Crashing Privacy: An Autopsy of a Web Browser’s Leaked Crash Reports

Kiavash Satvat  
University of Alabama at Birmingham  
kiavash@uab.edu

Nitesh Saxena  
University of Alabama at Birmingham  
saxena@uab.edu

ABSTRACT
Harm to the privacy of users through data leakage is not an unknown issue, however, it has not been studied in the context of the crash reporting system. Automatic Crash Reporting Systems (ACRS) are used by applications to report information about the errors happening during a software failure. Although crash reports are valuable to diagnose errors, they may contain users’ sensitive information. In this paper, we study such a privacy leakage vis-a-vis browsers’ crash reporting systems. As a case study, we mine a dataset consisting of crash reports collected over the period of six years. Our analysis shows the presence of more than 20,000 sessions and token IDs, 600 passwords, 9,000 email addresses, an enormous amount of contact information, and other sensitive data. Our analysis sheds light on an important security and privacy issue in the current state-of-the-art browser crash reporting systems. Further, we propose a hotfix to enhance users’ privacy and security in ACRS by removing sensitive data from the crash report prior to submit the report to the server. Our proposed hotfix can be easily integrated into the current implementation of ACRS and has no impact on the process of fixing bugs while maintaining the reports’ readability.

1 INTRODUCTION
A few studies were conducted, focusing on legal, financial, reputational, and other aspects of information disclosure [1, 10, 30]. Some of these studies analyzed a specific data leakage incident [1, 10]. Similarly, several studies analyzed the security and privacy of web browsers [8, 19, 31, 35]. However, no previous study was conducted to examine the risks that the current implementation of browsers’ Automatic Crash Reporting System (ACRS) may pose to the users’ privacy.

Crash as an inevitable fact occurs when a software terminates its normal execution process unexpectedly. The use of ACRS is a common approach to collect crash reports from the users. ACRS makes troubleshooting smooth amongst a variety of factors contributing to the crash, such as underlying hardware and associated software. Using ACRS, companies collect crash reports from clients to address application bugs and improve users’ experience during web browsing. This information is used by developers to detect the crash root cause and rectify the bug. Chrome, Firefox, Safari, and Opera are a few common names that use the ACRS to collect field crashes [4, 16, 25, 27]. In web browsers, ACRS is designed to collect the stack trace of the failing thread, user’s system environmental information (e.g., OS and browser version), and probable user’s feedback on the crash.

However, a fact that has been widely disregarded is the content of the crash collected from the users’ systems and the amount of private information that can be found in crash reports which may pose a threat to the users’ privacy.

In this paper, to demonstrate possible risks that the current implementation of ACRS may pose to users’ privacy, we study a dataset of partially anonymized crash reports released by one of the major browsers. The current state of the art browsers collect relatively similar data including browsed URL during the crash and runtime information, as they can be required for the debugging. To the best of our knowledge, this is the first case study which scrutinizes a database of 2.5 million partially anonymized crash reports, containing visited URLs, system runtime information and users’ descriptions. Conducting our inspection, we extracted a significant amount of private data including, approximately 20,000 session and token IDs, 600 clear text password, 9,000 email address, and an enormous amount of contact information, including names, addresses, and phone numbers. The obtained results signify the deficiency of the current approach and shows the potential for ACRS to be a compelling target for attackers. Furthermore, to address this deficiency, we proposed a hotfix to remove the sensitive information from the environmental information, users’ feedback, and visited URLs at the client side prior to disseminating to the server.

Our Contribution. The detail contributions of our system are as follow.

• Data Analysis (Section 4). We conduct an empirical study on partially anonymized crash reports released by one of the major web browsers. Through our inspection, we extract and represent a significant amount of personal information and confidential data presented in the reports, delineating to what extent the current implementation of ACRS can be harmful to users’ security and privacy. Our analysis sheds light on an important security and privacy issue in the current state-of-the-art browser crash reporting systems.

• Deploying A Hotfix (Section 5). According to the result of our data analysis, we propose a hotfix, to enhance users’ privacy and security in ACRS, by removing sensitive data from the users’ description field and URL. Our hotfix can be easily integrated into the current implementation of ACRS and has no impact on the process of fixing bugs while maintaining the reports’ readability. However, it protects users’ private data against unauthorized access in the cases server gets compromised or data get leaked.

Paper Organization. The rest of this paper is organized as follows. Section 2 provides the background about ACRS and explains the dataset used in this study. Section 3 defines the methodology used to analyze the data and presents the reports. In Section 4, we present the results of our data analysis. Section 5 describes insights

---

1 The name of the browser, year, and the fields of dataset are removed or anonymized as discussed in Section 3.1
and countermeasures. Finally, Section 6 concludes our study and suggests future research directions.

2 STUDY PRELIMINARIES AND DATASET

2.1 ACRS Background

The use of ACRS is a common approach for collecting and handling crashes. ACRS primarily consists of two main components. First, what we refer to as Collector, a set of client side libraries which collects the data from the clients and transfers it to the server for further processing. Second, the Processor which is a set of servers and services responsible for analyzing, categorizing, and reporting the collected crashes. On the client side, Collector, as an interacting interface between the Processor and clients, runs when the application terminates its normal execution process. Collector typically asks for the users’ permission to send the crash report to Processor or to quit without sending the report. While choosing to send the report to the Processor, the user has an option to include the visited website at the time crash occurred or not. The user also can provide additional information in the description field to explain the issue that caused the crash. Additionally, there might be an option for the user to provide an email address for future support and contributions. After these steps, Collector transfers a dump of crash report and collected details to Processor that is responsible for further processing and presenting for the view of developers and the general public.

The collected data from clients are referred to as Crash reports. Each crash report is composed of two components. The first component is the stack trace of the failed thread which is generally referred to the minidump and carries information about the system memory. The second component is the runtime information of the users’ system and the possible feedback provided by the user. While these details are essential for the developers to detect where the problem lies, they may contain users’ sensitive information. For instance, the minidump as a sensitive chunk of data may contain information such as username, password, and encryption key [7]. Also, it is not hard to expect that the runtime information and users’ comment may carry private information such as contact details and other sensitive data.

2.2 Dataset

In this paper, we study a dataset of 2,493,278 partially anonymized crash reports consists of details such as visited URLs, time of the crash, the client operating system, and user description of the crash. A dataset aggregated by a major web browser during the course of 6 years. The current approach employed by the software companies, including web browsers (e.g., [24, 28]), stores similar data in crashes. Our close inspection revealed that while some of the fields were fully anonymized (e.g., field email), unsuccessful de-identifications of the other fields left a notable amount of sensitive and personal information in the database. Private data like the usernames, passwords, and emails embedded in the URL field, or some confidential information in the description field which is shared by the users. We noticed the presence of three types of data in this dataset;

- **Deleted Fields**: Fields like email and login were deleted since they explicitly carry sensitive information.
- **Masked Fields**: In this category, some efforts have been taken for data de-identification. In ip, some records were de-identified by replacing the last two digits of IP address with zeros.
- **Untouched Fields**: All fields except email, login and address apparently remained untouched, however, not necessarily all of them convey any meaningful information. In this paper, we mostly focused on these attributes to demonstrate the breach of the information.

3 METHODOLOGY

In this section, we explain our approach for inspecting and analyzing the dataset. We also describe the methodology we use to present our results in Section 4.

3.1 Ethical Consideration

We conducted our research based on a dataset which was initially published and later removed following a report regarding the presence of sensitive information. During our scrutiny, however, we noticed the presence of similar data which is widely available for the public use. Therefore our study aims to highlight a potential pitfall associated with the current implementation of the ACRS to the researchers, software developers, and the society. Our study neither propagates the data nor does it draw attention to any specific company or individual. Hence, we believe our study causes no further harm to victims. To preserve the privacy of associated individuals, companies and third parties, and avoid working directly with this sensitive information, we used complex queries and avoided performing a manual check. Therefore, all results provided in Section 4 represents a close estimation based on the output of our queries. For the presentation, we removed identifiable information from our results, including but not limited to the names of individuals, websites, applications, companies and their contact information, such as phone numbers and addresses. The de-identification process was performed using a python script replacing this sensitive information with asterisks.

Similarly, we removed all dataset related details which may imply a vendor or result in identifying an individual including names, years, and other traceable information. We also changed the dataset’s fields’ names to avoid defaming company by possible cross referencing of presented data in this paper.

3.2 Our Approach

**Main Fields of the Database.** In this study, we mainly focus on two fields of `url` and description which carry more sensitive information.

- **description.** To understand what might be laid in this field we considered the following questions: “What are the possible users’ feedbacks on crashes?”, “What might be the response of an expert user to the crash?”, “Is it possible that a user shares his private information over the description field?”. According to these questions, we mined the fields, using keywords such as “username”, “password”, “my phone number”, and “please contact me” to inspect the presence of gripping information and the data that may impact users’ privacy.
A variety of risk analysis methodologies are proposed to appraise
Therefore, to present the result of our inspection, we define three
tions and gives each of these records its own unique characte ristic
and makes the prior methodologies defective towards this da taset.

For instance, while a record may only carry
these leakages consist of one type of entity. For instance, i n the case
context or type of the data, source of the leakage, and potent ial
risks [13, 34]. These methods consider several factors, includ ing
and calculate the breach severity and the impact of the assoc iated

Simple data about user’s browsing, the other may convey user ‘s
separate circumstance. For instance, to the breach. This similarity makes the assessment and eval ua-
tin the approximate location (e.g. city), and then spot the exact

3.3 Categorizing Data Breach Severity
A variety of risk analysis methodologies are proposed to appraise and calculate the breach severity and the impact of the associated risks [13, 34]. These methods consider several factors, including context or type of the data, source of the leakage, and potential impact to rank the severity of each incident. These methods are applicable to all previously reported incidents, as the majority of these leakages consist of one type of entity. For instance, in the case of the JPMorgan [29], as one of the most severe data breaches of all time [15], users’ contact information breached. In the case of Myspace data breach [39], subscribers’ account details were subjected to the breach. This similarity makes the assessment and evaluation of the associated risk relatively easy. However, in the case of browsers data leakage, we deal with a set of dissimilar and independent records, each being generated by a different user and in a disparate circumstance. For instance, while a record may only carry simple data about user’s browsing, the other may convey user’s medical condition or financial details. This diversity adds complications and gives each of these records its own unique characteristic and makes the prior methodologies defective towards this dataset. Therefore, to present the result of our inspection, we define three categories of risk severity that are designed to address our specific needs and are compatible with the current dataset. We present the data based on their severity into three categories:

• Low Risk: This category presents those records which do not impact individuals’ privacy, but accessing them may provide additional information to an attacker, helping to launch targeted attacks against specific population (e.g., users who are using a particular operating system in a small region or a specific range of IP address). This information may also be interesting to the researchers and developers as they statistically represent the web population.

• Medium Risk: This category presents those records which do not compromise users’ privacy or security, but a cross referencing with an auxiliary information can turn them into a threat to the users’ privacy.

• High Risk: This category presents records which explicitly violate users’ privacy and contains users’ sensitive information such as username, password, and contact details. Similar to the data presented in the previous categories data appeared in this category can be shared by users through description or URL field.

4 DATA ANALYSIS
In this section, we demonstrate the results of our data analysis and present them based on the severity level introduced in Section 3.

4.1 Low Risk
As discussed in Section 3.3, under the low risk category, we present information which does not violate privacy, but at the same time is highly descriptive in terms of providing statistical details, and can be a representative of the whole Internet users population. In contrast to reports offered by measurement websites (e.g., [22, 38]), our dataset encompasses a wide range of users, while they were visiting various websites. Moreover, the data has been collected over the course of six years and has no correlation with any specific application, company, or individual, which signifies the impartiality of our dataset.

Platforms and Protocols. Market share websites (e.g., [32, 33]) are the main source, to obtain statistic on the popularity of different platforms on the Internet. These websites build their database using the obtained data from their visitors’ browsers. Unlike these reports, the random distribution of users in the current dataset provides a more accurate statistic on the popularity of different platforms. In this study, such information were extracted from the platform field. Similarly, same approach can be taken to extract the of various web protocols amongst websites. Unlike other reports, which target the specific domains (e.g. most popular websites [18]) , the current dataset formed from a wide range of users and can be counted as a small sample of the whole web population.

Identified Attacks. Considering the fact that a significant number of attacks against websites are launched through URL, we queried the url to detect the presence of common attacks such as SQL Injection, Local File Inclusion, and Directory Traversal. Below are some of the known attacks we found during our inspection.

• SQL Injection: We used a combination of keywords such as “select”, “from”, “where” and “union” to detect the presence of SQL injection. Below URL illustrates a situation in which the attacker’s browser crashed while he was launching a SQL attack against a website.

http://www.*****.com/photo/gallery/search.php?
search_user=x%2527%20union%20select%20user_password
%20from%204images_user%20where%20Andrew%20Whyte=
%2572

• Directory Traversal: In this form of attack, the attacker can get access to restricted directories of the web servers (e.g., root directory). The attacker can use expression “…/” to instruct the system
to move between directories. To look up this type of attack, we searched for "/" as a keyword. The following is an instance of this attack found in our database.

http://www.*****.com/de_old/index.html?prm_popup=../../

• Phishing Attack: Since it was not possible to search for all the websites, except by querying the url, we tried to inspect the users feedback on the crash. The below example is a paradigm where the user was trying to report a phishing website.

http://******/www.wellsfargo.com/

Similarly, we used a same approach to extract other suspicious records. For instance, we searched for "/etc/password" as a keyword for detecting the Local File Inclusion or "document.cookie" for the Cross Site Scripting (XSS).

4.2 Medium Risk
IP Address. We observed three types of IPs in ip. The first type contains IPs which were fully deleted, the second type consists of records which were masked by turning into “10.2.0.0” and finally 261,388 records that were partially anonymized by replacing the last two blocks of IP with zeros. Moreover, a close inspection of the url revealed a significant number of URLs with embedded IP addresses. The below sample shows an IP address inside a URL.

http://******/?getpostdata=get&name=******&site=submit&ip=**.**.**&ref=000/

Email. We extracted 9139 emails including personal and business email addresses. Out of which 7731 email obtained from the description field, while the remaining 1462 extracted from the URLs. The below sample shows an embedded email inside the URL.

https://******.net/activate.php?user=*****137199&email=*****@gmail.com/

Location. A fraction of users could have been located based on the data in url and description fields. locating users can occur either through the URL (e.g. when the user was trying to submit a registration form with an embedded address in URL) or by querying the description for the cases which user shared his address.

4.3 High Risk
Username and Password. We were able to pinpoint over 538 passwords. This number of users’ credentials further escalate the severity of this leakage when we consider the fact of the pervasiveness of password reuse (43-51%) [12]. Some of these passwords were extracted from the URL, and the rest were embedded in the description field. In the case of URL, the credentials were exposed due to the weaknesses in design and development of the application where both the HTTP and GET method have been used to transfer the sensitive data between client and server while this method of transferring has been refused by the standards [37]. Below URL demonstrates a case where the browser crashed when a user was about to log into a website.

http://******/logon.do?jsessionid=abbTXFV3-1BSw7?username=******&password=******

The next series of passwords were extracted from the description, where the user shared his password in the description field of Crash Reporter aiming to receive future support.

I can login but the ***** don’t remember password for this site. I tell you the password to login: username: ****** password: ****** you can login south west of page. if you need translation go to http://www.*****.net/dic/.

Token & SessionID. Stolen SessionID within its life cycle, means that the account is stolen [42]. Our inspection revealed 21298 session and token IDs. Though at the time of writing this paper we cannot verify the persistence of vulnerability, considering the fact that sessions authenticate users, we can understand the risk posed to individuals’ security and privacy at the time of the crash. Moreover, in our analysis, we noticed the presence of 7000 Tokens (as more severe authentication method compared to session IDs). The below URL depicts one of the samples with token Id.

https://******.com/******/Main/Login_WS.aspx?tokenid=880217f4-94c3-496d-8628-b238b4ef299

Contact Information. During our data analysis, we dealt with a significant number of contact information some laid in URL while the others were provided by users intentionally in the description field to receive further support. Among these records, the latter is specifically more threatening as can be used to launch a targeted attack against the inexpert user. The attacker with access to such records may easily undermine users’ security and privacy by contacting the user as a member of browser’s support team to deceive the user to install a Malware or request additional information from the target.

Please call me at ***** to discuss details about missing info on my ebay bid pages. Never a problem in the past, now an everyday problem.

Similarly, we noticed a considerable number of phone numbers laid in url. Below demonstrates a number embedded inside URL.

http://******/jcp/OrderDetail.aspx?context=OrderHist
&iia=ya2z*****2ZTt&pid=1&OrderNum=******&Phone=******/

5 INSIGHT AND COUNTERMEASURES
A variety of direct and indirect factors contributed to the privacy harm caused by this data leakage incident. Factors such as disregarding the importance of data retention and its impacts on users’ privacy and security [5, 6, 11], improper use of GET method and the human error [21, 40, 41] in disclosing users’ information. While these kinds of factors are widely discussed, yet they are inevitable and as the result, we witness an unprecedented wave of data leakages. Therefore, in this section, we propose a hotfix that can be deployed into the current ACRS, helping to enhance the security and privacy of users against such a data disclosure, by removing sensitive information from the crash report.

5.1 Client Side Data Sanitization
In the case of ACRS, both server and client side approaches can be employed to enhance users’ privacy and security by removing sensitive information from the reports. While removing the users’
sensitive data at the server side may provide more flexibility for the developers to get the best readability of a report, the user should not be obligated to trust the developer and the system to send their sensitive data to the server. Therefore, in this paper, we propose a client side approach to remove sensitive data from the crash report’s environmental information, aiming to safeguard users’ sensitive data right at the client side. This approach, as a hotfix to the current ACRS implementation, can be easily integrated with the popular ACRS distributions ([14, 17]), aiming to sanitize users’ private information such as username, password, and social security number without any impact on the bug fixation and with minimal or no impact on the readability of reports. Unlike [7] and [9] which only focused on sanitizing minidump, we consider the crash runtime information and supportive data including description field and URL. Similar to [7], we hypothesize that developers do not want to get occupied by users’ sensitive data.

To sanitize a report at the client, we developed a light weight script which parses the URL and user’s description field and removes the private information prior to transferring them to the Processor. The program masks the possible sensitive data from the crash report based on the predefined list of sensitive data (which can be defined by developers). The program takes URL and description as input and checks them against an array of predefined sensitive keywords which may appear in those fields. In case of any match, the program masks the following 4 characters of the sensitive keyword. The result will be a sanitized report where sensitive data is unusable (or difficult to guess) in case of an unauthorized access, but at the same time the report maintains its readability for the debugging purpose.

We define readability as a situation where developers are able to replicate the crash. For instance, in case of a browser crash, developers may require the visited URL in addition to stack trace information. From the URL developers are specifically looking for details like the name of the visited website and status where the crash occurred (e.g., posting a form). While our hotfix leaves these details untouched, it masks users’ sensitive information which may exist inside URL and is not in the interest of developers (similar to [7] hypothesis). Similarly, using our hotfix, a URL that carries token ID will be unusable by attacker for the purpose of login, as its first 4 digits are masked, but it still denotes the necessary details which are required for the debugging purpose. Depending on the URL or description field, the list may include sensitive such as username, password, token, and session ID. The list of keywords can be defined by developers, giving them the opportunity to make a balance between users’ privacy and readability required for the crash debugging. Algorithm 1, represents our proposed method for removing private data from the report and description field.

As a preliminary evaluation, we tested our program, against 500 independent records from description and url. To preserve users’ privacy during the testing phase and to make sure that the data is not traceable back to a user, we take some steps as follow. First, we tested each field independently to ensure that data is segregated from the other fields in the same record and therefore is not traceable back to users. Second, for the case of URL sanitization, we replaced the domain names with the www.example.com (replacing the domain names with the www.example.com was only performed for the testing purpose. The application in the normal process keeps the domain untouched as the developers may need it for the further analysis). After running our program against de-identified records, we performed a manual check to assure that the URLs’ readability is not compromised. The result of our manual check showed that the program was fully successful in manipulating the keywords.

The results of our evaluation on 500 URLs carrying session ID and 500 description fields carrying password show 100% success rate on masking sensitive information from both fields. If we define readability such that the program still maintains all information required to determine a crash root cause from the URL, including the name of the website and possible application or directories, the readability rate in the case of URL was over 92%. The result of our manual check indicates that readability of 40 URLs out of 500 URLs has been impacted. As in some cases URL sanitization moved insensitive characters from the URL or caused removal of the accessed directory on the web application.

However, in the case of the description field, the result cannot be quantitatively evaluated due to the qualitative nature of the readability factor of the description field. To estimate the readability rate of the description field a study should be conducted to ask developers with authorized access to the original and masked data to compare and rank the readability of the descriptions. Given that such a study may jeopardize the privacy of the users, we were not able to conduct this study.

The proposed approach soundly works in the case of sanitizing URL, since the cases from which the keywords are derived are relatively limited. However, the mass filtering of data may result in removing insensitive information from the description fields. As an example, developers choices for defining sessionId in URL are likely limited to sessionid, session=, and sid=. But in the case of users’ description, we deal with a great volume of uncertainty due to the infinite ways of writing and explaining a topic. Spelling, use of abbreviation may also add more complication to the situation. For instance, while description ‘‘I saved my username in browser, but using the saved password it doesn’t get through the website! looks the password is wrong!’’ contains no sensitive data, mass filtering turns it to ‘‘I saved my username****rowser, but using the saved password ****oesn’t get through the website! looks

### Algorithm 1: Sanitizing Crash Report

**Data:** input (url, description)  
**Result:** anonymized input

1. initialization;  
2. lst = [x,y,z];  
3. open input;  
4. for <line in input> do  
5.     for i in lst do  
6.         regex = ('lst[i]\W+(\w+)');  
7.         match= findall(regex, line);  
8.         line = line.replace(match, "****");  
9.     end  
10. end
saved password ****rong!". It is not hard to guess that a considerable fraction of the descriptions may follow the same pattern and the outright filtering can result in the removal of insensitive data. We were also unable to provide a report on the frequency of such cases, since the process requires validation and manual comparison of the program’s output with the original report which would raise ethical concerns and may violate users’ privacy.

Moreover, it should be considered that the impact on the readability of the insensitive data remains negligible towards the final process of fixing bugs considering the huge number of crash reports received by the web browsers. In fact, many of the submitted reports (approximately 90%) are tagged as duplicates, and are not being processed [2, 36]. Further studies are required to examine the result of applying more advanced data sanitization techniques (e.g., [3, 20, 23, 26]) on the ACRS, including the use of natural language processing and deep learning. However, compared to our approach, applying these techniques may result in a total loss of a report when it is classified as sensitive and would be computationally expensive which may affect the system performance.

6 CONCLUSION

The plethora of data leakages has received a close review by the media and research communities. However, no previous study has been conducted in the context of browsers and the crash reporting systems. In this paper, we studied the disclosure of 2.5 million records of browser crashes that had been collected by one of the major browsers. In our inspection, we showed the potential privacy and security harm the current implementation of ACRS may pose. Our work presents a crucial preemptive step to raise community’s awareness of the security and privacy risks associated with the current implementation of ACRS. Further, we proposed a hotfix which masks the users’ sensitive data presented in the URL and description field without impacting the readability of the reports. Our proposed hotfix can be integrated with the current implementation of ACRS to safeguard users’ private data. While our proposed approach finely works toward protecting users private information, further studies are required to examine more advanced and intelligent techniques, including natural language processing and deep learning techniques.

REFERENCES

[1] Alessandro Acquisti, Allan Friedman, and Rahul Telang. 2006. Is there a cost to privacy breaches? An event study. ICIS 2006 Proceedings (2006), 94.
[2] Ilhekar Ahmed, Nitin Mohan, and Carlos J. Jensen. 2014. The Impact of Automatic Crash Reports on Bug Triaging and Development in Mozilla. In Proceedings of The International Symposium on Open Collaboration. ACM, 1.
[3] Ali Amiri. 2007. Dare to share: Protecting sensitive knowledge with data sanitization. Decision Support Systems 43, 1 (2007), 181–191.
[4] Apple. 2017. Analyzing Crash Reports. Available at: https://gooo/ft4uni. (2017).
[5] Francesca Bignami. 2007. Privacy and law enforcement in the European union: the data retention directive. Chi. J. Int’l L. 8 (2007), 233.
[6] Jean-François Blanchette and Deborah G Johnson. 2002. Data retention and the panoptic society: The social benefits of forgetfulness. The Information Society 18, 1 (2002), 33–45.
[7] Pete Broadwell, Matt Harren, and naveen Sastry. 2003. Scrath: A system for generating secure crash information. In Proceedings of the 12th conference on USENIX Security Symposium-Volume 12. USENIX Association, 19–19.
[8] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. 2012. An Evaluation of the Google Chrome Extension Security Architecture. In USENIX Security Symposium. 97–111.
[9] Miguel Castro, Manuel Costa, and Jean-Philippe Martin. 2008. Better bug reporting with better privacy. ACM SIGARCH Computer Architecture News 36, 1 (2008), 319–328.
[10] Sandra J Cereola and Ronald J Cereola. 2011. Breach of data at TJX: An instructional case used to study COSO and COBIT, with a focus on computer controls, data security, and privacy legislation. Issues in Accounting Education 26, 3 (2011), 521–545.
[11] Catherine Crump. 2003. Data retention: privacy, anonymity, and accountability online. Stanford Law Review (2003), 191–229.
[12] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. 2014. The Tangled Web of Password Reuse. In NISSS. Vol. 14. 23–26.
[13] Enisa. 2013. Recommendations for a methodology of the assessment of severity of personal data breaches. Available at: https://goo.gl/gjyG7H (2013).
[14] FireFox. 2010. Breakpad. Available at: https://wiki.mozilla.org/Breakpad (2010).
[15] Gemalto. 2017. Top Scoring Data Breaches. Available at: http://breachlevelindex.com/top-data-breaches. (2017).
[16] Google. 2017. Crash Reports. Available at: https://www.chromium.org/developers/crash-reports. (2017).
[17] Google. 2018. Chromiumium. Available at: https://chromium.google.com/. (2018).
[18] Luke Harsel. 2017. Why You Should Move Your Site to HTTPS: SEMrush Data Study. Available at: https://goo.gl/e14MSY. (2017).
[19] Collin Jackson, Andrew Borts, Dan Boneh, and John C. Mitchell. 2006. Protecting browser state from web privacy attacks. In Proceedings of the 15th international conference on World Wide Web. ACM, 737–744.
[20] Bo-Li, Yevgeny Vorobeychik, Muqiu Li, and Bradley Malin. 2015. Iterative classification for sanitizing large-scale datasets. In Data Mining (ICDM), 2015 IEEE International Conference on. IEEE, 841–846.
[21] Divakaran Lignilal, Inook Sim, and Lara Khanna. 2009. How significant is human error as a cause of privacy breaches? An empirical study and a framework for error management. computers & security 28, 3 (2009), 215–228.
[22] MarketShare. 2017. Market Share Reports. Available at: https://www.netmarketshare.com/. (2017).
[23] Kato Murvile and Claude Turner. 2013. A comparative analysis of data privacy and utility parameter adjustment, using machine learning classification as a gauge. Procedia Computer Science 20 (2013), 414–419.
[24] Mozilla. 2017. Helping with crashes. Available at: https://support.mozilla.org/en-US/kb/helping-crashes/. (2017).
[25] Mozilla. 2017. Mozilla Crash Reporters. Available at: https://support.mozilla.org/en-US/kb/mozillacrashreporter. (2017).
[26] Stanley RM Oliveira et al. 2003. Protecting sensitive knowledge by data sanitization. In nul IEEE, 613.
[27] Opera. 2017. When Opera crashes. Available at: http://help.opera.com/Linux/12.10/en/crash.html. (2017).
[28] Opera. 2018. Google Chrome Privacy Whitepaper. Available at: https://www.google.com/chrome/privacy/whitepaper.html. (2018).
[29] James O’Toole. 2014. JPMorgan: 76 million customers hack. Available at: https://gooo/FtrLZ. (2014).
[30] Sasha Romanosky, Rahul Telang, and Alessandro Acquisti. 2011. Do data breaches disclosed by identity theft? Journal of Policy Analysis and Management 30, 2 (2011), 256–286.
[31] Kiavash Satvat, Matthew Forshaw, Feng Hao, and Ehsan Toreini. 2014. On the privacy of private browsing—a forensic approach. In Data Privacy Management and Autonomous Spontaneous Security. Springer, 380–389.
[32] W3 School. 2017. OS Platform Statistics. Available at: https://www.w3schools.com/browsers/browsers_os.asp. (2017).
[33] StatCounter. 2017. Operating System Share Worldwide. Available at: http://gs.statcounter.com/os-market-share. (2017).
[34] Richard Stiennon. 2013. Categorizing data breach severity with a breach level index. SafeNet Inc (2013), 3.
[35] Mike Ter Lourw, Jun Soon Lim, and Venkat N Venkatakrishnan. 2008. Enhancing web browser security against malware extensions. Journal in Computer Virology 4, 3 (2008), 179–195.
[36] Laura Thomson. 2010. Socorro: Mozilla AZs Crash Reporting System. Available at: https://blog.mozilla.org/webdev/2010/05/19/socorro-mozilla-crash-reports/ (2010).
[37] W3. 1999. ISO Security Considerations. Available at: https://www.w3schools.com/Browsers/default.asp. (2017).
[38] Elizabeth Weise. 2016. 360 million Myspace accounts breached. Available at: https://gooo/larnYM. (2016).
[39] Michael E Whitman. 2003. Enemy at the gate: threats to information security. Commun. ACM 46, 8 (2003), 91–95.
[40] Charles Cresson Wood and William W Banks. 1993. Human error: an overlooked but significant information security problem. Computers & Security 12, 1 (1993).
[41] Songqing Wu and Lin Zhao. 2015. Web Security: A Whistleblat Perspective. CRC Press.