Voice Privacy: Analyzing Trends and Patterns in Amazon Alexa Voice History

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Gabriel J. Simmons

Spring 2021

Technical Project Team Members

Matthew Hancock, *Undergraduate Student, SEAS, University of Virginia*

Danny Huang, *Asst. Professor, Dept. of Computer Science, New York University*

Tu Le, *Graduate Student, SEAS, University of Virginia*

Yuan Tian, *Asst. Professor, Dept. of Computer Science, University of Virginia*

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Yuan Tian, Department of Computer Science

Voice Privacy: Analyzing Trends and Patterns in Amazon Alexa Voice History

Matthew Hancock, Gabriel Simmons, Tu Le, Danny Huang, Yuan Tian
University of Virginia
{mgh3x, gjs3qd, tnl6wk, yt2e}@virginia.edu

***Abstract*—Virtual personal assistants (VPAs) offer incredible technological advancements that can aid less technologically-inclined individuals. However, they are rife with privacy concerns, not least of which includes VPAs being woken unintentionally and recording the user and potentially collecting personal data about them. This data could potentially be stolen through malicious, black-hat attacks, or simply sold off by the VPA service providers themselves to the highest bidder. Therefore, it is crucial that users are aware of these and other similar potential privacy risks presented by VPAs. This project was intended to raise user awareness of such issues by presenting them with a fast, easy-to-use browser extension which scrapes their entire voice history and audio recordings and can download it directly to the user's computer, something which is otherwise quite challenging for the average user. By doing this, users seeing the large amount of data, including potentially large amounts of unintended audio recordings, will have an increased awareness of the risks associated with VPAs. After a large development period, a working browser extension was created which fulfilled these goals in a timely and minimally CPU-intensive fashion.**

***Index Terms*—VPA, IoT, JavaScript, Amazon, Alexa, Browser Extension**

I. Introduction

The integration of Internet of Things (IoT) virtual personal assistants (VPAs), which "listen" to users' voiced commands and can execute various audio operations, presents many new and exciting opportunities to improve the lives of countless people [14]. VPAs can significantly increase searching efficiency, quality of decision making, and boost the e-commerce economy by simplifying the purchasing process [3]. Moreover, VPAs lower the bar of required technological skills to operate – one needs only to give a command to control it, after all [3]. This has exciting implications for elderly and cognitively impaired individuals [15]. However, in the few short years since the appearance of such devices, many public concerns have arisen. There are economic concerns [1] [7], ethical and legal concerns [4], and security threat concerns. In the latter case specifically, there have been many demonstrations of malicious skills (skills are applications or processes for VPAs) bypassing a VPA's security policies [2].

Despite all of these issues, many users are not aware how easily their privacy can be compromised. Dubois et al. revealed that even something as simple as a loud TV program can wake VPAs, causing them to record your conversations without your knowledge for up to 10 seconds [6]. Zhang et al. brought to light two novel adversarial attacks in their 2018 paper which require relatively little technical expertise to carry out [16]. If attackers got a hold of these unwanted and private recordings of users giving away personal information like passwords, devastating results could follow. However, it is not only malicious attacks for which users must be aware – even the companies running the VPA's services themselves collect user information with the intention of selling it to advertisers, including personal data that users may have accidentally disclosed [12]. Often, the user is not even aware that this is happening.

It is this lack of user recognition that drove this project, which sought to increase users' awareness. However, a more accurate definition of "awareness" is needed. Endsley [8] defines a theoretical model of user situational awareness which can be applied to the privacy risks VPAs present [12]. It consists of three levels: perception, comprehension, and projection. Level one, perception, is simply the understanding of what certain components do or represent in a system [8]. For a user of VPAs, this might refer to a user's observation that every time they say "Alexa," their VPA "listens" and responds to their interaction accordingly. Level two, comprehension, refers to a more general understanding of how different elements, such as the use of a wake word, work together to produce an outcome [8]. For example, a user might combine their knowledge of the fact that when they say "Alexa," the VPA "listens" with the insight that VPAs use manipulable voice recognition that will often "mishear" one word as another. Putting these together allows one to come to the conclusion that VPAs might "hear" information not intended for it. Finally, in level three, projection, one applies their comprehension of a system to predict future outcomes [8]. A user might anticipate that because VPA providers probably want to maximize their profits, they might use personal data collected through their service to maximize their profit, selling it to the highest bidder. Worse, attackers could find this data (which is especially possible given the large security holes contained in VPAs [2]) and exploit it in an even worse way than an advertiser. It is at this point that the user is fully aware of the risks of using VPAs.

As such, this technical project sought to provide users a way to track and interact with their personal data such that they can visualize the pattern of VPA behavior, with the ultimate goal of moving uninformed users from the perception level to the projection level. To do this, it was determined that a Google Chrome browser extension would be created to help monitor user interactions with Amazon's Alexa Voice Assistant technology. The short-term goal of the extension

was to provide users with the capability to manage their voice history such that they can quickly download it, and identify unintended or unwanted data. Making users aware of these data grants users the opportunity to delete interactions that they otherwise would not known existed. This extension also offers an indirect user benefit – they will gain knowledge of their own usage patterns, how much Alexa "hears," and potentially what triggers unwanted interactions. In the longer term, users will be more mindful of what they say around their VPA, which will have the effect of protecting their personal data [10].

II. Related Work

Huang et al. [9] developed and released the IoT Inspector, a tool allowing users to monitor web-traffic from smart home devices on their home networks. They found that many smart home devices, including those made by Amazon and Google and smart TVs from at least 10 vendors, send unencrypted traffic using outdated TLS versions. Additionally, some of this network traffic is sent to advertisers and tracking services, and some of it is sent to Internet Services in countries with potentially poor privacy advocates. The research team will release the IoT Inspector data to the public to assist future research.

Dubois et al. [6] studied how spoken words from television could accidentally trigger smart speakers. They worked in a controlled environment, using 134 hours of Netflix content on the Google Home Mini, Apple Homepod, Harmon Kardon Invoke by Microsoft, and several Amazon Echo Dots. They found that the Invoke and Echo were set off the most with 0.40 activations per hour, followed by the Homepod with 0.38 activations per hour. Notably, *The West Wing* set off the Google Home Mini 0.95 times per hour. Accidental triggers were recorded for up to 10 seconds at a time. Schönherr et al. [13] conducted a similar study, producing artificial triggers with a pronouncing dictionary and a weighted, phone-based Levenshtein distance, while also exploring gender and language biases in accidental triggers. They will publish a dataset with over 1000 accidental triggers to assist future research.

Malkin et al. [11] collected opinions from smart speaker users. They surveyed 116 owners of Amazon and Google speakers based on randomly selected recordings of saved interactions with their devices. They found that almost half of the participants didn't know their recordings were being stored, and very few deleted any. Even if their own data wasn't particularly sensitive, they were unhappy with how often their devices recorded children and guests and disliked Amazon and Google's policies on permanent data retention.

Cheng et al. [5] looked at Amazon's system for vetting third party skills. They managed to get 234 policy-violating skills certified, finding that the skill certification process is not proper or effective. They concluded that vulnerable skills exist on Amazon's skill store, putting users, especially children, at risk when using voice assistant services.

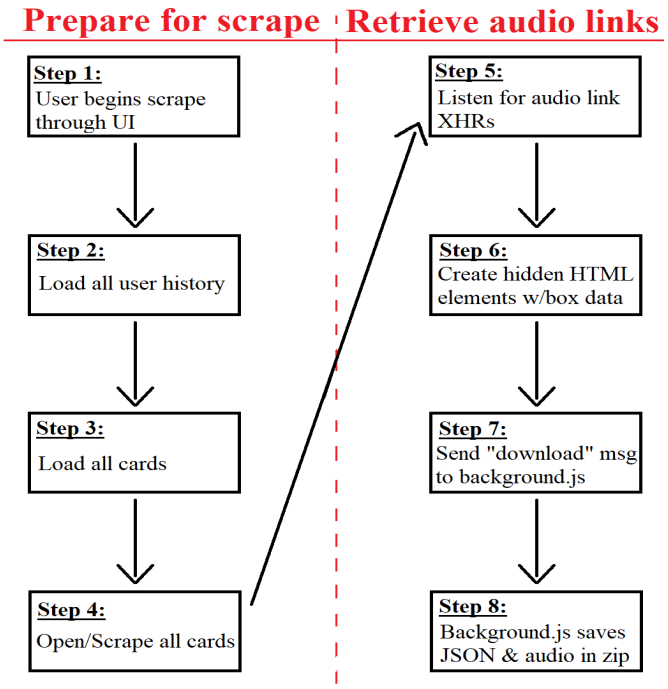

Fig. 1. A general flow of the steps described within this section to scrape user history data. We assume the user is already on the correct web page and is logged in.

III. Technical Approach: Extension Development

To help raise user situational awareness of the potential privacy issues of VPAs, these researchers decided to write a Google Chrome Extension in order to demonstrate to users just how much VPAs "hear" you speak, especially cases of unintentional audio captures. For this project, it was decided that the Amazon Alexa VPA (AAVPA) website would be scraped due to these researchers' familiarity with AAVPA and the clean, easy to use Amazon website which sorts each AAVPA interaction nicely into its own HTML element (Figure 2). This interaction data was scraped and condensed into a JSON file and, if the user requested it, every audio file for a given user could also be downloaded into a zip folder (which also contained the afore mentioned JSON). This is significant because while Amazon allows users to listen to previous audio data, it did not allow them to download it, restricting user interaction and perhaps downplaying the sheer volume of information they have on that user. By visually seeing the potentially large volume of (small) audio files and all of the JSON data collected on them, the idea was that previously uninformed or under-informed users would begin to understand potential privacy issues and move closer towards level three of Endsley's [8] model.

Of course, on top of the stated goals of this research, the extension was expected to be reasonably fast, and minimally CPU-intensive. Data regarding the speed and CPU-intensity was collected systematically to demonstrate these expectations.

A Google Chrome Extension was written mainly in the

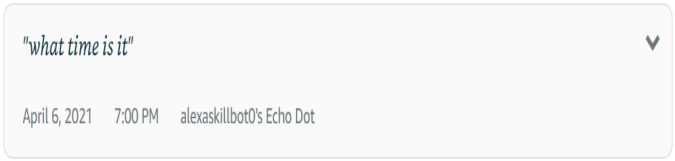

Fig. 2. This is how each Amazon Alexa VPA interaction is represented on Amazon's "Review Voice History" web page when the drop down button in the top right of the box is not clicked – this is a "closed" box.

JavaScript programming language with a bit of HTML for formatting purposes. The main task for our code specifically was to scrape pertinent data off of a user's Amazon account. To do this, the extension redirects the user to their "Review Voice History" web page (RVHWP) on their Amazon account[1], where their entire history of voice commands can be found.

The app is composed of three main parts: the user interface (UI), the file downloading module (FDM), and the data scraper (DS). The UI is a very simple HTML pop-up which gives the user the ability to begin running the application after they accept the linked terms and conditions. By default, the terms and conditions will not be accepted to prevent any unwanted runs. See Figure 3 below.

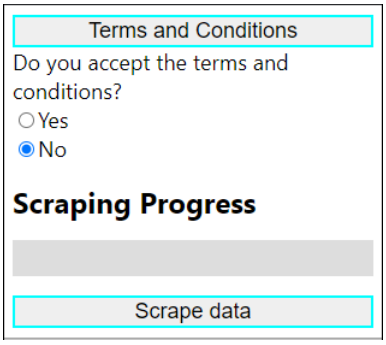

Fig. 3. The user interface

The FDM is part of the code which loads and eventually downloads all of the audio files to the user's individual computer. Utilizing the *JSZip* API[2], each audio file link was added into a file package, which was then zipped and downloaded using Chrome's Download API[3]. This part of the program will not run until the end of the DS, whereupon the DS code will indicate to the FDM that the audio and JSON files are ready to be downloaded.

Finally, there remains the DS. This was by far the most time consuming and CPU-intensive part of the extension. This whole process is represented in the main-scrape.js file, which alone contains 432 lines of code. The DS's purpose is to get the data from every interaction a user has had with AAVPA, contained in the afore mentioned HTML elements. (See Figure 2) As shown in Figure 4 below, each box contains the user's question or command in the bolder, black text (In Figure 4: "what time is it"), Alexa's response below that in gray text ("It's 7:00PM."), the date and time ("April 6, 2021" and "7:00 PM", respectfully), and the device name and type ("alexaskillbot0" and "Echo Dot", respectfully). It is important to note that boxes can necessarily contain more than one interaction. An interaction is defined as a single user command followed by a response by AAVPA. Thus, while Figure 4 shows a single-interaction box, boxes may contain multiple interactions between the user and AAVPA. This significantly complicated the development process, as will be shown later.

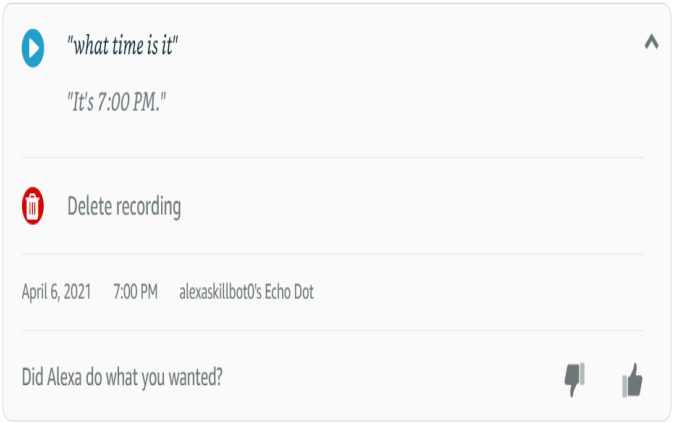

Fig. 4. This is how each Amazon Alexa VPA interaction is represented on Amazon's "Review Voice History" page when the drop down button in the top right of the box is clicked – this is an "open" box.

Another important point regarding the development process is the fact that, unlike many web pages, Amazon's RVHWP hides many HTML elements from view until the appropriate part of the web page is clicked. Using the afore mentioned boxes as an example, the Alexa response is impossible to scrape from the web page without clicking on the drop-down arrow in the top-right of the box. Once clicked, the Alexa response is displayed, and naturally becomes visible in the HTML, as well. However, the fact that is is not visible without clicking the box presented a problem which was later solved by clicking all of the boxes open in a loop. Similar solutions were employed in other similar instances.

With that out of the way, there were six major components of the DS, which will now be discussed in depth.

*1) **Viewing All History**:* By default, Amazon's RVHWP only displays a user's interactions with Alexa from the last 24 hours. However, the extension must scrape the entire user history. The process to access the entire history was relatively simple – it requires three clicks in the "Displaying" menu (Figure 5). Note that from here on, when it is said that clicks were needed by the extension to operate, this implies use of the JavaScript *click* function.

*2) **Loading All Cards**:* Once all history has been selected, Amazon does not automatically load *all* of the user's history. It will begin by only displaying the most recent 20 boxes. As the user scrolled down to the bottom of the page, the next 20 cards would load, and this process would continue until the entire history was displayed. Therefore, before the

[1]"Review Voice History" web page link: https://www.amazon.com/alexa-privacy/apd/rvh

[2]See JSZip's API page for more info: https://stuk.github.io/jszip

[3]See Chrome Downloads' API for more info: https://developer.chrome.com/docs/extensions/reference/downloads

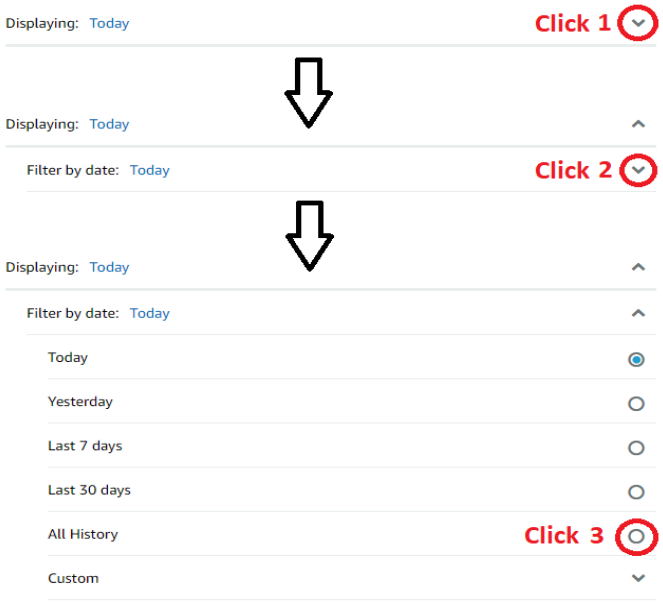

Fig. 5. The process for retrieving all user interactions from their entire history.

DS could commence, *all* cards had to be loaded. To do this, it was found that scrolling to the bottom of the page was unnecessary; clicking underneath the final displayed box on the page was enough to load the next 20 cards. Therefore, in a loop, the bottom of the page was clicked until all cards were loaded.

However, complicating the matter is that clicking an element at the bottom of the page continuously would take up a lot of CPU usage. Therefore, it made more sense to wait for the next 20 box elements to appear before clicking again. The code would check every 500 ms if they had appeared. If they had, another click was executed; if not, the program waited another 500 ms for the next check.

3) ***Open all cards/Scrape Data****:* This reasoning behind this part of the program has already been discussed. To recap, essentially the "closed" cards do not provide all interaction data, and the rest of the data is not visible in the page's HTML until the box is "opened" by clicking on the drop-down button in the top-right of the box. Therefore, in order to release this needed information, the cards were opened one-by-one in a loop until all were open. This is the most time-consuming portion of the code, as will be seen in the timing analysis in the results section. This is mostly for the same reason as the one presented in the previous step – waiting for elements. Since the interactions and audio links have to be collected in order, the next card cannot be opened until the previous one had been scraped; otherwise out-of-order data was risked.

Once a given card is open, before moving on to the next card the current card's data is scraped and stored in a JavaScript (JS) object array. When the scraping is complete, this array of JS objects will comprise the final JSON file. Hearkening back to the earlier conversation regarding the fact that boxes can contain more than one interaction, it is important to note that each JS object represents one *interaction*, not one box. Therefore, within each box scrape, the interactions are looped through. This operation is somewhat complicated by the fact that it is possible for multiple Alexa responses to appear back-to-back, both seeking to answer the user command. Therefore, it was determined that any consecutive Alexa responses would be combined into one string and treated as one response. There were other similar odd cases like this that had to be handled, but success was eventually achieved in correctly splitting the box into individual interactions.

4) ***Listen to XMLHttpRequests (XHRs) for audio links****:* One of the main stated goals for the project was the ability to download one's own audio files from Amazon's servers. However, Amazon seemingly purposely hid these files and did not place them *directly* on the RVHWP. Instead, they put a blue "play" button to the left of each recorded user audio. This play button sends a XHR request to Amazon's servers which returns the audio of the file. However, unfortunately, these links had certain hashed values which made link recreation from the page-present data impossible. Thus, the only way to get these audio links was to actually intercept the appropriate XHR requests and wait for a response from Amazon's servers with the correct audio link.

Thankfully, after a few days of research, a way was found to intercept this data. These researchers found that that an XHR response was received every time a box was opened which contained the audio file code that, when appended to the base URL *https://www.amazon.com/alexa-privacy/apd/rvh/audio?uid=*, returned the intended audio file. Therefore, no additional clicking was necessary since the cards already had to be opened one-by-one anyway. To actually perform the XHR interception, an XHR override script was written; this script was directly injected into the web page, allowing it access to the XHR response data sent by Amazon. Of course, other various XHRs were happening besides the targeted audio ones. In order to filter those out, a simple if statement checking the URL of the request was used. Finally, once the Amazon response was received, two fields were returned in JSON format – the audio code to be appended to the end of the above URL, and a binary field indicating whether the audio was playable or not. The latter field is significant because for an unknown reason, not all captured audio recordings from users are playable, which is odd considering that all "non-playable" audio have associated audio links. This field was checked in the program to ensure empty or non-existent files were not attempted to be downloaded.

As implied last paragraph, this whole process happens concurrently with the data scraping portion of the code. To keep the audio links in order such that they can each be associated with the correct interaction, each audio response was waited for in a similar fashion as the "load all history" section describes. Once the response was received and the hidden page element was created (see the following section), the okay was given to advance to the next box.

There is but one downside of the otherwise elegant solution of directly injecting an XHR listener into the RVHWP; it could

not directly pass the response data into the DS content script code. A creative solution was required in order to pass this data over to the DS content script, as a result.

*5) **Create hidden HTML elements to scrape link data**:* Because the DS content script did not have access by default to the XHR responses, the intercepted audio links were stored in hidden elements at the bottom of the web page, where the content script could later collect them and associate them with the correct interaction. This was usually a very simple process; however, there were odd cases to consider, mainly boxes that had more than one interaction. XHR responses for boxes containing more than one audio file were complicated in that rather then sending multiple XHRs for each different audio file, only one request and response were sent. The response contained *all* audio links in the box, separated by commas and brackets. This issue was resolved with a for loop that iterated through all user commands in a box with a corresponding audio link and associated the correct one using two different counters. To scrape each individual audio link from the hidden element, some basic string manipulation was performed to split the string into an array of audio link elements.

Note that while the audio links were stored in the boxes and each box checked for this link, the audio file was *not yet downloaded*. That happens in the FDM at the very end of the DS process.

*6) **Send message to save files**:* Finally, after iterating through all boxes on the RVHWP, the content script which runs the DS code sends a message to the background file telling it to fire the FDM and download the JS objects passed in by the DS as a JSON file as well as all the audio links which were stored in that same JS object. For user clarity, each audio file was named using its date and time, as well as the first 20 characters of the user's command. (The file name would never terminate in the middle of a word – it was possible to include past the first 20 characters if the last word took it beyond that point.) After the file was downloaded, the program terminated.

IV. Testing

Tests were performed on our extension's code in order to check how runtime and resource management were affected by machine specifications and different aspects of individual accounts, most notably the overall number of audio interactions.

	Machine 1	Machine 2
Operating System	macOS Big Sur 11.2.3	Windows 10 v. 10.0.19041
CPU brand	Intel	Intel
CPU model	1.4 GHz Quad-Core Intel Core i5	2.60 GHz Dual-Core Intel Core i5
Chrome Version	89.0.4389.128	89.0.4389.128
SSD or HD	SSD	SSD

TABLE I
TECHNICAL SPECIFICATIONS FOR MACHINES 1 AND 2

Testing was performed on two different machines. Machine 1 ran macOS 11.2.3 on a quad-core Intel Core i5 and Machine 2 ran Windows 10 on a dual-core Intel Core i5 (Table I).

Three different Amazon accounts were used for testing. Machine 1 performed tests on accounts 1, 2, and 3. Machine 2 performed tests on accounts 2 and 3. Account 1 had 55 Alexa interactions, account 2 had 81, and account 3 had 24 (Table II).

Account	One	Two	Three
Number of Boxes/Cards	55	81	24
Number of Interactions	55	82	30
Number of Audio Files Downloaded	55	80	26
Avg. Size of Audio File	64.35 KB	64.95 KB	53.88 KB
Std. Dev of Size of Audio Files	37.72 KB	35.99 KB	13.99 KB
# of Intentional Audio Files	42	58	25
Avg. Size of Intended Audio Files	75.55 KB	76.69 KB	53.52 KB
Std. Dev of Size of Intended Audio Files	36.22 KB	35.19 KB	14.15 KB

TABLE II
DETAILS REGARDING NUMBER OF AUDIO INTERACTION AND FILE SIZES FOR EACH ACCOUNT. INTENTIONAL AUDIO FILES ARE ONES THAT ARE NOT LABELLED AS "AUDIO COULD NOT BE UNDERSTOOD" OR "AUDIO NOT INTENDED FOR ALEXA"

Account 1 and 2 had similar average sizes and standard deviations for audio files. Account 3's audio files were slightly smaller in size, and the standard deviation was smaller. Variations in mean and standard deviations for file sizes were affected more by how the user spoke to Alexa than any other factor. Account 3 had shorter interactions while accounts 1 and 2 had similar length audio interactions. Almost all unintentional recordings ("Audio not intended for Alexa" or "Audio not understood") had a file size of 29 Megabytes, making them easy to spot. Removing unintentional audio recordings brought the average audio file size up by 10 Megabytes on accounts 1 and 2 and lowered the standard deviation slightly. There was no significant change for account 3 when removing unintentional recordings.

Machines 1 and 2 both saw an average CPU usage of 9% for the duration of the extension's runtime. CPU usage was highest when loading all history on the voice history webpage,ranging from 14-20%. It was lowest while loading cards, ranging from 2-5%. When opening cards, CPU usage ranged from 8-12%. Machine 1 did not see any variation in memory usage while running the extension. Machine 2's memory usage would generally be 20-80 Megabytes more than idle when running the extension. CPU and memory usage were not affected by the number of audio interactions on an account.

	Machine 1	Machine 2
Account 1 Mean	26.219	N/A
Account 1 SD	0.560	N/A
Account 2 Mean	32.899	35.478
Account 2 SD	1.405	2.549
Account 3 Mean	8.201	9.423
Account 3 SD	1.264	0.512

TABLE III
MEAN AND STANDARD DEVIATION FOR OVERALL RUNTIME FOR ALL ACCOUNTS ON BOTH MACHINES

As the number of audio interactions increased, mean runtime and standard deviation increased (Table III). Account 2, which had the most audio interactions, had the longest overall runtimes on average. Account 3, which had the least interactions, had the shortest overall runtimes. Account 1's number of audio interactions was nearly right between accounts 2 and 3, but its average overall runtimes were only a bit shorter than account 2's. Machine 2 generally had longer runtimes and longer standard deviations, which was likely due to the difference in machine specs. A single outlier on machine 1's trials with account 3 did change the standard deviation and put it above account 2's. A breakdown of runtimes for all steps of the extension may be found in the appendix.

The extension was found to prioritize CPU usage. Machine 1 had a slightly more powerful CPU than machine 2, resulting in slightly faster runtimes. Machine 1's more powerful CPU may have also accounted for the lower memory usage. The most important factor was the number of boxes. More audio boxes would increase the runtime of all steps of the extension. Users are recommended to delete audio interactions as they are downloaded to increase performance of the extension.

V. Discussion

Currently, Amazon does not seem to have any plans to make managing voice history simpler, and voice data continues to be stored on their servers indefinitely. For users who wish to see their personal data but keep it away from Amazon, the extension's relatively quick runtimes will prove to be beneficial. Similar names and filesizes for unintentional audio interactions allow users to quickly see how much Alexa recorded that it shouldn't have.

Several challenges were encountered in the creation of the extension. The research team initially tried to use selenium, a coding library used for interacting with web browsers, to scrape the DOM data. It was found that selenium was not needed, for scraping the DOM data could be done entirely with internal JavaScript functions. Once the necessary methods and elements of Amazon's voice history page were found to scrape the DOM data, the next challenge was working around the way the webpage worked. Elements would not appear as soon as Amazon's webpage was loaded, they would instead appear after a short time as the webpage got set up. The extension would not run if the elements were not present. Additionally, if the user wanted to scrape all audio data, they would have to repeatedly need to scroll to the bottom of the webpage to load all audio interactions. To solve the first challenge with the webpage, the extension was instructed to wait a few milliseconds before beginning to scrape data, allowing all elements to load in. The research team found that although there was no way for users to click any button to load more audio interactions, there was a clickable element in the webpage's HTML allowing the extension to load more interactions. This step was added to the scraping process.

Some edge cases in audio interactions would cause problems with the downloaded JSON file. These edge cases included multiple audio interactions in one card, interactions where the user did not speak, and cards where several interactions had the user saying the exact same thing (i.e. "no" several times). The first edge case would cause an error in the extension and halt the scraping process. It was resolved by appending all Alexa responses to each other before moving onto another card. The second edge case would result in no transcript of the interaction being downloaded. The research team added a conditional statement to the extension's code which would proceed with scraping a transcript if it encountered an empty user response. The final edge case was resolved by allowing JSON data and audio recordings to have the same name, so multiple recordings of the user saying the thing were saved.

The final challenge the research team encountered was a bug which would associate audio files with the wrong audio interaction. It appeared to be a result of the issue with multiple audio interactions in one card, causing the associations between audio files and interactions to be off by one. It was resolved with the fix for multiple audio interactions.

Future improvements to the extension include UI enhancements and runtime improvements. The UI is currently a bit barren and lacks feedback, so a progress bar to report which step the extension is in is in the works. Runtime improvements in progress involve scraping audio cards as they appear rather than loading every card before scraping. This runtime improvement comes with the benefit of allowing the user to specify an audio card to stop scraping at, negating the need to scrape all audio interactions with each use of the extension.

Future research teams will expand on the chrome extension by sending users' downloaded voice data to a server (with user consent) for further analysis. Voice data will be deleted when it is not needed anymore. The extension will be released on the chrome web store for download by the public. Using data from extension users, the research team will investigate the voice data to find how individual users may accidentally trigger voice interactions and how they can be prevented. The findings will be published so Amazon may see how to better protect its users' data. Additionally, there is potential for this work to be extended to other VPA platforms, such as Google Assistant and Microsoft Cortana, as both of those VPAs also have similar voice history interfaces as Amazon Alexa.

VI. Conclusion

In this work, we went over the creation of a tool for downloading voice history from Amazon Alexa. Using a Google Chrome extension, we were able to download audio recordings and JSON files containing transcripts and other data for each user interaction with Alexa. We tested functionality of the extension to look for ways it can be improved in the future. The delay between entering the voice history page on Amazon and data being loaded proved to be challenging to this project, which may vary functionality of the extension for each user. By having access to their personal data without needing to keep it on Amazon's servers and being able to quickly see all unintentional recordings, many users of the extension will be brought from Endsley's [8] level one of user

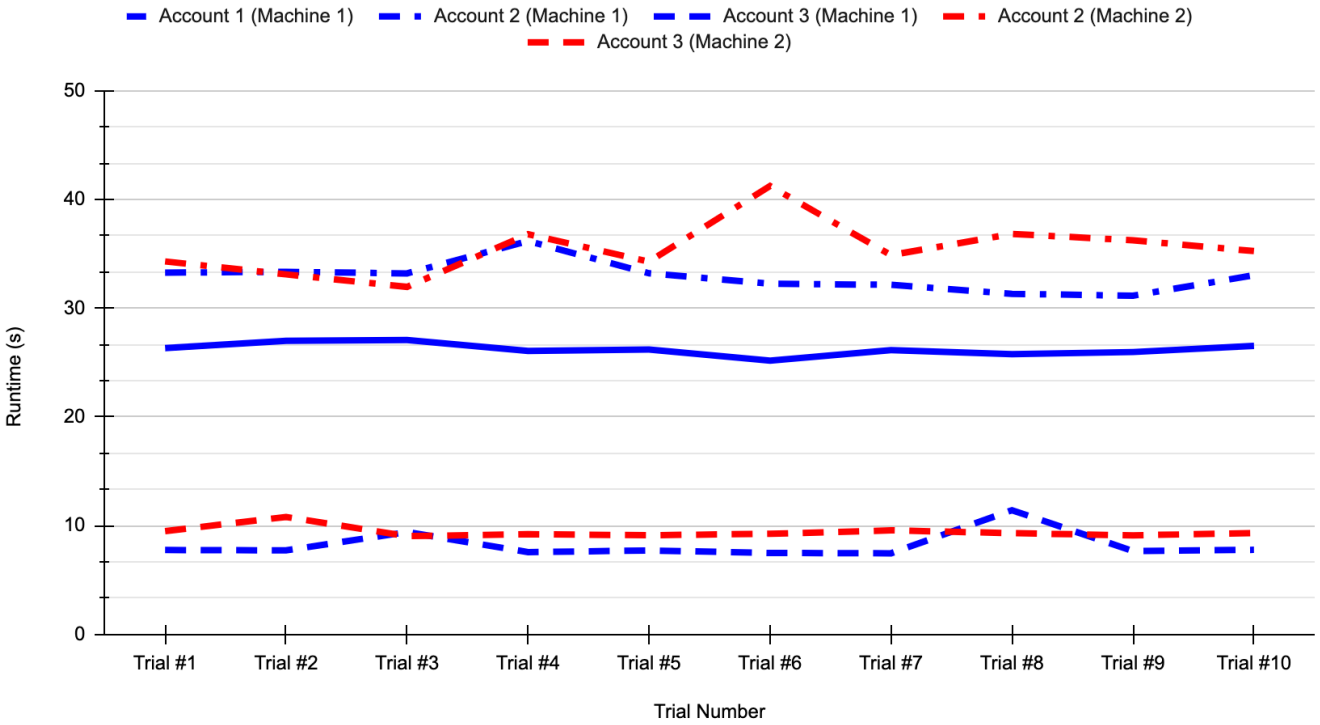

Fig. 6. Overall run times for all accounts on both machines

situational awareness to at least a level two. Future research on this project will further advance user awareness. Amazon's data retention policies may be forced to change in the future as users become more aware and protective of their privacy.

References

[1] A. Acquisti, C. Taylor, and L. Wagman, "The economics of privacy," *Journal of Economic Literature*, vol. 54, no. 2, pp. 442–492, June 2016. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/jel.54.2.442

[2] F. Bräunlein and L. Frerichs, "Smart spies: Alexa and google home expose users to vishing and eavesdropping," Security Research Labs, December 2019. [Online]. Available: https://srlabs.de/bites/smart-spies

[3] O. Budzinski, V. Noskova, and X. Zhang, "The brave new world of digital personal assistants: Benefits and challenges from an economic perspective," *NETNOMICS: Economic Research and Electronic Networking*, vol. 20, no. 2, pp. 177–194, 2019.

[4] A. J. Campbell and L. Barrett, "In the matter of request for investigation of amazon, inc.'s echo dot kids edition for violating the children's online privacy protection act," Letter to Federal Trade Commission, Counsel for Campaign for a Commercial Free Childhood & Center for Digital Democracy, May 2019. [Online]. Available: https://www.echokidsprivacy.com

[5] L. Cheng, C. Wilson, S. Liao, J. Young, D. Dong, and H. Hu, "Dangerous skills got certified: Measuring the trustworthiness of skill certification in voice personal assistant platforms," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1699–1716. [Online]. Available: https://doi.org/10.1145/3372297.3423339

[6] D. J. Dubois, R. Kolcun, A. M. Mandalari, M. T. Paracha, D. Choffnes, and H. Haddadi, "When speakers are all ears: Characterizing misactivations of iot smart speakers," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 4, 2020. [Online]. Available: https://par.nsf.gov/biblio/10192512

[7] S.-A. Elvy, "Paying for privacy and the personal data economy," *Colum. L. Rev.*, vol. 117, no. 6, pp. 1369–1460, October 2017.

[8] M. R. Endsley, "Toward a theory of situation awareness in dynamic systems," *Human factors*, vol. 37, no. 1, pp. 32–64, 1995.

[9] D. Y. Huang, N. Apthorpe, F. Li, G. Acar, and N. Feamster, "Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 4, no. 2, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3397333

[10] E. Kritzinger and S. H. von Solms, "Cyber security for home users: A new way of protection through awareness enforcement," *Computers & Security*, vol. 29, no. 8, pp. 840–847, 2010.

[11] N. Malkin, J. Deatrick, A. Tong, P. Wijesekera, S. Egelman, and D. Wagner, "Privacy attitudes of smart speaker users," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 4, 2019.

[12] A. McCarthy, B. R. Gaster, and P. Legg, "Shouting through letterboxes: A study on attack susceptibility of voice assistants," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 2020, pp. 1–8.

[13] L. Schönherr, M. Golla, T. Eisenhofer, J. Wiele, D. Kolossa, and T. Holz, "Unacceptable, where is my privacy? exploring accidental triggers of smart speakers," 2020.

[14] N. Turner-Lee, "Can emerging technologies buffer the cost of in-home care in rural america?" *Generations*, vol. 43, no. 2, pp. 88–93, 2019.

[15] R. Yaghoubzadeh, M. Kramer, K. Pitsch, and S. Kopp, "Virtual agents as daily assistants for elderly or cognitively impaired people," in *International workshop on intelligent virtual agents*. Springer, 2013, pp. 79–91.

[16] N. Zhang, X. Mi, X. Feng, X. Wang, Y. Tian, and F. Qian, "Understanding and mitigating the security risks of voice-controlled third-party skills on amazon alexa and google home," *arXiv preprint arXiv:1805.01525*, 2018.

VII. Appendix

Account 1	RT - Load Hist.	RT - Load Cards	RT - Open Cards	RT - Misc.	RT - Total
Trial #1	2.12	6.72	17.28	0.2	26.32
Trial #2	1.75	7.73	17.34	0.17	26.99
Trial #3	1.69	8.25	16.9	0.23	27.07
Trial #4	1.68	7.24	16.98	0.17	26.07
Trial #5	2.46	7.24	16.34	0.15	26.19
Trial #6	1.58	7.25	16.17	0.17	25.17
Trial #7	1.68	6.76	17.53	0.17	26.14
Trial #8	1.69	7.24	16.67	0.16	25.76
Trial #9	1.59	7.3	16.85	0.22	25.96
Trial #10	1.68	7.7	16.88	0.26	26.52
Account 2	**RT - Load Hist.**	**RT - Load Cards**	**RT - Open Cards**	**RT - Misc.**	**RT - Total**
Trial #1	2.19	10.92	19.85	0.3	33.26
Trial #2	1.92	10.53	20.67	0.21	33.33
Trial #3	1.81	10.94	20.16	0.28	33.19
Trial #4	1.82	10.9	23	0.44	36.16
Trial #5	2.15	9.93	20.94	0.18	33.2
Trial #6	1.74	10.9	19.4	0.2	32.24
Trial #7	1.78	10.03	20.06	0.27	32.14
Trial #8	2.07	9.53	19.37	0.34	31.31
Trial #9	1.74	10	19.14	0.26	31.14
Trial #10	2.16	9.87	20.79	0.2	33.02
Account 3	**RT - Load Hist.**	**RT - Load Cards**	**RT - Open Cards**	**RT - Misc.**	**RT - Total**
Trial #1	1.02	1.05	5.67	0.02	7.76
Trial #2	0.97	1.07	5.66	0.03	7.73
Trial #3	2.57	1.03	5.79	0.02	9.41
Trial #4	1.05	1.06	5.43	0.03	7.57
Trial #5	1.09	1.06	5.55	0.02	7.72
Trial #6	0.94	1.05	5.47	0.03	7.49
Trial #7	1.12	1.05	5.25	0.04	7.46
Trial #8	3.33	1.06	7	0.03	11.42
Trial #9	0.85	1.06	5.73	0.03	7.67
Trial #10	0.96	1.05	5.74	0.03	7.78

TABLE IV
These are the raw run times for all ten trials for accounts 1, 2, and 3 performed on machine 1.

Account 2	RT - Load Hist. (s)	RT - Load Cards (s)	RT - Open Cards (s)	RT - Misc. (s)	RT - Total (s)
Trial #1	2.91 s.	10.77	23.32	0.20	34.29
Trial #2	2.44 s.	10.09	22.66	0.35	33.10
Trial #3	2.62 s.	10.17	21.59	0.19	31.95
Trial #4	3.07	10.87	22.63	0.22	36.79
Trial #5	2.57	9.62	21.88	0.20	34.27
Trial #6	7.94	10.72	22.46	0.10	41.22
Trial #7	2.95	9.75	21.98	0.19	34.87
Trial #8	2.67	11.66	22.30	0.19	36.82
Trial #9	2.81	10.69	22.53	0.20	36.23
Trial #10	2.53	10.29	22.23	0.19	35.24
Account 3	**RT - Load Hist. (s)**	**RT - Load Cards (s)**	**RT - Open Cards (s)**	**RT - Misc. (s)**	**RT - Total (s)**
Trial #1	1.83	1.06	6.58	0.03	9.50
Trial #2	1.90	1.05	7.82	0.03	10.80
Trial #3	1.75	1.05	6.20	0.04	9.04
Trial #4	1.75	1.09	6.34	0.03	9.21
Trial #5	1.77	1.07	6.25	0.03	9.12
Trial #6	1.84	1.07	6.31	0.03	9.25
Trial #7	2.22	1.08	6.22	0.05	9.57
Trial #8	2.01	1.07	6.20	0.03	9.31
Trial #9	1.78	1.07	6.22	0.04	9.11
Trial #10	1.68	1.04	6.56	0.04	9.32

TABLE V
RUNTIMES FOR ALL STEPS ON ALL ACCOUNTS TESTED ON MACHINE 2
*NOTE THAT ACCOUNT 1 WAS A PERSONAL ACCOUNT MAINTAINED BY THE OPERATOR OF MACHINE 1. MACHINE 2'S OPERATOR WAS NOT ALLOWED ACCESS IT FOR SECURITY REASONS, AND THUS SKIPPED ACCOUNT 1 IN TESTING.