location accesses ($|W_C|$) for each computer are then computed. Figure 5 shows the ratio for each of these labels across the population of computers within the data set.

Using the four different types of IOC data described in Section 4, we then label a given computer as being *compromised*[6] by one or more of those IOC types if the computer is associated with the IOC type during the 6 months. The four IOC labels, as previously discussed, are *Antivirus (AV)*, *IR*, *Phishing*, and *Proximity*. While this is a very coarse labeling of compromise over a potentially long period of time, we find that the model yields interesting results and suspect that more fine grained labeling would improve the model's fidelity and usefulness (with significantly higher data and computation costs).

When we consider the set of computers exhibiting any of these four types of IOC events, we find that the ratios of adopters are rare at the low and high endpoints, as seen in Figure 6. The lack of low ratios is easily explained by the notion that computers that do not access unique web locations or go to locations as the first set of visitors (when exploits may more likely exist) are much less likely to be compromised. While the result is more difficult explain, we see three explanations for why high adopter ratios would exist without associated IOC events:

- High MA ratio computers are very likely to have few to no compromise events since they only access very well known and often accessed locations on the Internet; sites we have previously asserted are not associated with compromise events.
- For a small set of computers with high UA and EA ratios, we find they are crawling many, diverse web locations in an intended automated fashion.
- Somewhat more speculatively, it may be that computers with very high levels of UA and EA activity are representing users with a higher level of knowledge in avoiding compromise. This idea requires further exploration to assert or reject.

[5] Again, note that some locations χ are not labeled since they do not meet the definitions for the 3 adopter types.

[6] We assume the logged IOC was correct and not a false positive for our statistical purposes.

When a computer's ratio of traffic for UA and/or EA exceeds 1%, we label that computer as being of generally type UA and/or EA. We choose the somewhat arbitrary 1% cutoff based on the significant increase that occurs approximately at this value, as seen in Figure 5. Similarly, we used a 99% cutoff for a computer to be labeled as a mainstream adopter; 99% or more of its accesses to unique web locations χ must be labeled as mainstream accesses. Using these computer-based labels and the association of computers to the various IOC events described in Section 4, Figure 7 shows the overall results of applying WAM using our data sets. The strong association to potential IOCs relating to computers visiting phishing labeled web locations is particularly noticeable. We assert that phishing and proximity data are most strongly associated adopter behavior due to their pure web-based association. In contrast, antivirus and IR events can occur through other compromise mechanisms that are not associated with web browsing activity. In addition, antivirus data is only collected from a subset of Windows-based computers compared with the larger set of web browsing computers (18,000 versus 24,000). Nonetheless, there is still significant association between all of the IOC types and computers labeled as UA and EA.

Using these results, we can now estimate probabilities of an IOC of various types occurring, given a computer C has one or more of these labels, We define the estimated probability for an adopter label L as:

$$\hat{P}(L) = \frac{|L(C)_\gamma|}{|L(C)|},$$

where $L(C)_\gamma$ is the set of computers $\{C\}$ with adopter label L associated with a potential IOC γ during time period T and $L(C)$ is the set of computers with adopter label L. These probabilities for both having and not having the associated labels can be seen in Figure 8. Note the particularly high probability of having a phishing IOC associated with a computer that is both an unique and early adopter. Also note the extremely low probability for computers label as mainstream adopters for any type of IOC–these computers are obviously not the source of most compromise from the Internet. The $UA \vee EA$ results are not shown due to insignificant differences in the values and brevity.

One particularly important question about how useful these adopter types are for predictive methods is whether these labels are leading or trailing indicators for the different compromise events. We used a coarse method of dividing the data sets in to two 3 month pieces and assigned adopter type labels and IOC labels to each computer independently within the 3 month data sets. Thus, if a computer had an adopter label in the first 3 months that drove to an IOC in the second 3 months, we assume

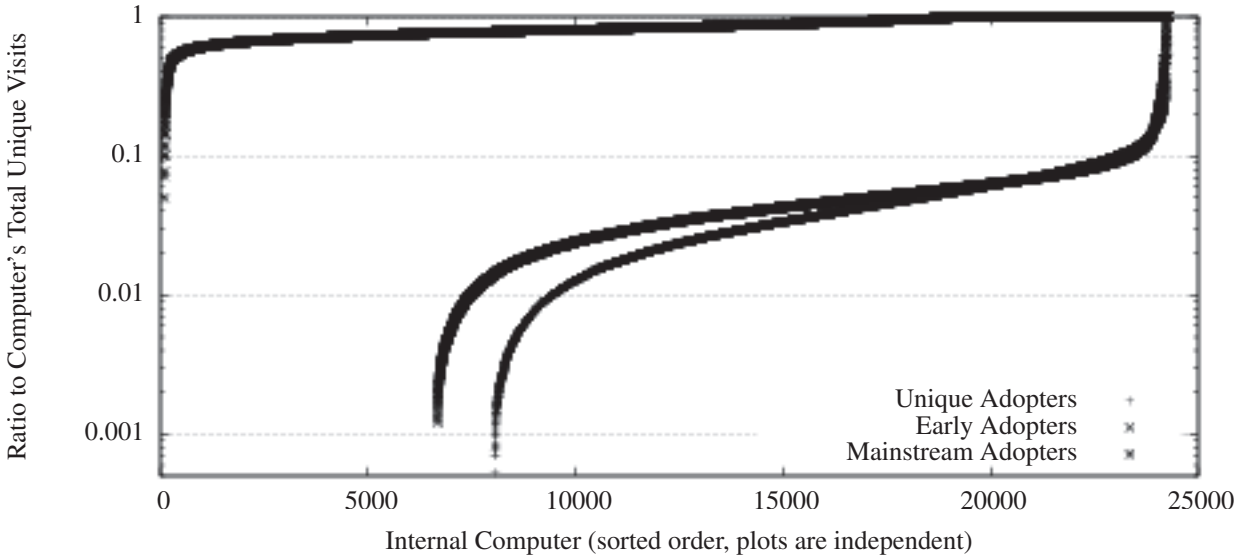

Figure 5: The empirical distribution of the 24,292 internal computers in terms of the ratio of their individual total unique web location visits over the 6 months as UA, EA, and MA. Note the graph is log scale on the y-axis.

the adopter label is a *leading indicator*. Inversely, if an IOC occurred in the first 3 months that led to an adopter label in the second 3 months, we assuming the adopter label was a *trailing indicator*. If the adopter label and IOC both occurred in the same three month period, we call that a (time) *local indicator*. These indicators are not mutually exclusive when applied to a computer's activity over the 6 months. For example, a computer could be a leading, trailing, and local indicator if the same adopter label and the same type of IOC event occurred in both 3 month time periods.

Figure 9 shows the volume of computers showing a leading, trailing, and/or local indicator for each adopter label and data set. As is shown, the adopter labels show association for the antivirus and phishing data sets. However, the labels for the IR data set are seen more strongly as a trailing indicator. The high volume of UA and EA labeled computers that show association within time local IOC events shows the strong association to UA and EA behavior and similar-in-time risky behavior. In contrast, the lower volume of time local IOC events to MA behavior reconfirms the relatively low risk that MA labeled computers show. This low volume is particularly apparent for phishing IOC's since, as previously discussed, the phishing IOC data set does not contain any mainstream web locations. The combined $UA \wedge EA$ and $UA \vee EA$ sets were not shown due to insignificant differences in the results and for brevity of presentation.

While this initial analysis on leading and trailing indicators lacks truly useful granularity, we believe our results provide some generalized differentiation.

5.1 Application

These differences in distinct risk groups that are represented in the data can be used to help prioritize and localize the placement of traditional intrusion detection sensors and the sensitivity of these sensors. Placing sensors at or near MA labeled computers has little value but placing more sensors near computers that are both labeled as unique and early adopters has much higher value than otherwise random placement. More specifically, non-MA labeled computers have a 10-fold increase in being associated with any of the IOCs and for the phishing IOC its a 417-fold increase.

5.2 Regression Model for Prediction

We also tested the significance of the overall model for statistical fit and predictive viability using logistic regression. We tested WAM for each computer within the data set at predicting IOC events against both an intercept (the null hypothesis that WAM has no effect on compromise) and using the total unique web location counts. For the adopter parameters we used the ratio of each adopter type for each computer to the total unique web location visits for the computer (we did not just use the simplified adopter labels discussed previously). The adopter parameters UA, EA, and MA fit the regression model while other parameters were redundant. Using these three parameters we find that WAM does much better then the intercept. However, when we use just total unique web location counters, we find equivalent predictive power indicating that this simpler counter has significant and equiv-

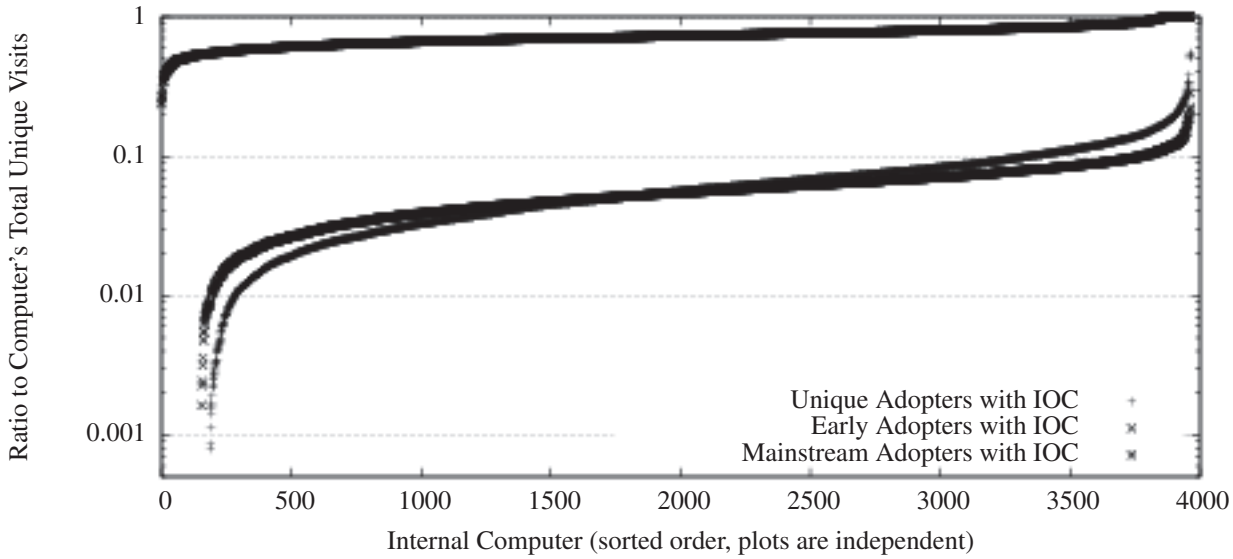

Figure 6: The empirical distribution of the 3964 internal computers that had any IOC during the 6 months in terms of the ratio of their individual total unique web location visits as UA, EA, and MA. In comparison to the total population in Figure 5, note the reduction in computers with no UA or EA ratios and the reduction computers with extremely high UA, EA, and MA ratios. Note the graph is log scale on the y-axis.

Type	**Population**	*UA*	*EA*	$UA \vee EA$	$UA \wedge EA$	*MA*
Pop.	24,292	14,825 (61.0%)	16,519 (68.0%)	17,453 (71.9%)	13,891 (57.2%)	5213 (21.5%)
AV	848 (3.49%)	685 (80.8%)	726 (85.6%)	747 (88.1%)	664 (78.3%)	63 (7.43%)
IR	401 (1.65%)	323 (80.6%)	337 (84.0%)	346 (86.3%)	314 (78.3%)	40 (9.98%)
Phish	3032 (12.5%)	2975 (98.1%)	2988 (98.6%)	3015 (99.4%)	2948 (97.2%)	2 (0.0662%)
Prox.	19 (0.0782%)	19 (100%)	19 (100%)	19 (100%)	19 (100%)	0 (0.0%)
Any	3964 (16.3%)	3681 (92.9%)	3746 (94.5%)	3801 (95.9%)	3626 (91.5%)	104 (2.62%)

Figure 7: Summary of each of the analyzed data sets and set memberships. For each of the IOC types, the number computers (and percentage of total IOC-tagged computers) is show as being labeled as UA, EA, either, both, or MA. To be labeled as UA or EA requires 1% or more of the computer's total unique web access traffic to be as an UA or EA (respectively) to web locations. To be labeled as MA, the computer needs 99% or more of its traffic to be to mainstream defined web locations. Population (Pop.) and Proximity (Prox.) are abbreviated for formatting purposes.

alent statistical predictive capability. In fact, we see that the adopter parameters, in combination, *represent* unique web location visits and individually delineated do not increase predictive capability. In other words, a computer's UA, EA, and MA parameters together present that computer's web behavior that when reduced can be equivalently stated as the unique location visit count; at least in terms of statistical prediction.

A comparison of predictive power between the two models is shown in the ROC curve in Figure 10. As seen in the figure, both the WAM model and the quantity of unique web locations visited provide better then chance but are still far from perfect predictors. The lines are nearly identical, indicating that they likely represent the same predictive capability. Given the fact that counting unique web location visits in a time period is much easier than calculation of the WAM parameters, the obvious conclusion is that it is simpler and more appropriate to use the unique visit count as the best predictor of future compromise behavior by a computer.

6 Lessons Learned

The most significant observation and theme from the research represented in this paper is the importance of simplicity in approach, whenever possible. While it is motivating to gravitate towards more complicated approaches as *good science*, we believe that given the relative immaturity of the cyber research domain, there is significant value and importance in the simplest approaches; at least

Type	$\hat{P}(UA)$	$\hat{P}(\neg UA)$	$\hat{P}(EA)$	$\hat{P}(\neg EA)$	$\hat{P}(UA \wedge EA)$	$\hat{P}(\neg(UA \wedge EA))$	$\hat{P}(MA)$	$\hat{P}(\neg MA)$
Antivirus	4.62%	1.72%	4.40%	1.57%	4.78%	1.77%	1.21%	4.11%
IR	2.18%	0.825%	2.04%	0.824%	2.26%	0.837%	0.767%	1.89%
Phishing	20.1%	0.606%	18.1%	0.570%	21.2%	0.810%	0.038%	15.9%
Proximity	0.128%	0.0%	0.115%	0.0%	0.137%	0.0%	0.0%	0.100%
Any	24.8%	3.00%	22.7%	2.81%	26.1%	3.25%	2.00%	20.2%

Figure 8: Estimated probabilities of various compromise types existing given a computer's membership in the various defined sets (or not). Using these estimated probabilities or frequencies as a basis for predicting future potential events occurring in the various populations (sets) assumes that set membership is a viable leading indicator of risky behavior.

Adp.	IOC	Pop.	Leading	Trailing	Local
UA	AV	681	77.8%	52.6%	96.9%
UA	IR	312	30.1%	69.2%	93.3%
UA	Phish	2809	52.1%	58.4%	99.4%
EA	AV	694	78.4%	55.0%	98.3%
EA	IR	323	31.0%	70.0%	94.7%
EA	Phish	2818	52.8%	59.2%	99.5%
MA	AV	88	59.1%	36.4%	67.1%
MA	IR	44	36.4%	43.2%	77.3%
MA	Phish	47	66.0%	34.0%	2.1%

Figure 9: Trending for each adopter label (Adp.) as to how often it has a leading, trailing or time local (similar in time) association to computers observed with each IOC type. The associations are not mutually exclusive. The population (Pop.) size represents the number of computers that have that adopter label in either time period and at least one associated IOC event of the given type over the two 3 month consecutive time periods.

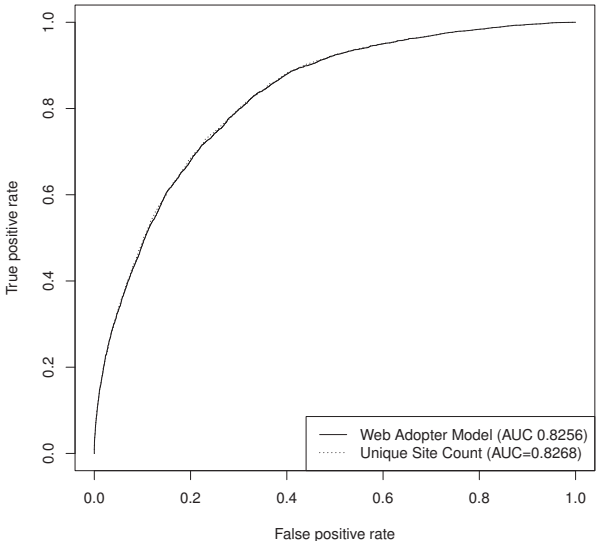

Figure 10: ROC curve showing both the fit (and likely predictive capability) of both early adopter logistic regression and simple unique web location logistic regression models. The two curves are nearly identical.

until these direct methods are *demonstrated* to be insufficient.

We assert that the regression-based prediction results we show are an excellent example of Occam's razor, where the simpler hypothesis with fewer assumptions produces equivalent results to the more complex approach [8]. We have shown that simple web counts provide equivalence to a much more complex and frankly better sounding approach.

Another key observation that was initially not expected in this research was the significant uniqueness seen within web URLs across the large population of hosts. This variation drove us to the web location approach described in Section 4.1 that allowed better URL grouping, while still enabling some diversity beyond just domain name. Without this approach, the data analysis was overwhelmed with one-event URL's that did not allow for multiple adopters except primarily in the Mainstream group. While we did try more complex methods to detect and normalize similar URLs (e.g., timestamp detection and random *ID* string detection within the URL string), we found that our simple approach provided an equivalent capability with significantly less processing and complexity. Our final approach is also much easier to describe and replicate.

Also worth noting, this paper shows the importance of using real world data in significant volume for effective hypothesis testing within the cyber domain. Using contrived data, WAM could have easily shown predictive capability stronger then the real data since such testing data is often modeled from the hypothesis itself. Obviously, the real data demonstrated a more realistic outcome.

7 Conclusion

The intended purpose of WAM was to define an objective model that delineates varying behavior according to the potential risk of compromise within a population of

computers. Though the simpler web location measure significantly reduces the overall relevance of WAM, we believe that the comparison, results, and the WAM model itself are still useful and important applied cyber security research.

While there is continued opportunity for improvement in terms of fidelity and fit, we believe the model we have presented in this paper presents a newly explored approach and at least an objective means to judge risky behavior. Even more important and rare, this paper uses a significant real-world data set to validate and quantify the model. WAM demonstrates a model for showing the association of compromise through differentiated web browsing behavior over a population of computers. While we also showed the WAM model does not provide additional predictive capability over unique web location counts, we have demonstrated the value of this simple count. Indeed, more web surfing does equate to a higher chance of compromise.

References

[1] BAYKAN, E., HENZINGER, M., MARIAN, L., AND WEBER, I. Purely URL-based topic classification. In *Proceedings of the 18th International Conference on World Wide Web* (New York, NY, USA, 2009), WWW '09, ACM, pp. 1109–1110.

[2] BILENKO, M., WHITE, R. W., RICHARDSON, M., AND MURRAY, G. C. Talking the talk vs. walking the walk: salience of information needs in querying vs. browsing. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2008), ACM, pp. 705–706.

[3] BORDERS, K., AND PRAKASH, A. Quantifying information leaks in outbound web traffic. In *Security and Privacy, 2009 IEEE Symposium on* (2009), IEEE, pp. 129–140.

[4] CANALI, D., COVA, M., VIGNA, G., AND KRUEGEL, C. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th International Conference on World Wide Web* (New York, NY, USA, 2011), ACM, pp. 197–206.

[5] COCKBURN, A., AND MCKENZIE, B. What do web users do? an empirical analysis of web use. *International Journal of Human-Computer Studies 54*, 6 (2001), 903 – 922.

[6] DANTU, R., KOLAN, P., AND CANGUSSU, J. Network risk management using attacker profiling. *Security and Communication Networks 2*, 1 (2009), 83–96.

[7] DAVIS, G., GARCIA, A., AND ZHANG, W. Empirical analysis of the effects of cyber security incidents. *Risk Analysis 29*, 9 (2009), 1304–1316.

[8] DOMINGOS, P. The role of Occam's razor in knowledge discovery. *Data mining and knowledge discovery 3*, 4 (1999), 409–425.

[9] ECKNER, A. A framework for the analysis of unevenly spaced time series data. `http://www.eckner.com/papers/unevenly_spaced_time_series_analysis.pdf`, August 2012.

[10] GEER, D., J., HOO, K., AND JAQUITH, A. Information security: Why the future belongs to the quants. *Security and Privacy, IEEE 1*, 4 (2003), 24–32.

[11] GOPALAKRISHNA, R., SPAFFORD, E., AND VITEK, J. Efficient intrusion detection using automaton inlining. In *Security and Privacy, 2005 IEEE Symposium on* (2005), IEEE, pp. 18–31.

[12] GRIER, C., TANG, S., AND KING, S. Secure web browsing with the OP web browser. In *Security and Privacy, 2008 IEEE Symposium on* (2008), IEEE, pp. 402–416.

[13] HEIN, D., MOROZOV, S., AND SAIEDIAN, H. A survey of client-side web threats and counter-threat measures. *Security and Communication Networks 5*, 5 (2012), 535–544.

[14] HERDER, E. Characterizations of user web revisit behavior. In *Workshop on Adaptivity and User Modeling in Interactive Systems ABIS05* (2005), pp. 32–37.

[15] INVERNIZZI, L., COMPARETTI, P., BENVENUTI, S., AND KRUEGEL, C. EVILSEED: a guided approach to finding malicious web pages. In *Security and Privacy, 2012 IEEE Symposium on* (2012), pp. 428–442.

[16] KENT, A. D., AND LIEBROCK, L. M. Statistical detection of malicious web sites through time proximity to existing detection events. In *Resilience Week Symposium* (2013), IEEE.

[17] KIRDA, E., KRUEGEL, C., BANKS, G., VIGNA, G., AND KEMMERER, R. Behavior-based spyware detection. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15* (Berkeley, CA, USA, 2006), USENIX Association.

[18] KUMAR, R., AND TOMKINS, A. A characterization of online browsing behavior. In *Proceedings of the 19th International Conference on World Wide Web* (2010), ACM, pp. 561–570.

[19] LIU, C., WHITE, R. W., AND DUMAIS, S. Understanding web browsing behaviors through weibull analysis of dwell time. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2010), ACM, pp. 379–386.

[20] MA, J., SAUL, L. K., SAVAGE, S., AND VOELKER, G. M. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2009), ACM, pp. 1245–1254.

[21] MOORE, T., AND CLAYTON, R. Examining the impact of website takedown on phishing. In *Proceedings of the anti-phishing workinggroup 2nd annual eCrime researchers summit* (New York, NY, USA, 2007), ACM, pp. 1–13.

[22] MOSHCHUK, A., BRAGIN, T., GRIBBLE, S., AND LEVY, H. A crawler-based study of spyware on the web. In *Proceedings of the 2006 Network and Distributed System Security Symposium* (2006), pp. 17–33.

[23] PROVOS, N., MAVROMMATIS, P., RAJAB, M. A., AND MONROSE, F. All your iFRAMEs point to Us. In *Proceedings of the 17th Conference on Security Symposium* (2008), USENIX Association, pp. 1–15.

[24] PROVOS, N., MCNAMEE, D., MAVROMMATIS, P., WANG, K., AND MODADUGU, N. The ghost in the browser analysis of web-based malware. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets* (2007), HotBots'07, USENIX Association, p. 4.

[25] ROGERS, E. *Diffusion of Innovations, Fifth Edition*. Free Press, 2003.

[26] SYMANTEC. Internet security threat report, April 2012.

[27] TEEVAN, J., ADAR, E., JONES, R., AND POTTS, M. A. S. Information re-retrieval: repeat queries in yahoo's logs. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2007), ACM, pp. 151–158.

[28] VERENDEL, V. Quantified security is a weak hypothesis: a critical survey of results and assumptions. In *Proceedings of the 2009 workshop on New security paradigms workshop* (New York, NY, USA, 2009), NSPW '09, ACM, pp. 37–50.

Dismal Code: Studying the Evolution of Security Bugs

Dimitris Mitropoulos, Vassilios Karakoidas, Panos Louridas
Department of Management Science and Technology,
Athens University of Economics and Business
dimitro@aueb.gr, bkarak@aueb.gr, louridas@aueb.gr

Georgios Gousios
Software Engineering Research Group, Delft University of Technology
G.Gousios@tudelft.nl

Diomidis Spinellis
Department of Management Science and Technology,
Athens University of Economics and Business
dds@aueb.gr

Abstract

Background. Security bugs are critical programming errors that can lead to serious vulnerabilities in software. Such bugs may allow an attacker to take over an application, steal data or prevent the application from working at all.
Aim. We used the projects stored in the Maven repository to study the characteristics of security bugs individually and in relation to other software bugs. Specifically, we studied the evolution of security bugs through time. In addition, we examined their persistence and their relationship with a) the size of the corresponding version, and b) other bug categories.
Method. We analyzed every project version of the Maven repository by using FindBugs, a popular static analysis tool. To see how security bugs evolve over time we took advantage of the repository's project history and dependency data.
Results. Our results indicate that there is no simple rule governing the number of security bugs as a project evolves. In particular, we cannot say that across projects security-related defect counts increase or decrease significantly over time. Furthermore, security bugs are not eliminated in a way that is particularly different from the other bugs. In addition, the relation of security bugs with a project's size appears to be different from the relation of the bugs coming from other categories. Finally, even if bugs seem to have similar behaviour, severe security bugs seem to be unassociated with other bug categories.
Conclusions. Our findings indicate that further research should be done to analyze the evolution of security bugs. Given the fact that our experiment included only Java projects, similar research could be done for another ecosystem. Finally, the fact that projects have their own idiosyncrasies concerning security bugs, could help us find the common characteristics of the projects where security bugs increase over time.

1 Introduction

A security bug is a programming error that introduces a potentially exploitable weakness into a computer system [34]. This weakness could lead to a security breach with unfortunate consequences in different layers, like databases, native code, applications, libraries and others. Despite the significant effort to detect and eliminate such bugs, little attention has been paid to study them in relation to software evolution [26]. One of the most common approaches to identify security bugs is *static analysis* [6]. This kind of analysis involves the inspection of the program's source or object code without executing it.

In this paper we present how we used a large software ecosystem to analyse the relationship of different types of security vulnerabilities to evolving software packages. For our research we used *FindBugs*,[1] a static analysis tool that examines bytecode to detect software bugs and has already been used in research [1, 19, 36]. Specifically, we ran FindBugs on all the project versions of all the projects that exist in the *Maven Central Repository*[2] (approximately 265GB of data—see Section 3.2). Then we observed the changes that involved the security bugs

and their characteristics.

We chose to focus our study on security bugs rather than other types of software defects. This is because compared to other bug categories, failures due to security bugs have two distinct features: they can severely affect an organization's infrastructure [33], and they can cause significant financial damage to an organization [39, 2]. Specifically, whereas a software bug can cause a software artifact to fail, a security bug can allow a malicious user to alter the execution of the entire application for his or her own gain. In this case, such bugs could give rise to a wide range of security and privacy issues, like the access of sensitive information, the destruction or modification of data, and denial of service. Moreover, security bug disclosures lead to a negative and significant change in market value for a software vendor [38]. Hence, one of the basic pursuits in every new software release should be to mitigate such bugs.

The motivation behind our work was to validate whether programmers care for the risk posed by security bugs when they release a new version of their software. In addition, we wanted to investigate other critical features associated with such vulnerabilities like the persistence of a bug; in essence, to see whether critical bugs stay unresolved for a long time. Also, we wanted to elaborate more on the relation of security bugs with other bug categories. In the same manner, we tried to examine the relationship between the size of a project release and the number of security bugs that it contains, knowing the that research has produced contradictory results on this issue [35, 28, 13]. Finally, we examined the Maven ecosystem as a whole from a security perspective. Its structure gave us the opportunity to see if a project version that is a dependency of a large number of others contains a low rate of security bugs.

In this work we:

- Analyze how security bugs found through static analysis evolve over time. To achieve this, we inspected all releases of every project. Our hypothesis is that security bugs should decrease as a project evolves, for they form critical issues, which developers should eliminate.

- Examine security bug persistence across releases. We expect that security bugs should be eliminated earlier than other bugs.

- Study the relation between security bugs and a project release's size. Our hypothesis is that security bugs are proportional to a project release's size (defined in terms of bytecode size).

- Examine the correlation of security bugs with other bug categories. Our hypothesis is that security bugs appear together with bugs that are related with performance, coding practices, and product stability.

In the rest of this paper we outline related work (Section 2), describe the processing of our data and our experiment (Section 3), present and discuss the results we obtained (Section 4), and end up with a conclusion and directions for future work (Section 5).

2 Related Work

There are numerous methods for mining software repositories in the context of software evolution [20]. In this section we focus on the ones that highlight the relationship between software bugs and evolution and try to extract useful conclusions.

Refactoring identification through software evolution is an approach used to relate refactorings with software bugs. Weißgerber et al. found that a high ratio of refactorings is usually followed by an increasing ratio of bug reports [40]. In addition, they indicated that software bugs are sometimes introduced after an incomplete refactoring [12]. Ratzinger et al. [31] showed that the number of bugs decreases, when the number of refactorings increases. Finally, Kim M. et al. [22] indicated that API-level refactorings aid bug fixes.

Micro patterns, proposed by Kim et al. [24] detect bug-prone patterns among source code. Micro patterns describe programming idioms like inheritance, data management, immutability and others. The approach involved the examination of all revisions of three open-source projects to extract bug introduction rates for each pattern. Gil et al. [11] analysed the prevalence of micro patterns across five Sun JDK versions to conclude that pattern prevalence tends to be the same in software collections.

Querying techniques are used to answer a broad range of questions regarding the evolution history of a project [17]. Bhattacharya et al. [4, 3] proposed a framework that is based on recursively enumerable languages. The framework can correlate software bugs with developers in various ways. For instance, return the list of bugs fixed by a specific developer. Fischer et al. [10] proposed an approach for populating a release history database that combines code information with bug tracking data. In this way, a developer can couple files that contain common bugs, estimate code maturity with respect to the bugs, etc. The "Ultimate Debian Database" [29] is an SQL-based framework that integrates information about the Debian project from various sources to answer queries related to software bugs and source code.

D'Ambros et al. have used *bug history analysis* to detect the critical components of a project [7]. This is done

by using an evolutionary meta-model [8]. The same approach was also used by Zimmermann et al. [42] to check the correlation of bugs with software properties like code complexity, process quality and others and to predict future properties.

The evolution of software artifacts has also been analysed to *reduce the false alarms* of the various static analysis tools. To achieve this, Spacco et al. [36] introduced *pairing* and *warning signatures*. In the former, they tried to pair sets of bugs between versions in order to find similar patterns. In the latter, they computed a signature for every bug. This signature contained elements like the name of the class where the bug was found, the method and others. Then they searched for similar signatures between versions. In their research they studied the evolution of 116 sequential builds of the Sun Java Sevelopment Kit (JDK). Their findings indicated that high priority bugs are fixed over time. To improve the precision of bug detection, Kim et al. [23] proposed a history-based warning prioritization algorithm by mining the history of bug-fixes of three different projects. Working towards the same direction, Heckman et al. [15, 14] have introduced benchmarks that use specific correlation algorithms and classification techniques to evaluate alert prioritization approaches.

Lu et al. [25] studied the *evolution of file-system code*. Specifically, they analysed the changes of Linux file-system patches to extract bug patterns and categorize bugs based on their impact. Their findings indicated that the number of file-system bugs does not die down over time. By categorizing bugs they also showed the frequency of specific bugs in specific components.

Completing the above approaches, our work focuses on the subset of security bugs. Focusing on such bugs is not a new idea. Ozment and Schechter [30] examined the code base of the OpenBSD operating system to determine whether its security is increasing over time. In particular, they measured the rate at which new code has been introduced and the rate at which defects have been reported over a 7.5 year period and fifteen releases. Even though the authors present statistical evidence of a decrease in the rate at which vulnerabilities are being reported, defects seem to appear persistent for a period of at least 2.6 years. Massacci et al. [27] observed the evolution of software defects by examining six major versions of Firefox. To achieve this they created a database schema that contained information coming from the "Mozilla Firefox-related Security Advisories" (MFSA) list,[3] Bugzilla entries and others. Their findings indicated that security bugs are persistent over time. They also showed that there are many web users that use old versions of Firefox, meaning that old attacks will continue to work. Zaman et al. [41] focused again on Firefox to study the relation of security bugs with performance bugs. This was also done by analysing the project's Bugzilla. Their research presented evidence that security bugs require more experienced developers to be fixed. In addition, they suggested that security bug fixes are more complex than the fixes of performance and other bugs. Shahzad et al. [34] analysed large sets of vulnerability data-sets to observe various features of the vulnerabilities that they considered critical. Such features were the functionality and the criticality of the defects. Their analysis included the observation of vulnerability disclosures, the behavior of hackers in releasing exploits for vulnerabilities, patching and others. In their findings they highlighted the most exploited defects and showed that the percentage of remotely exploitable vulnerabilities has gradually increased over the years. Finally, Edwards et al. [9] have recently conducted a study similar to ours in which they have considered only four projects. Their results demonstrate that the number of exploitable bugs does not always improve with each new release and that the rate of discovery of exploitable bugs begins to drop three to five years after the initial release.

3 Methodology

Our experiment involved the collection of the metric results of the FindBugs tool. Before and during the experiment, we performed a number of filters on the data coming from the Maven repository, for reasons that we will describe below.

3.1 Experiment

The goal of our experiment was to retrieve all the bugs that FindBugs reports, from all the project versions existing on the Maven repository (in the Maven repository, versions are *actual* releases). The experiment involved four entities: a number of *workers* (a custom Python script), a *task queue* mechanism (RabbitMQ—version 3.0.1), a *data repository* (MongoDB—version 2.2), and the *code repository*, which in our case it was the public Maven repository.

Maven is a build automation tool used primarily for Java projects and it is hosted by the Apache Software Foundation. It uses XML to describe the software project being built, its dependencies on other external modules, the build order, and required plug-ins. To build a software component, it dynamically downloads Java libraries and Maven plug-ins from the Maven central repository, and stores them in a local cache. The repository can be updated with new projects and also with new versions of existing projects.

First, we scanned the Maven repository for appropriate JARs and created a list that included them. We discuss the JAR selection process in the next section. With the JAR

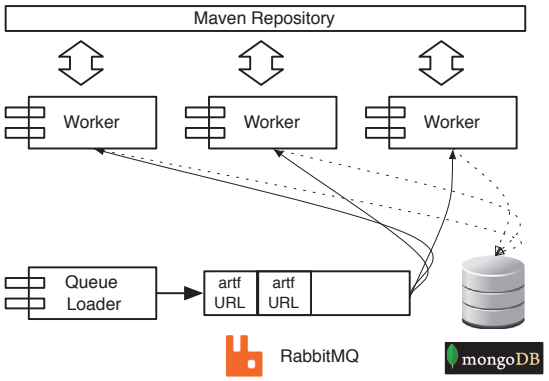

Figure 1: The data processing architecture.

Table 1: Descriptive statistics measurements for the Maven repository.

Measurement	Value
Projects	17,505
Versions (total)	115,214
Min (versions per project)	1
Max (versions per project)	338
Mean (versions per project)	6.58
Median (versions per project)	3
Range (over versions)	337
1^{st} Quartile (over versions)	1
3^{rd} Quartile (over versions)	8

list at hand, we created a series of processing tasks and added them to the task queue. Then we executed twenty five (Unix-based) workers that checked out tasks from the queue, processed the data and stored the results to the data repository.

A typical processing cycle of a worker included the following steps: after the worker spawned, it requested a task from the queue. This task contained the JAR name, which was typically a project version that was downloaded locally. First, specific JAR metadata were calculated and stored. Such metadata included its size, its dependencies, and a number that represented the chronological order of the release. This order was derived from an XML file that accompanies every project in the Maven repository called *maven-metadata.xml*. Then, FindBugs was invoked by the worker and its results were also stored in the data repository. When the task was completed the queue was notified and the next task was requested. This process was executed for all the available JARs in the task queue. A schematic representation of the data processing architecture can be seen in Figure 1.

3.2 Data Provenance

Initially, we obtained a snapshot (January 2012) of the Maven repository and handled it locally to retrieve a list of all the names of the project versions that existed in it. A project version can be uniquely identified by the triplet: *group id*, *artifact id* and *version*.

FindBugs works by examining the compiled Java virtual machine bytecodes of the programs it checks, using the bytecode engineering library (BCEL). To detect a bug, FindBugs uses various formal methods like *control flow* and *data flow analysis*. It has also other detectors that employ *visitor patterns* over classes and methods by using *state machines* to reason about values stored in variables or on the stack. Since FindBugs analyses applications written in the Java programming language, and the Maven repository hosts projects from languages other than Java such as Scala, Groovy, Clojure, etc., we filtered out such projects by performing a series of checks in the repository data and metadata.

In addition, we implemented a series of audits in the worker scripts that checked if the JARs are valid in terms of implementation. For instance, for every JAR the worker checked if there were any *.class* files before invoking FindBugs. After the project filtering, we narrowed down our data set to 17,505 projects with 115,214 versions. Table 1 summarises the data set information and provides the basic descriptive statistic measurements. The distribution of version count among the selected projects is presented in Figure 2.

The statistical measurements presented in Table 1 indicate that we have 17,505 projects and the data set's median is 3, which means that almost 50% (8,753 projects) of the project population have 1 to 3 versions. In general, most projects have a few number of versions, there are some projects with ten versions and only a few with hundreds of versions. The maximum number of versions for a project is 338. The 3^{rd} quartile measurement also indicated that 75% (13,129) of the projects have a maximum of 8 versions.

3.3 Threats to Validity

A threat to the internal validity of our experiment could be the false alarms of the FindBugs tool [1, 18]. False positives and negatives of static analysis tools and how they can be reduced is an issue that has already been discussed in the literature (see Section 2). In addition, reported security bugs may not be applicable to an application's typical use context. For instance, FindBugs could report an SQL injection vulnerability [32] in an application that receives no external input. In this particular context, this would be a false positive alarm.

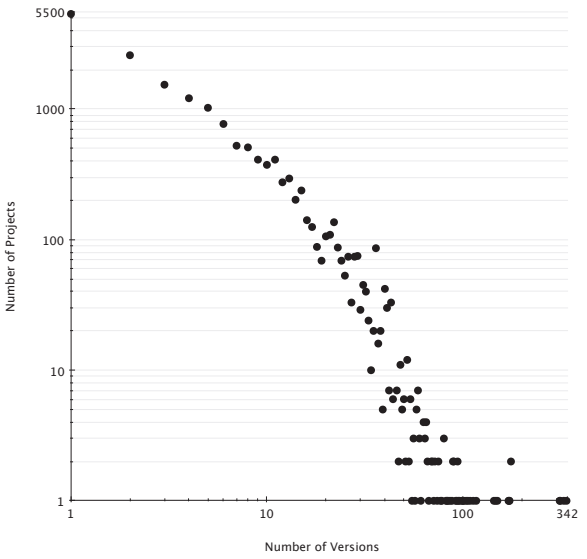

Figure 2: Distribution of version count among project population.

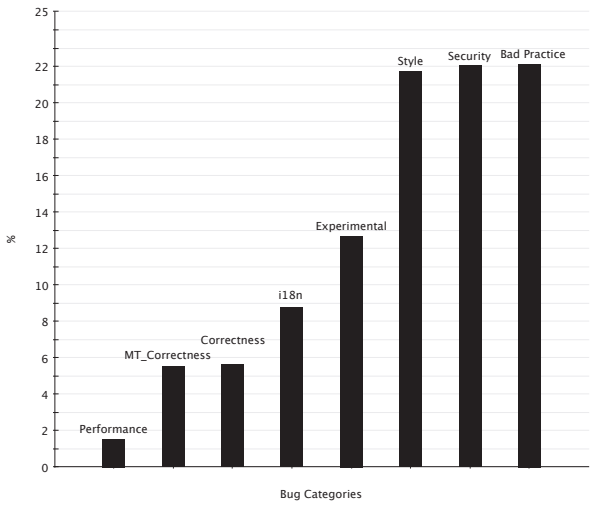

Figure 3: Bug percentage in Maven repository.

Furthermore, given that our analysis is done on open-source projects written in the Java programming language and hosted on Maven, a threat to the external validity of our work is the fact that our results may not be applicable to other programming languages, ecosystems, and development cultures. In particular, a large class of security problems such as buffer overflows [21] do not apply in our study since Java enforces bound checking at runtime.

4 Results and Analysis

Our findings can be analysed at two levels. First, we discuss some primary observations concerning the security bugs of the Maven repository as a whole. Then, we provide a comprehensive analysis of the results and highlight our key findings.

4.1 Overview and Initial Results

FindBugs separates software bugs into nine categories (see Table 2). Two of them involve security issues: *Security* and *Malicious Code*. From the total number of releases, 4,353 of them contained at least one bug coming from the first category and 45,559 coming from the second.

Our first results involve the most popular bugs in the Maven repository. Figure 3 shows how software bugs are distributed among the repository. Together with the *Bad Practice* bugs and the *Style* bugs, security bugs (the sum of the *Security* and *Malicious Code* categories - 0.21% + 21.81%) are the most popular in the repository ($\geq 21\%$ each). This could be a strong indication that programmers write code that implements the required functionality without considering its many security aspects; an issue that has already been reported in literature [37].

Another observation involves bugs that we could call *Security High* and they are a subset of the *Security* category. Such bugs are related to vulnerabilities that appear due to the lack of user-input validation and can lead to damaging attacks like SQL injection and Cross-Site Scripting [32]. To exploit such vulnerabilities, a malicious user does not have to know anything about the application internals. For almost all the other security bugs (coming from the *Malicious Code* category and the rest of the *Security* category bugs), another program should be written to incorporate references to mutable objects, access non-final fields, etc. Also, as bug descriptions indicate,[4] if an application has bugs coming from the *Security High* category, it might have more vulnerabilities that FindBugs doesn't report. Table 3 presents the number of releases where at least one of these bugs exists. In essence, 5,501 releases ($\approx 4,77\%$), contained at least one severe security bug. Given the fact that other projects include these versions as their dependencies, they are automatically rendered vulnerable if they use the code fragments that include the defects. The remaining bugs of the *Security* category are grouped together with the bugs of the *Malicious Code* category in another subcategory that we call *Security Low*. This category contains for the most part, bugs that imply violations of good OOP (object-oriented programming) design (i.e. keeping variables private to classes and others). The above categorization was done specifically to point out the behaviour of bugs that currently top the corresponding lists of most

Table 2: Bug categorisation according to FindBugs.

Category	Description
Bad Practice	Violations of recommended and essential coding practice.
Correctness	Involves coding misting a way that is particularly different from the other bug sakes resulting in code that was probably not what the developer intended.
Experimental	Includes unsatisfied obligations. For instance, forgetting to close a file.
Internationalization (i18n)	Indicates the use of non-localized methods.
Multi-Threaded (MT) Correctness	Thread synchronization issues.
Performance	Involves inefficient memory usage allocation, usage of non-static classes.
Style	Code that is confusing, or written in a way that leads to errors.
Malicious Code	Involves variables or fields exposed to classes that should not be using them.
Security	Involves input validation issues, unauthorized database connections and others.

Table 3: Number of project releases that contain at least one "Security High" bug.

Bug Description	Number of Project Releases
HRS: HTTP cookie formed from untrusted input	151
HRS: HTTP response splitting vulnerability	1,579
PT: absolute path traversal in servlet	103
PT: relative path traversal in servlet	57
SQL: non-constant string passed to execute method on an SQL statement	1,875
SQL: a prepared statement is generated from a non-constant String	1,486
XSS: JSP reflected cross site scripting vulnerability	18
XSS: Servlet reflected cross site scripting vulnerability in error page	90
XSS: Servlet reflected cross site scripting vulnerability	142

security providers.[5]

Linus's Law states that "given enough eyeballs, all bugs are shallow". In a context like this, we expect that the project versions that are dependencies to many other projects would have a small number of security bugs. To examine this variation of the Linus's Law and highlight the *domino effect* [39] we did the following: during the experiment we retrieved the dependencies of every version. Based on this information we created a graph that represented the snapshot of the Maven repository. The nodes of the graph represented the versions and the vertices their dependencies. The graph was not entirely accurate. For instance, if a dependency was pointing only to a project (and not to a specific version), we chose to select the latest version found on the repository. Also, this graph is not complete. This is because there were missing versions. From the 565,680 vertices, 191,433 did not point to a specific version while 164,234 were pointing to missing ones. The graph contained 80,354 nodes. Obviously, the number does not correspond to the number of the total versions (see Section 3.2). This is because some versions did not contain any information about their dependencies so they are not represented in the graph. After creating the graph, we ran the PageRank algorithm [5] on it and retrieved all PageRanks for every node. Then we examined the security bugs of the fifty most popular nodes based on their PageRank. Contrary to Linus's Law, thirty three of them contained bugs coming from the *Security Low* subcategory, while two of them contained *Security High* bugs. Twenty five of them were latest versions at the time. This also highlights the domino effect.

4.2 Analysis

Here, we present our key findings concerning the evolution of security bugs.

4.2.1 How Security Bugs Evolve Over Time

The relation between bugs and time can be traced from the number of bugs per category in each project version. We can then calculate the Spearman correlations between the defects count and the ordinal version number across all projects to see if bigger versions relate to higher or lower defect counts. The results are shown in Table 4. Although the tendency is for defect counts to increase, this tendency is extremely slight.

The zero tendency applies to all versions of all projects together. The situation might be different in individual projects. We therefore performed Spearman correlations between bug counts and version ordinals in all projects we examined. These paint a different picture from the above table, shown in Figure 4. The spike in point zero is explained by the large number of projects for which no correlation could be established—note that the scale is logarithmic. Still, we can see that there were projects where a correlation could be established, either positive or negative. The *Security High* category is particularly bimodal, but this is explained by the small number of correlations that could be established, nine in total.

Overall, Table 4 and Figure 4 suggest that **we cannot say that across projects defect counts increase or decrease significantly across time**. In individual projects, however, defect counts can have a strong upwards or downwards tendency. There may be no such thing as a "project" in general, only particular projects with their own idiosyncrasies, quality features, and coding practices.

Another take on this theme is shown in Figure 5, which presents a histogram of the changes of different bug counts in project versions. In most cases, a bug count does not change between versions; but when it does change, it may change upwards or downwards. Note also the spectacular changes of introducing or removing thousands of defects; this may be the result of doing and undoing a pervasive code change that runs foul of some bug identification rule.

Table 4: Correlations between version and defects count.

Category	Spearman Correlation	p-value
Security High	0.08	$\ll 0.05$
Security Low	0.02	$\ll 0.05$
Style	0.03	$\ll 0.05$
Correctness	0.04	$\ll 0.05$
Bad Practice	0.03	$\ll 0.05$
MT Correctness	0.09	$\ll 0.05$
i18n	0.06	$\ll 0.05$
Performance	*(0.01)*	0.07
Experimental	0.09	$\ll 0.05$

4.2.2 Persistence of Security Bugs

To examine the relation between the persistence of different kinds of bugs, and of security bugs in particular, we used as a persistence indicator the number of versions a bug remains open in a project. To "tag" a bug we created a bug identifier by using the type of the bug, the method name and the class name in which the bug was found in. We chose not to use the line number of the location of the bug since it could change from version to version and after a possible code refactoring. We grouped the persistence numbers by bug categories and then performed a Mann-Whitney U [16] test among all bug category pairs. The results are presented in Table 6 (at the end of this paper). Cells in brackets show pairs where no statistically significant difference was found.

In general, although the average number of versions bugs in different bug categories that remained open was statistically different in many cases, the difference is not spectacular. **In all cases a bug persists on average between two and three versions**, with the difference being in the decimal digits.

4.2.3 The Relation of Defects with the size of a JAR

We explored the relation between defects with the size of a project version, measured by the size of its JAR file by carrying out correlation tests between the size and the defect counts for each project and version. The results, all statistically significant ($p \ll 0.05$) can be seen in Table 5. **The** *Security High* **category stands out by having a remarkably lower effect than the other categories**, even *Security Low* that nearly tops the list. As we mentioned earlier, bugs that belong to the *Security High* category are related to user-input validation issues. Hence, even if a programmer adds code to a new version, if this code does not require user input, the possibility of such bug is minimal.

Table 5: Correlations between JAR size and defects count.

Category	Spearman Correlation	p-value
Security High	0.19	$\ll 0.05$
Security Low	0.65	$\ll 0.05$
Style	0.68	$\ll 0.05$
Correctness	0.51	$\ll 0.05$
Bad Practice	0.67	$\ll 0.05$
MT Correctness	0.51	$\ll 0.05$
i18n	0.53	$\ll 0.05$
Performance	0.63	$\ll 0.05$
Experimental	0.36	$\ll 0.05$

4.2.4 Security Bugs VS Other Bug Categories

To see whether bugs flock together we performed pairwise correlations between all bug categories. We calculated the correlations between the number of distinct bugs that appeared in a project throughout its lifetime, see Figure 6. We found significant, but not always strong, correlations between all pairs. In general, the *Security*

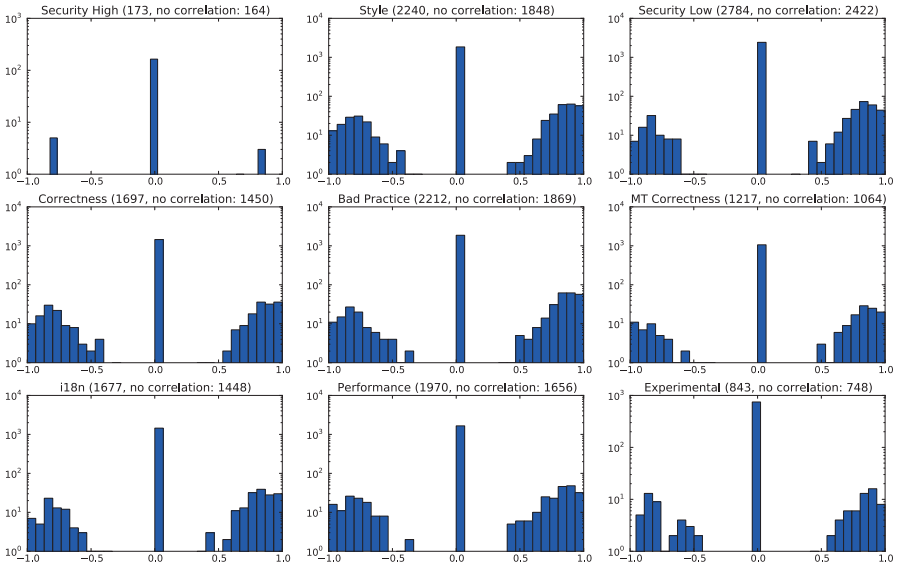

Figure 4: Histograms of correlations between bug counts and version ordinals per project. In brackets the total population size and the number of no correlation instances.

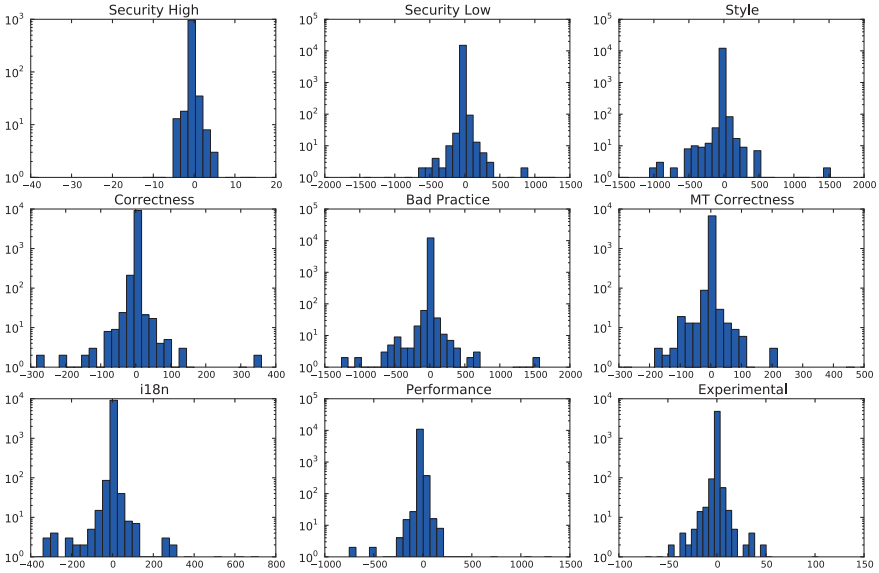

Figure 5: Changes in bug counts between versions.

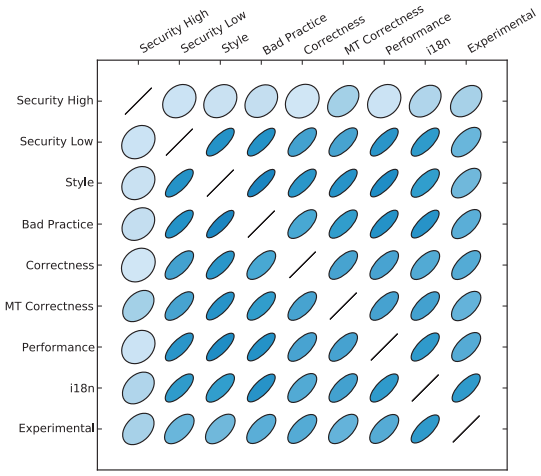

Figure 6: Correlation matrix plot for bug categories.

High category showed the weakest correlations with the other categories. Our results show that in general **bugs do flock together**. We do not find projects with only a certain kind of bug; bugs come upon projects in swarms of different kinds. Bugs of the *Security High* category, though, are different: they are either not associated with other bugs, or only weakly so. Perhaps it takes a special kind of blunder to make it a security hazard. Thus, to find such defects, code reviewers with experience in software security issues might be needed.

5 Conclusions and Future Work

We analysed more than 260GB of interdependent project versions to see how security bugs evolve over time, their persistence, their relation with other bug categories, and their relationship with size in terms of bytecode.[6]

Our primary hypothesis was that security bugs, and especially severe ones, would be corrected as projects evolve. We found that, although bugs do close over time in particular projects, we do not have an indication that across projects they decrease as projects mature. Moreover, defect counts may increase, as well as decrease in time. Contrary to our second research hypothesis, we found that security bugs are not eliminated in a way that is particularly different from the other bugs. Also, having an average of two to three versions persistence in a sample where 60% of the projects have three versions, is not a positive result especially in the case of the *Security High* bugs. Concerning the relation between severe security bugs and a project's size we showed that they are not proportionally related. Given that, we could say that it would be productive to search for and fix security bugs even if a project grows bigger. Furthermore, the pairwise correlations between all categories indicated that even though all the other categories are related, severe bugs do not appear together with the other bugs. Also, it is interesting to see that security bugs were one of the top two bug categories existing in a large ecosystem. Finally, we highlighted the domino effect, and showed evidence that indicates that Linus's Law does not apply in the case of the security bugs.

Contrary to the approaches that examine versions formed after every change that has been committed to the repository, our observations are made from a different perspective. The versions examined in this work were actual releases of the projects. As a result we do not have an indication of how many changes have been made between the releases. In essence, these JARs were the ones that were or still are, out there in the wild, being used either as applications, or dependencies of others.

Furthermore, the fact that projects have their own idiosyncrasies concerning security bugs, could help us answer questions like: what are the common characteristics of the projects where security bugs increase over time? In addition, by examining source code repositories more closely we could see how different development styles (i.e. size of commits, number of developers) affect projects.

By selecting an large ecosystem that includes applications written only in Java, we excluded by default measurements that involve vulnerabilities like the infamous buffer overflow defects [21]. Still, by examining software artifacts with similar characteristics facilitates the formation of an experiment. Thus, future work on our approach could also involve the observation of other ecosystems, that serve different languages, in the same manner such as, Python's PyPY (Python Package Index), Perl's CPAN (Comprehensive Perl Archive Network), and Ruby's RubyGems.

6 Acknowledgments

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework - Research Funding Program: Heracleitus II. Investing in knowledge society through the ESF.

References

[1] AYEWAH, N., AND PUGH, W. The google FindBugs fixit. In *Proceedings of the 19th international symposium on Software testing and analysis* (New York, NY, USA, 2010), ISSTA '10, ACM, pp. 241–252.

[2] BACA, D., CARLSSON, B., AND LUNDBERG, L. Evaluating the cost reduction of static code analysis for software security. In

Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security (New York, NY, USA, 2008), PLAS '08, ACM, pp. 79–88.

[3] BHATTACHARYA, P. Using software evolution history to facilitate development and maintenance. In *Proceedings of the 33rd International Conference on Software Engineering* (New York, NY, USA, 2011), ICSE '11, ACM, pp. 1122–1123.

[4] BHATTACHARYA, P., AND NEAMTIU, I. Bug-fix time prediction models: can we do better? In *Proceedings of the 8th Working Conference on Mining Software Repositories* (New York, NY, USA, 2011), MSR '11, ACM, pp. 207–210.

[5] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst. 30*, 1-7 (Apr. 1998), 107–117.

[6] CHESS, B., AND WEST, J. *Secure programming with static analysis*, first ed. Addison-Wesley Professional, 2007.

[7] D'AMBROS, M. Supporting software evolution analysis with historical dependencies and defect information. In *ICSM* (2008), pp. 412–415.

[8] D'AMBROS, M., AND LANZA, M. A flexible framework to support collaborative software evolution analysis. In *CSMR* (2008), pp. 3–12.

[9] EDWARDS, N., AND CHEN, L. An historical examination of open source releases and their vulnerabilities. In *Proceedings of the 2012 ACM conference on Computer and communications security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 183–194.

[10] FISCHER, M., PINZGER, M., AND GALL, H. Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance* (Washington, DC, USA, 2003), ICSM '03, IEEE Computer Society, pp. 23–.

[11] GIL, J. Y., AND MAMAN, I. Micro patterns in java code. In *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (New York, NY, USA, 2005), OOPSLA '05, ACM, pp. 97–116.

[12] GÖRG, C., AND WEISSGERBER, P. Error detection by refactoring reconstruction. In *Proceedings of the 2005 international workshop on Mining software repositories* (New York, NY, USA, 2005), MSR '05, ACM, pp. 1–5.

[13] GRAVES, T. L., KARR, A. F., MARRON, J. S., AND SIY, H. Predicting fault incidence using software change history. *IEEE Trans. Softw. Eng. 26*, 7 (2000), 653–661.

[14] HECKMAN, S., AND WILLIAMS, L. On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (New York, NY, USA, 2008), ESEM '08, ACM, pp. 41–50.

[15] HECKMAN, S., AND WILLIAMS, L. A model building process for identifying actionable static analysis alerts. In *Proceedings of the 2009 International Conference on Software Testing Verification and Validation* (Washington, DC, USA, 2009), ICST '09, IEEE Computer Society, pp. 161–170.

[16] HETTMANSPERGER, T. P., AND MCKEAN, J. W. *Robust nonparametric statistical methods*. Kendall's Library of Statistics, 1998.

[17] HINDLE, A., AND GERMAN, D. M. SCQL: a formal model and a query language for source control repositories. In *Proceedings of the 2005 international workshop on Mining software repositories* (New York, NY, USA, 2005), MSR '05, ACM, pp. 1–5.

[18] HOVEMEYER, D., AND PUGH, W. Finding bugs is easy. *SIGPLAN Not. 39*, 12 (Dec. 2004), 92–106.

[19] HOVEMEYER, D., AND PUGH, W. Finding more null pointer bugs, but not too many. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering* (New York, NY, USA, 2007), PASTE '07, ACM, pp. 9–14.

[20] KAGDI, H., COLLARD, M. L., AND MALETIC, J. I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol. 19*, 2 (Mar. 2007), 77–131.

[21] KEROMYTIS, A. D. Buffer overflow attacks. In *Encyclopedia of Cryptography and Security (2nd Ed.)*. 2011, pp. 174–177.

[22] KIM, M., CAI, D., AND KIM, S. An empirical investigation into the role of api-level refactorings during software evolution. In *Proceedings of the 33rd International Conference on Software Engineering* (New York, NY, USA, 2011), ICSE '11, ACM, pp. 151–160.

[23] KIM, S., AND ERNST, M. D. Which warnings should i fix first? In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* (New York, NY, USA, 2007), ESEC-FSE '07, ACM, pp. 45–54.

[24] KIM, S., PAN, K., AND WHITEHEAD, JR., E. J. Micro pattern evolution. In *Proceedings of the 2006 international workshop on Mining software repositories* (New York, NY, USA, 2006), MSR '06, ACM, pp. 40–46.

[25] LANYUE LU, ANDREA C. ARPACI-DUSSEAU, REMZI H. ARPACI-DUSSEAU, SHAN LU. A Study of Linux File System Evolution. In *Proceedings of the 11th Conference on File and Storage Technologies (FAST '13)* (San Jose, California, February 2013).

[26] LEHMAN, M. M., RAMIL, J. F., WERNICK, P. D., PERRY, D. E., AND TURSKI, W. M. Metrics and laws of software evolution - the nineties view. In *Proceedings of the 4th International Symposium on Software Metrics* (Washington, DC, USA, 1997), METRICS '97, IEEE Computer Society, pp. 20–.

[27] MASSACCI, F., NEUHAUS, S., AND NGUYEN, V. H. After-life vulnerabilities: a study on firefox evolution, its vulnerabilities, and fixes. In *Proceedings of the Third international conference on Engineering secure software and systems* (Berlin, Heidelberg, 2011), ESSoS'11, Springer-Verlag, pp. 195–208.

[28] NAGAPPAN, N., BALL, T., AND ZELLER, A. Mining metrics to predict component failures. In *ICSE '06: Proceedings of the 28th international conference on Software engineering* (New York, NY, USA, 2006), ACM, pp. 452–461.

[29] NUSSBAUM, L., AND ZACCHIROLI, S. The ultimate debian database: Consolidating bazaar metadata for quality assurance and data mining. In *Proceedings of the 2010 international workshop on Mining software repositories* (2010), MSR '10, pp. 52–61.

[30] OZMENT, A., AND SCHECHTER, S. E. Milk or wine: does software security improve with age? In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15* (Berkeley, CA, USA, 2006), USENIX-SS'06, USENIX Association.

[31] RATZINGER, J., SIGMUND, T., AND GALL, H. C. On the relation of refactorings and software defect prediction. In *Proceedings of the 2008 international working conference on Mining software repositories* (New York, NY, USA, 2008), MSR '08, ACM, pp. 35–38.

[32] RAY, D., AND LIGATTI, J. Defining code-injection attacks. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (New York, NY, USA, 2012), POPL '12, ACM, pp. 179–190.

[33] SHAHRIAR, H., AND ZULKERNINE, M. Mitigating program security vulnerabilities: Approaches and challenges. *ACM Comput. Surv. 44*, 3 (June 2012), 11:1–11:46.

[34] SHAHZAD, M., SHAFIQ, M. Z., AND LIU, A. X. A large scale exploratory analysis of software vulnerability life cycles. In *Proceedings of the 2012 International Conference on Software Engineering* (Piscataway, NJ, USA, 2012), ICSE 2012, IEEE Press, pp. 771–781.

[35] SHEN, V. Y., YU, T.-J., THEBAUT, S. M., AND PAULSEN, L. R. Identifying error-prone software an empirical study. *IEEE Trans. Softw. Eng. 11*, 4 (Apr. 1985), 317–324.

[36] SPACCO, J., HOVEMEYER, D., AND PUGH, W. Tracking defect warnings across versions. In *Proceedings of the 2006 international workshop on Mining software repositories* (New York, NY, USA, 2006), MSR '06, ACM, pp. 133–136.

[37] STAMAT, M. L., AND HUMPHRIES, J. W. Training $\neq$ education: putting secure software engineering back in the classroom. In *Proceedings of the 14th Western Canadian Conference on Computing Education* (New York, NY, USA, 2009), WCCCE '09, ACM, pp. 116–123.

[38] TELANG, R., AND WATTAL, S. Impact of software vulnerability announcements on the market value of software vendors - an empirical investigation. In *Workshop on the Economics of Information Security* (2007), p. 677427.

[39] TEVIS, J.-E. J., AND HAMILTON, J. A. Methods for the prevention, detection and removal of software security vulnerabilities. In *Proceedings of the 42nd annual Southeast regional conference* (New York, NY, USA, 2004), ACM-SE 42, ACM, pp. 197–202.

[40] WEISSGERBER, P., AND DIEHL, S. Are refactorings less error-prone than other changes? In *Proceedings of the 2006 international workshop on Mining software repositories* (New York, NY, USA, 2006), MSR '06, ACM, pp. 112–118.

[41] ZAMAN, S., ADAMS, B., AND HASSAN, A. E. Security versus performance bugs: a case study on firefox. In *Proceedings of the 8th Working Conference on Mining Software Repositories* (New York, NY, USA, 2011), MSR '11, ACM, pp. 93–102.

[42] ZIMMERMANN, T., NAGAPPAN, N., AND ZELLER, A. Predicting bugs from history. In *Software Evolution*. 2008, pp. 69–88.

Notes

[1] `http://findbugs.sourceforge.net/`

[2] `http://mvnrepository.com/`

[3] `http://www.mozilla.org/projects/security/known-vulnerabilities.html`

[4] `http://findbugs.sourceforge.net/bugDescriptions.html`

[5] `http://cwe.mitre.org/top25/index.html`

[6] Our data and code are available online at: `https://github.com/bkarak/evol_security_publication_2012`

Table 6: Bug persistence comparison.

Security High	(0.04, $p = 0.97$ *2.72, 2.36* *243, 35048)*	2.22, $p < 0.05$ 2.72, 2.12 243, 49043	(−0.51, $p = 0.61$ *2.72, 2.50* *243, 12905)*	2.77, $p < 0.01$ 2.72, 2.11 243, 49324	(1.02, $p = 0.31$ *2.72, 2.48* *243, 10227)*	(−1.19, $p = 0.23$ *2.72, 2.74* *243, 10718)*	(−1.00, $p = 0.32$ *2.72, 2.65* *243, 23598)*	(−0.33, $p = 0.74$ *2.72, 2.85* *243, 2686)*
Security Low		20.27, $p \ll 0.05$ 2.36, 2.12 35048, 49043	−3.59, $p \ll 0.05$ 2.36, 2.50 35048, 12905	25.17, $p \ll 0.05$ 2.36, 2.11 35048, 49324	5.59, $p \ll 0.05$ 2.36, 2.48 35048, 10227	−7.55, $p \ll 0.05$ 2.36, 2.74 35048, 10718	−8.19, $p \ll 0.05$ 2.36, 2.65 35048, 23598	(−1.39, $p = 0.17$ *2.36, 2.85* *35048, 2686)*
Style			−17.96, $p \ll 0.05$ 2.12, 2.50 49043, 12905	5.66, $p \ll 0.05$ 2.12, 2.11 49043, 49324	−6.84, $p \ll 0.05$ 2.12, 2.48 49043, 10227	−20.61, $p \ll 0.05$ 2.12, 2.74 49043, 10718	−26.18, $p \ll 0.05$ 2.12, 2.65 49043, 23598	−8.30, $p \ll 0.05$ 2.12, 2.85 49043, 2686
Correctness				21.38, $p \ll 0.05$ 2.50, 2.11 12905, 49324	7.44, $p \ll 0.05$ 2.50, 2.48 12905, 10227	−3.57, $p \ll 0.05$ 2.50, 2.74 12905, 10718	−2.91, $p < 0.01$ 2.50, 2.65 12905, 23598	(0.40, $p = 0.69$ *2.50, 2.85* *12905, 2686)*
Bad Practice					−10.02, $p \ll 0.05$ 2.11, 2.48 49324, 10227	−23.63, $p \ll 0.05$ 2.11, 2.74 49324, 10718	−30.32, $p \ll 0.05$ 2.11, 2.65 49324, 23598	−9.98, $p \ll 0.05$ 2.11, 2.85 49324, 2686
MT Correctness						−10.17, $p \ll 0.05$ 2.48, 2.74 10227, 10718	−10.83, $p \ll 0.05$ 2.48, 2.65 10227, 23598	−4.03, $p \ll 0.05$ 2.48, 2.85 10227, 2686
i18n							(1.29, $p = 0.20$ *2.74, 2.65* *10718, 23598)*	2.46, $p < 0.05$ 2.74, 2.85 10718, 2686
Performance								(1.92, $p = 0.05$ *2.65, 2.85* *23598, 2686)*

Security Low
Style
Correctness
Bad Practice
MT Correctness
i18n
Performance
Experimental

The matrix presents pairwise Mann-Whitney U test results between the different bug categories. Each cell contains the test result (the value of U), the p-value, the average for each category and the sample size for each category. Cells in brackets show pairs where no statistically significant difference was found.