“Alexa, hack…” Speech Recognition hackbots for the modern Cyber Warrior

Authors: Alexander Master, *CPT, U.S. Army*. John Fernandes, *CPT, U.S. Army.*

***Abstract*—The purpose of this project was to explore the possibility of using speech recognition to conduct hacking activities for us. To accomplish this, our study sought to create a code base that used digital assistants to run the tool nmap for network discovery and reconnaissance. This would serve as a proof of concept and framework for future endeavors. Ultimately, we were not completely successful because current digital assistants are limited in their implementation. However, our work conveys that hacking with speech recognition is certainly possible and lays a foundation for future advances in the field.**

I. Introduction

Voice recognition software, once only associated with science fiction, has permeated into the lives of everyday people. “Digital assistants” such as Apple’s Siri, Amazon’s Alexa, and Microsoft’s Cortana, perform tasks for people on cell phones, tablets, computers, and even dedicated assistant speaker devices. They provide a “more human” interaction with the device, using human speech to run commands rather than text or touch input. While speech recognition software has existed for decades, recent implementations in mobile devices have brought rudimentary implementations of the technology to the consumer market, and applications like Alexa and Cortana are easily recognized household names. Because of algorithmic and computing advances, programs recognizing speech have moved from a niche market to one with boundless and disruptive potential. Instead of being a focused on stenography and those with disabilities, it will be able to touch every facet of a person’s life.

People use digital assistants to conveniently execute simple, concrete tasks. The focus of this paper is exploiting this potential for hacking. While hacking is a nontrivial, nonlinear task, it often consists of subtasks that are both trivial and linear. Though simple, these subtasks consume time and energy. Digital assistants have the potential reduce this time and energy and allow hackers to focus on the difficult problems associated with hacking, such as “Can my bladder last until I finish debugging this code?,” and “Do I have enough Monster Energy drinks to last through this penetration test?”

II. Background

In order to tap the potential of digital assistants to assist hacking, we must understand the user interaction with the assistant and the tasks the user would seek to accomplish. To narrow the scope of our study, we focused on three implementations of digital assistants, Amazon’s Alexa, Microsoft’s Cortana, and Binary Fortress’s Voicebot. We focused on that tasks conducted using command line tools, and we used nmap as a representative sample.

In order to be effective, a digital assistant must assign meaning to sounds, derive intent from the meaning, and execute that intent. The applications we studied conduct this process in a similar manner. Alexa and Cortana use statistical algorithm to match sound to expected meanings. The words “eye” and “I” have the same sound, but different meaning. The programs select one meaning based on their assessment. These meanings are predefined as “utterances.” The applications map “utterances” to “intents” or “commands.” For instance, the utterances “what is the time?” and “tell me the time” would both indicate a request for the time. After determining which command is required, the digital assistant initiates the appropriate process(es).

The progression from sound to command is either stateful or stateless interaction. A dialog is a stateful transaction. Each line of speech is understood in terms of the ones which preceded it. The alternative, stateless interaction, is one which the digital assistant does not recall previous interactions. Digital assistants are generally stateless, and rely on additional programs to provide command execution and any stateful behavior.

Many hacking tasks are routine and systematic in nature. These tasks are automated using tools that run from the command line. When these tools launch from the command line, users provide a set of options according to a defined syntax. Therefore, to execute a hacking task, the digital assistant must select the desired set of options and run the tool with the correctly formatted parameters.

III. Analysis

A. *Project Goals*

The goal of this project is to develop a code base that executes nmap commands based on user verbal interaction, in order to provide a proof of concept and a modular framework for implementing additional hacking tools. Because digital assistants execute hacking tasks in a similar manner using their CLI, one tool is a sufficient for a proof of concept. We chose nmap because it is a popular tool with significant functionality. However, hacking requires an ever-changing variety of tools. Therefore, this code base should serve as the seed for a larger modular framework that is easily extended.

B. Alexa

Amazon's Alexa uses Amazon Web Services (AWS) and "skills" to conduct its interaction. Skills are applications that are loaded into a user's Alexa account. They include the mapping of utterances to intents and intents to "events". When a user interacts with their Alexa enabled device, the information is processed in the cloud based on the loaded skills. The skills dictate which internet-based service executes the command.

Development for Alexa is relatively simple, but extremely limited and not suited for hacking. First, interacting with the cloud adds greater exposure to the users hacking activity. The hacker's interactions, the skill, and the execution service are all at risk for identification and exploitation. Second, Alexa is not hardware agnostic. Although devices other than the Echo may use Alexa, it is a non-trivial task to enable them using sufficient authentication. Finally, the utterance specification is limited and the interaction is stateless which could result in significant delays as invalid user input is pushed through amazon to the execution service and back. Therefore, although Alexa is well suited for safe simple commands such as "what is the weather?" it is a n00b with regard to complex utterances for sensitive tasks. Because of these reasons, we shifted focus to another digital assistant, Cortana.

C. Cortana

Microsoft's Cortana is a digital assistant, available for Windows 10 and Windows Mobile platforms. It still uses external networks calls to recognize speech, but unlike Alexa, Cortana's speech recognition engine executes programs on the local machine. This minimizes the user's potential exposure and is more advantageous from a developer and security perspective. Applications can operate independently using Cortana utterances or can integrate with other preexisting software at the user's disposal. However, by design this latter task is not a default capability for Microsoft Applications.

In order to implement behavior with Cortana, we used Microsoft's Universal Windows Platform (UWP). This platform allowed us to develop an application that would work on all Window Operating Systems that use Cortana. We were able to define a number of commands that the user could call directly from the OS's Cortana interface. These commands either execute and bring the application to the foreground or execute in the background and display information on the Cortana interface (the start menu for desktops). Unfortunately, the UWP "sandboxes" its applications making it difficult to call external executables or execute commands with elevated privileges. The code necessary to circumvent these native protections was outside the scope of this project. Instead, our application demonstrates the ability to receive verbal commands and execute code on the local machine by loading and storing information. In order to successfully, execute commands in an external application, we turned to a lighter-weight and less proprietary piece of software called VoiceBot.

D. VoiceBot

Binary Fortress is a different kind of digital assistant, with a consumer focus solely on computer users rather than mobile implementations. VoiceBot markets itself particularly to PC gamers, to allow them to execute more UI inputs simultaneously than they could with simply their two hands on a keyboard and mouse. The Binary Fortress website claims, "VoiceBot lets you take command with your voice! Say commands out loud to send actions to your games and applications. Use your voice to type keyboard shortcuts, click and move your mouse, and execute macros and scripts."

Utilizing VoiceBot was our most successful implementation of speech recognition with nmap. VoiceBot translates spoken utterances to text and matches them to a predefined wordlists. Based on the wordlist, a script executes the appropriate nmap command. VoiceBox's significant advantage over Cortana and Alexa is its inherent ability to run Windows desktop scripts, as well as pass text-based command line arguments to programs. After VoiceBox had done the hard work of translating human speech and correlating it to a specific intent, it passes that intent word into a Python script that matches the intent to a specific nmap command and options, with a previously defined, target IP address. This implementation served us extremely well for demonstration purposes, showing that nmap use is certainly possible using voice commands. However, there are obvious downsides to this implementation. It is not OS or hardware agnostic, and it is not suitable for mobile applications. In its current form, the target IP addresses must be defined with text input before voice commands can be executed, due to the nature of current speech recognition software (i.e., validating IP address voice input, 192.168.0.1 being translated as "one nine two dot one six eight dot…" by speech recognition). Finally, this implementation has multiple dependencies on the hosting system, rather than being a singular piece of software that is easily executed on its own or modules added to increase functionality.

E. Dialog Interaction

Each of the three applications discussed use primarily stateless interactions. The Cortana and VoiceBot implementations introduce stateful characteristics because they execute locally and can use the local systems memory. However, neither program natively implements dialog interactions. To explore this capability, we developed a program from scratch using Python. Its design has two parts: an interface program and a dialog-handler or execution engine.

The function of the interface program is to parse input and pass valid "intents" to the execution engine. Unlike the other applications, utterance validation occurs with functions rather than being constrained to pattern matching. This allows for a more flexible and logic-based recognition. Our functionality includes recognizing valid IP addresses and mapping various analogous utterances to the same intent. For our proof of concept, we did not implement the full JSON-like utterances that are used by Alexa and Cortana. The second unique aspect is the interface’s use of utterances based on the dialog-handler.

The dialog-handler provides the interface with the set of valid intents. Previous utterances dictate the current options available to the user. This changes the interaction-model from a flat structure to one resembling a tree. Menu objects, which associated intents to actions, are the foundation of this behavior. When directed, the dialog-handler executes the specified nmap command and updates logs based on its actions.

Although this approach uses text input and output it demonstrates a dialog capability not observed in the other three approaches. As we begin considering future work, one option includes extending this program with text-to-speech (TTS) and speech-to-text (STT) tools or integrating it with one of the other approaches. However, a TTS plug-in would not be able to use the probabilistic determination of input based on expected value.

IV. Conclusion and future work

Ultimately, we were unable to provide a single framework robust enough to execute a significant portion of nmap’s functionality using a digital assistant. Our study discovered that the dominant digital assistants are limited in both their implementation for development and their integration with preexisting software. They have proprietary application programming interfaces (APIs), have rudimentary utterance mapping ability, and are not portable (because of OS or hardware). The development communities for these assistants are immature and driven by the assistant’s creators (i.e. Microsoft and Amazon). By developing programs to execute various portions of our goal, we have demonstrated the potential of digital assistants if these disparate parts are integrated or if the current implementations of digital assistants become more flexible.

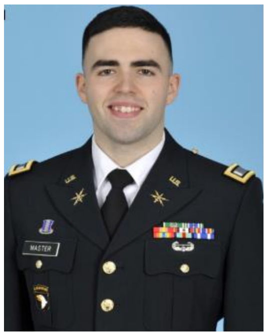
Our application and research serves as a starting point. Future work may extend any of our four avenues of study or add an additional avenue. Researchers who pursue securing their communications and execution service can leverage Alexa to execute tasks based on simple commands. Experienced Microsoft developers may make rapid gains by enabling Cortana to execute external applications (such as nmap). Researchers interested in open-source solutions may use TTS and STT programs to extend the dialog approach. Finally, future work might focus on technologies not covered in this study, such as Siri or Google speech recognition. Although we sought to explore the breadth of the possible, rather than pursue one constrained path, we barely scratched to surface of digital assistant’s ultimate potential for hacking. It will not be long before hackers are talking to their computers rather than merely yelling obscenities at them.

References

[1] R. Chen. (2017, May.). Microsoft CortanaVoiceCommand Samples. Available: https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/CortanaVoiceCommand

[2] (2017, May.). Binary Fortress VoiceBot. Available: https://www.voicebot.net/

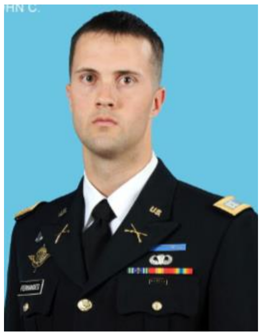
[3] (2017, May) Learn to Build with Alexa. Available: https://developer.amazon.com/alexa



Alexander Master was born in South Bend, IN in 1990. He received a B.S. in Computer & Information Technology – Network Engineering, from Purdue University in 2013.

From 2008 to 2011, he was an Information Technology Intern at Culver Military Academy in Culver, IN. From 2013 to 2015, he served as a U.S. Army Field Artillery Officer, at Fort Campbell, KY and Jalalabad, Afghanistan. He served as a Fire Support Officer and Chief Operations Officer in 3rd Brigade of the 101st Airborne Division. Since 2015, he has served as a Cyber Operations Officer at Fort Gordon, GA.

CPT Master is a Certified Information Systems Security Professional (CISSP), and has earned numerous other technical certifications from Industry leaders including (ISC)2, Cisco, VMware, SANS Institute, CompTIA, and EC-Council, as well as various technical and professional development courses from the U.S. Military.



John Fernandes was born in Portsmouth, VA in 1990. He received B.S. degrees in Mathematics and Computer Science from the United States Military Academy in 2012.

As part of his undergraduate education, he completed internships with Army Research Labs, the NSA, and the NRO. From 2013 to 2016 he served as an Infantry Officer in Vicenza, Italy. He served as a plans officer, platoon leader, and deputy logistics officer in 2-503rd IN Regiment of the 173rd Infantry Brigade Combat Team (Airborne). Since late 2016, CPT Fernandes has served as a Cyber Operations Officer at Fort Gordon, GA.

CPT Fernandes is a Certified Ethical Hacker (CEH) and has GCIA and GPEN certifications from the SANS Institute. His military certifications and training include the Expert Infantryman's Badge, Ranger Tab, and Parachutist's Badge.