Voice Biometric Integration

Product Requirements Document, Version 1

October 28, 2016

Team Tres Commas

Members:
Jonathan Easterman, Sasha Shams,
Arda Ungun, Carson Holoien, Vince Nicoara

Mentors:
Mike Weaver, Colin Kelley, James Brown
Chandra Krintz, Ankita Singh

Introduction

PROBLEM

We want to bring authentication to the 21st century. Existing methods of automated user authentication over telephony have many drawbacks; entering codes via touch-tone is cumbersome and error-prone, existing robo-callers are unreliable and repetitive.

VISION

We will utilize modern voice biometrics to identify and authorize users in a seamless process. No more having to remember years-old pins and passwords. No more having to share private, sensitive information with strangers over the phone as you answer security questions. Using our technology, an individual's identity will be ascertained in the utterance of a sentence.

Implementing voice authentication will improve convenience and user experience for customers, while automation will be cheaper and more efficient for businesses. Authenticating a user can be 100% autonomous, which will speed up the process for users and cut down the amount of needed for call center employees.

COMPETITION

There are many APIs that offer a voice biometrics service via the Internet, but there is only one company that offers a complete voice authentication service. Nuance offers voice authentication as a costly and monolithic product[1]. We plan to offer an open-source, configurable product that will target smaller companies.

INNOVATION

Our solution is secure, lightweight, easy to deploy, and modular. We will offer improved security by integrating automated voice authentication, and target small- to mid-level customers through our lightweight architecture and ease of deployment. The plug-and-play nature of our application allows other developers to leverage our product to the services that suit their needs. We implement Microsoft's voice authentication API, but future developers can choose to replace that portion of our project with an improved or proprietary solution instead.

[1] We tried contacting Nuance about their product and potential support for developers and we have not heard back from them yet.

Project Specifics

OBJECTIVES

Our project aims to add another layer of security to Invoca’s call systems by adding voice authentication and shorten the amount of time spent by employees authenticating users.

BACKGROUND

In existing solutions, when users call to a make changes to one of their accounts, they must verify their identity by answering a series of security questions. These security questions can include things like their social security number and other personal questions. Going about verification in this manner has a few issues that can be improved upon. First of all, it can take a long time – maybe the user does not remember their specific account information (“What is your favorite animal?”). Secondly, it is not 100% secure – if a nefarious character (or even the individual on the other end of the call) got ahold of their personal information, they can access the user’s account and wreak havoc. Our voice authentication vision seeks to provide another layer of security as well as make the entire process quicker for the user and ultimately improve efficiency for call center administrators.

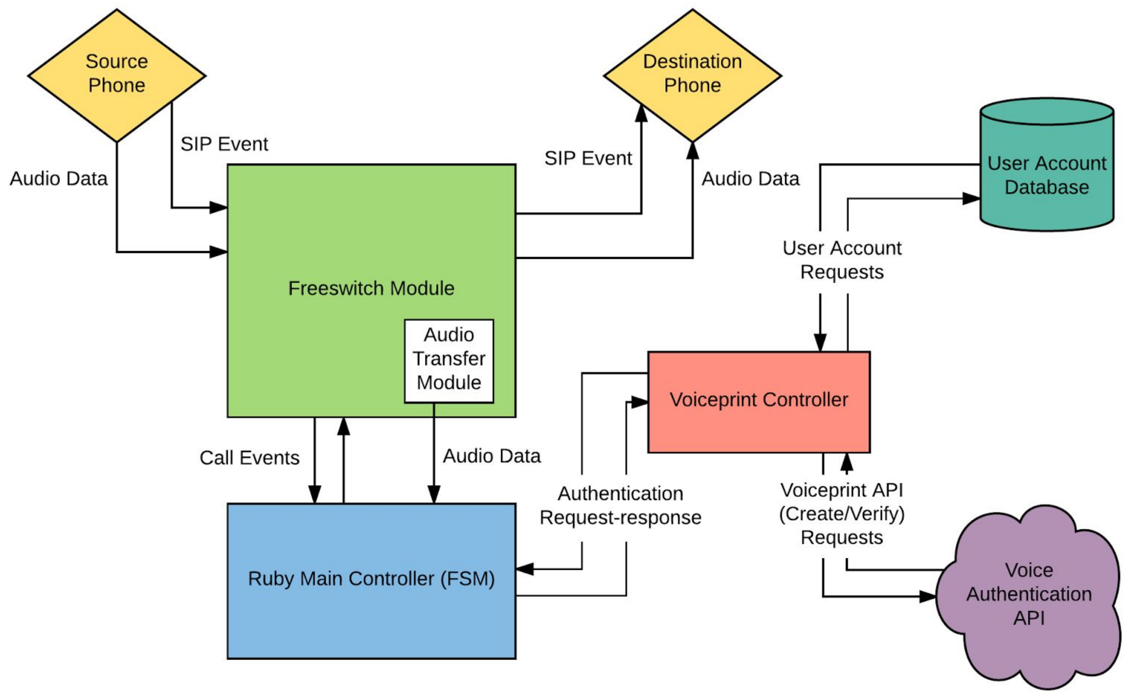
ASSUMPTIONS

- Users desire faster authentication when accessing their account over the phone
- Users desire to have more security when accessing their account (2FA)
- Account administrators desire to have users with more secure accounts, so that they do not have to deal with the issues of compromised accounts
- Account administrators desire to have call centers with improved efficiency as time is wasted authenticating users manually
- Users’ voices may fluctuate short term and change over time, so alternative methods of authentication should be in place to enable users to update their voice profiles
- Vulnerabilities in voice authentication technology exist, therefore there must be preventative measures in place such as randomization of voiceprint phrases

System Architecture Overview

HIGH LEVEL DIAGRAM

Illustrated below is our preliminary system architecture. Each piece handles specific responsibilities, which are outlined. The Freeswitch Module, Ruby Main Controller, Voiceprint Controller, and User Account Database are all within their own docker containers and communicate over sockets.



Freeswitch Module:

This module handles all of the protocols to initiate a phone call and conversate. The freeswitch module monitors call events such as calling, answering, hanging up, and entering digits. These events are transferred to the Ruby Main Controller in the form of Call Event objects.

Ruby Main Controller:

This module contains the main “brains” of the whole system. It facilitates the IVR (Interactive Voice Response) tree. In our system, the IVR tree maintains the current state of the caller and transfers them accordingly based on their user input (Ex: “Prompt: Press 1 for Directions, 2 for operator”). Based on the current state of the caller, the ruby controller interfaces with the voiceprint controller to retrieve user info and issue requests for authentication.

Voiceprint Controller:

The voiceprint controller handles the logistics behind creating and retrieving users and their voiceprints in the system. When the main controller issues a request to create a voiceprint, the controller interfaces with the API to get a unique identifier for the new user. The voiceprint controller then inserts the new user account into the database with attributes: First Name, Last Name, Phone Number, and Verification Profile UUID. When the controller receives a verify request, it will retrieve the voiceprint UUID from the database as a parameter for the API request to verify. The response is returned to the ruby controller and ultimately presented to the agent.

User Account Database:

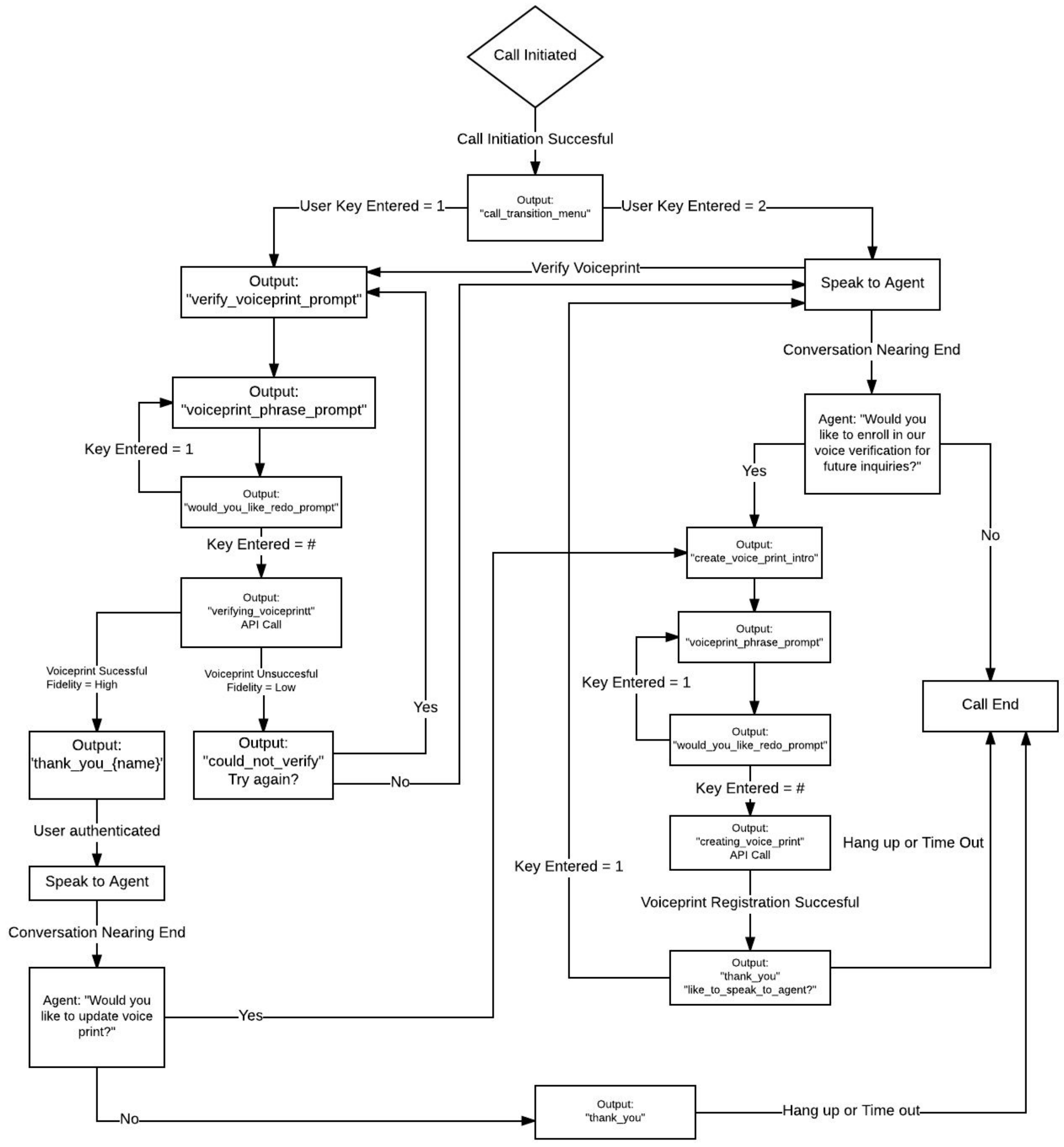
Due to privacy reasons, Microsoft Speaker Recognition API does not store personal info with a user’s voiceprint and simply creates a verification profile ID. Therefore, we must maintain our own database of that a user’s name, phone number, and verification profile ID for later retrieval.

Voice Authentication API:

After much analysis of the voice authentication APIs (Voice API Analysis) we decided on Microsoft Cognitive Services - Speaker Recognition API. All of the voice APIs shared similar limitations, only specific phrases provided by the API could be spoken by the user to be verified. These phrases bring out certain characteristic patterns that are used to create a unique user’s voiceprint. The API will be used to handle all speech processing.

USER EXPERIENCE DIAGRAM

In addition to the high level system diagram, we have outlined the general pipeline of events for the Ruby Main Controller. In the diagram below, outputs are indicated by quotes and often correspond with a pre-recorded sound file to be played. User inputs/Agent transfers are indicated by transitions between states. This diagram helps illustrate our user stories.



Call Initiated
Call Initiation Succesful
Output: "call_transition_menu"
User Key Entered = 1
User Key Entered = 2
Output: "verify_voiceprint_prompt"
Verify Voiceprint
Speak to Agent
Output: "voiceprint_phrase_prompt"
Key Entered = 1
Output: "would_you_like_redo_prompt"
Key Entered = #
Output: "verifying_voiceprintt" API Call
Voiceprint Sucessful Fidelity = High
Voiceprint Unsuccesful Fidelity = Low
Output: 'thank_you_{name}'
Output: "could_not_verify" Try again?
Yes
No
User authenticated
Speak to Agent
Conversation Nearing End
Agent: "Would you like to update voice print?"
Yes
No
Output: "thank_you"
Hang up or Time out
Conversation Nearing End
Agent: "Would you like to enroll in our voice verification for future inquiries?"
Yes
No
Output: "create_voice_print_intro"
Output: "voiceprint_phrase_prompt"
Key Entered = 1
Output: "would_you_like_redo_prompt"
Key Entered = #
Output: "creating_voice_print" API Call
Hang up or Time Out
Key Entered = 1
Voiceprint Registration Succesful
Output: "thank_you" "like_to_speak_to_agent?"
Call End

Source: Prototype of User Experience Diagram

Requirements

PRIORITIZED USER STORIES

1. As a **customer**, I can call the call center so that I can take care of business.

Deliverable	User calls a number FreeSwitch picks up call User is prompted with welcome message
Prototype	GitHub Ruby Controller Interface with Freeswitch
Tests	● Simulate a call with softphone ● Check that freeswitch answers and parks the call ● Hang up call using freeswitch

2. As a **customer**, I can call the call center and create a voice print so that I can use voice authentication in the future

Deliverable	Ruby code to call Microsoft Voice API
Prototype	Ruby code to handle Create Voiceprint API requests
Tests: (manual)	● POST request to API ● Ensure response code is valid ● GET request to API with response id ● Ensure profile exists

Deliverable	Code that creates audio recordings from call
Tests:	● Create/record audio file ● Delete audio file

3. As a **customer**, I can call the call center and be authenticated using my voice print before talking to an agent so that I can:

- Authenticate faster and painlessly
- Add another level of security
- Keep my details private

Nonfunctional Requirement	Authentication response in under a second
Tests:	• Time GET request to Microsoft API • Ensure response comes back < 1 second

Nonfunctional Requirement	90% of the time, it works 100% of the time
Tests:	• Make 10 verification requests • Ensure 9 responses come back with 'High' certainty response

Deliverable	User is prompted to say voiceprint passphrase. User says passphrase. User is told asked whether they would like to submit or re-record passphrase. User submits recording. System tells user whether they were authenticated User gets transferred to agent
Prototype:	FSM diagram describing call transitions and pipeline. Prototype of User Experience Diagram
Tests:	• Verify that audio file was created • Verify API request was sent out • Verify API authentication response • User/call has state 'authenticated' change from 'true' to 'false'

4. As a **customer support agent**, I can know whether a customer was properly authenticated when their call is transferred to me.

Deliverable	Ruby code that plays a recording through FreeSWITCH
Tests:	● Verify the audio file was sent to the Voice API. ● Confirm there is feedback from voice API
Prototype	Ruby code to handle Verify Voiceprint API requests

5. As a **call center admin**, I can spin up the docker environment in order to set up the product.

Deliverable:	● Docker-Compose file ● Docker files ● Docker images hosted on Docker Hub
Prototype	GitHub Docker Files and Freeswitch
Tests (manual)	● Try setting up the system on a fresh machine

6. As a **customer**, I can bypass the authentication system so that I can authenticate with a customer support agent.

Deliverable	Ruby code that redirects the user from the IVR to the Agent through freeswitch.
Tests:	● Verify the call remains connected during transition ● Verify that the agent is notified that the user will be connecting to them shortly

7. As a **customer**, I can update my voiceprint so that I can use a more recent voiceprint

Deliverable:	IVR system that prompts the user with a new passphrase, records the new audio file, plays it back, and asks if they would like to: save, redo or choose another passphrase.
Tests (manual)	● Verify that call is sent to voiceprint state of IVR. ● Verify the new audio file has been saved. ● Verify user's input is stored and transferred to correct stage.

8. As a **customer**, I can delete my voiceprint so that I can remove my voice data from the system.

Deliverable:	● IVR states for this? ● Ruby code
Tests (manual)	● Verify API request was sent out ● Verify API authentication response ● Verify voiceprint is gone

9. As a **customer support agent**, I can delete a user's voiceprint so that they remove their data from the system.

Deliverable:	● IVR states for this? ● Ruby code
Tests (manual)	● Verify API request was sent out ● Verify API authentification response ● Verify voiceprint is gone

10. As a **customer support agent**, I transfer the customer to a specific stage of authentication (such as create fingerprint, update fingerprint) so that I can help the customer with their voice authentication

Deliverable:	● Ruby code
Tests (manual)	● Verify that call is sent to specific voiceprint state of IVR.

Appendix

TECHNOLOGIES EMPLOYED

Product

Docker	Development platform for building, shipping and running applications
Freeswitch	Open source telephony server
Ruby Sinatra	Lightweight web server used offer interface for call center agent
Librevox	Ruby wrapper for mod_event_socket Freeswitch interface
Microsoft Speaker Recognition API	Voice Authentication
SQLite3	User account database

Internal

Slack	Team Communication
Github	Source Version Control and code repository management.
Waffle.io	Agile/Scrum Task Board
TravisCI	Continuous Integration code testing

WHAT WE DON'T DO:

1. Conversational automated voice authentication

We will not authenticate users in the background based on a natural conversation. Users will have to speak predetermined phrases in order for our solution to work. This is a result of the limitations of the available voice authentication APIs.

2. Conversational manual voice authentication

We will not specify how a call center agent should authenticate a person in the case where automated authentication fails. This is up to the particular business to determine if they would like to use security questions, pins, password, etc.