Life after Speech Recognition: Fuzzing Semantic Misinterpretation for Voice Assistant Applications

Yangyong Zhang, Lei Xu, Abner Mendoza, Guangliang Yang, Phakpoom Chinprutthiwong, Guofei Gu
SUCCESS Lab, Dept. of Computer Science & Engineering
Texas A&M University
{yangyong, xray2012, abmendoza, ygl, cpx0rpc}@tamu.edu, guofei@cse.tamu.edu

***Abstract*—Popular Voice Assistant (VA) services such as Amazon Alexa and Google Assistant are now rapidly appifying their platforms to allow more flexible and diverse voice-controlled service experience. However, the ubiquitous deployment of VA devices and the increasing number of third-party applications have raised security and privacy concerns. While previous works such as hidden voice attacks mostly examine the problems of VA services' default Automatic Speech Recognition (ASR) component, our work analyzes and evaluates the security of the succeeding component after ASR, i.e., Natural Language Understanding (NLU), which performs semantic interpretation (i.e., text-to-intent) after ASR's acoustic-to-text processing. In particular, we focus on NLU's Intent Classifier which is used in customizing machine understanding for third-party VA Applications (or vApps). We find that the semantic inconsistency caused by the improper semantic interpretation of an Intent Classifier can create the opportunity of breaching the integrity of vApp processing when attackers delicately leverage some common spoken errors.**

In this paper, we design the first linguistic-model-guided fuzzing tool, named LipFuzzer, to assess the security of Intent Classifier and systematically discover potential misinterpretation-prone spoken errors based on vApps' voice command templates. To guide the fuzzing, we construct adversarial linguistic models with the help of Statistical Relational Learning (SRL) and emerging Natural Language Processing (NLP) techniques. In evaluation, we have successfully verified the effectiveness and accuracy of LipFuzzer. We also use LipFuzzer to evaluate both Amazon Alexa and Google Assistant vApp platforms. We have identified that a large portion of real-world vApps are vulnerable based on our fuzzing result.

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-891562-55-X
https://dx.doi.org/10.14722/ndss.2019.23525
www.ndss-symposium.org

I. Introduction

The Voice User Interface (VUI) is becoming a ubiquitous human-computer interaction mechanism to enhance services that have limited or undesired physical interaction capabilities. VUI-based systems, such as Voice Assistants (VA), allow users to directly use voice inputs to control computational devices (e.g., tablets, smart phones, or IoT devices) in different situations. With the fast growth of VUI-based technologies, a large number of applications designed for Voice Assistant (VA) services (e.g., Amazon Alexa Skills and Google Assistant Actions) are now available. Amazon Alexa currently has more than 30,000 applications, or vApps[1].

Several attacks have been reported to affect the integrity of existing Automatic Speech Recognition (ASR) component in vApp processing. For example, acoustic-based attacks [19], [36], [33], [35], [23] leverage sounds that are unrecognizable or inaudible by the human. More recently, Kumar et al. [28] presented an empirical study of vApp squatting attacks based on speech misinterpretation (for example, an Alexa Skill named "Test Your Luck" could be routed to a maliciously uploaded skill with a confusing name of "Test Your Lock"). This attack works, with proof-of-concept, in a remote manner that could potentially be more powerful than acoustic-based attacks.

Despite recent evidence [28] of launching potential vApp squatting attacks, little effort has been made to uncover the root cause of speech misinterpretation in vApp processing. In this paper, we devise a first representative Voice Assistant (VA) Architecture, as shown in Figure 1, from which we study the core components involved in proper speech interpretation. After closely scrutinizing the VA architecture, we found that both the Automatic Speech Recognition (ASR) and the Natural Language Understanding (NLU) components play a central role in

[1] vApp is the generalized name used in this paper for Amazon Alexa Skills, Google Assistant Actions.

proper speech recognition and interpretation. Previous works have only studied the ASR text interpretation component. In the NLU, an Intent Classifier uses voice command templates (or templates) to match intents (similar to Intent Message in Android) with textual data obtained after ASR's text interpretation. Then, intents are used to reach vApps with specific functionality. From the vApp security perspective, the Intent Classifier plays a more important role when interpreting users' voice commands. The reason is that the Intent Classifier is the last step of the interpretation process. It has the capability of not only determining users' semantic intents, but also fixing the ASR's potential transcription errors. Second, while ASR is a default built-in service component, the construction of the Intent Classifier's semantic classification is contributed by both vApp developers and service providers. Particularly, third-party developers can upload voice command templates to modify the unified intent classification tree used by all users (more details are illustrated in Section II). As a result, it creates the opportunity for misbehaving developers to maliciously modify the intent matching process of the NLU.

However, it is challenging to systematically study how NLU's Intent Classifier is penetrated to incur semantic inconsistency between users' intents and VA's machine understanding. First, mainstream VA platforms, such as Amazon Alexa, Google Assistant, and almost all vApps are in closed development. Thus, it is difficult to conduct a white box analysis. Also, due to the strong privacy enforcement, it is impossible to get real users' speech input and the corresponding vApp response output. Thus, conducting large-scale, real-world data-driven analysis is also very challenging.

Our Approach. In this work, we assess speech misinterpretation through the black-box mutation fuzzing of voice command templates, which are used as inputs of Intent Classifier for matching intents. Our goal is to systematically evaluate how Intent Classifier behaves when inputting different forms of voice commands. However, it is not straight-forward to design such a fuzzing scheme. First, the computability of vApp I/O is very limited as both input and output of vApps are in the speech form, i.e., you can only speak with a VA device or listen to the audio response[2]. Thus, we have to determine mutation fields that we can work on in the context of vApp processing. Moreover, it is important to eliminate the effect of ASR's text interpretation so that error propagation is minimized. Second, in [28], the authors suggest that vApp squatting attacks are caused by pronunciation-based interpretation errors of ASR. However, in reality, ambiguous natural languages incur many more different forms of confusing voice commands. For example, a user could be simply using regional vocabulary (i.e. words or expressions that are used in a dialect area but not commonly recognizable in other areas) rather than pronunciation issue alone. Moreover, it is unpredictable when and how a user would speak differently. All of these factors make the voice command fuzzing difficult because of the large searching space.

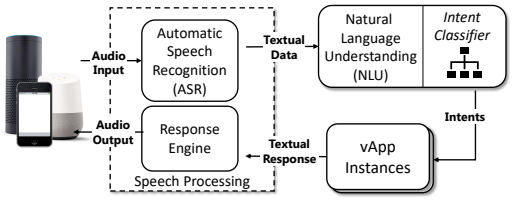


Fig. 1: VUI-based VA Architecture.

To overcome the aforementioned design challenges, we propose a novel linguistic-model-guided fuzzing tool, called LipFuzzer. Our tool generates potential voice commands that are likely to incur a semantic inconsistency such that a user reaches an unintended vApp/functionality (i.e. users think they use voice commands correctly but yield unwanted results). For convenience, we name any voice commands that can lead to such inconsistency as LAPSUS.[3] LipFuzzer addresses the two aforementioned challenges in two components. First, to decide mutation fields, we realize that the state-of-the-art Natural Language Processing (NLP) techniques can be used to extract computational linguistic data. Thus, we design LipFuzzer's basic template fuzzing by first mutating NLP pre-processed voice command templates. Then, we input mutated voice commands to VA devices by using machine speech synthesis to eliminate the human factor of producing ASR-related misinterpretation. Second, to reduce the search space of the basic fuzzing, in the linguistic modeling component, we train adversarial linguistic models named LAPSUS Models by adopting Bayesian Networks (BNs). BNs are constructed statistically with linguistic knowledge related to LAPSUS. The template fuzzing is then guided with linguistic model query results. As a result, LipFuzzer is able to perform effective mutation-based fuzzing on seed template inputs (gathered from existing vApp user guidance available online).

In our evaluation, we first showcase that Intent Classifier is indeed the root cause of the misinterpretation. Then, we show that LipFuzzer can systematically pinpoint semantic inconsistencies with the help of linguistic models generated from existing linguistic knowledge.

Furthermore, we scan Alexa Skill Store and Google Assistant Store to assess the security problem we found.

[2]With a VUI setting, no other user interfaces such graphic user interface (or GUI) are being used.

[3]We use LAPSUS for its Latin meaning "slip".

Our result shows that 26,790 (out of 32,892) Amazon Alexa Skills and 2,295 (out of 2,328) Google Assistant Actions are potentially vulnerable. A further sampling-based verification shows that around 71% of these Amazon Alexa vApps and 29% of these Google Assistant vApps are actually vulnerable. We provide detailed results and analysis in Section VI-C.

Contributions. The main contributions of this paper are highlighted as follows:

(i) We analyze the problem of the semantic inconsistency of speech interpretation in vApp, and uncover that this problem is deeply-rooted in NLU's Intent classifier.

(ii) We model existing voice command errors to enable systematic analysis. More specifically, with SRL and NLP techniques, our modeling process is able to convert linguistic knowledge into computational statistical relational models.

(iii) We design an automated linguistic-model-guided mutation fuzzing scheme, named LipFuzzer, to assess Intent Classifier at a large scale. We will publish the source code and associated linguistic models to help fortify the security of vApp processing. We are also communicating with those affected vApp developers and platforms to assist in understanding and fixing the security vulnerabilities/problems.

II. Background

In this section, we first introduce Intent Classifier in NLU. Next, we briefly discuss linguistic-related LAPSUS. Then, we show a motivating example of how a voice squatting attack affects the Intent Classifier. Lastly, a threat model is presented.

A. Natural Language Understating (NLU) of vApp

As shown in Figure 1, after acoustic inputs (i.e., voice commands received by VA devices) are transcribed into textual data, a typical VUI-based VA platform processes the textual data to understand user's intents with NLU [7] [11]. As a result, intents are generated for later vApp processing.

Intent Classifier. To produce accurate intents, NLU performs a two-step procedure: NLP transformation and Intent Classification. The NLP part follows standard procedures such as Tokenization (word segmentation), Coreference Resolution (COREF), and Named Entity Recognition (NER) [14]. This gathers the syntax information of given textual data. Next, to understand the semantic meaning, NLU further matches the syntax data with a pre-built intent classification tree. In the tree, the branch with the highest confidence will be selected to produce intents. Note that there may exist vApp-irrelevant built-in branches for matching VA's default services, such as for alarm and default music.

A classification tree is the result of aggregating a VA system's built-in services' voice commands (e.g. voice search and time query) and developer-defined voice command templates created through Interaction Model [2]. In this paper, our focus is to analyze how malicious third-party developers can affect an Intent Classifier. Thus, using high-level examples[4], we introduce how developers contribute to building the intent classification tree in two parts: developer-defined built-in intent generation, and custom intent definition. First, to generate built-in intents for a vApp, a developer needs to define the installation and invocation names.

```
1  #1.Developer-defined
2  "vApp Installation Name":
3  "The True Bank Skill",
4  "vApp Invocation Name":
5  "True Bank",
6  #2.Auto Generated Intents
7  "SYSTEM.InstallIntent":
8  {"Alexa, enable The True Bank Skill.",
9  "Alexa, install The True Bank Skill."},
10 "SYSTEM.LaunchIntent":
11 {"Alexa, open True Bank.",
12 "Alexa, ask True Bank to ... ."}
```

Listing 1: Example Template with Built-in Intent.

As shown in Listing 1, an example vApp template (defined through programming an Interaction Model) is defined with an installation name at Line 2-3, and the invocation name is defined at Line 4-5. The VA system will then automatically generate the voice command templates (from Line 6 to 12) accordingly for updating the classification tree. For instance, Amazon Alexa typically follows a format like "Alexa, install `Installation Name`" for installing a vApp. Other built-in intents, such as `SYSTEM.CancelIntent`, `SYSTEM.HelpIntent`, are tied to default words (e.g. "stop", "cancel") with no requirement for developer involvement.

```
1  #3 Custom Intents
2  "CUSTOM.BalanceQueryIntent":
3  {"Tell me my balance",
4  "Alexa, ask True Bank about my account balance.",
5  "What is my balance?"},
6  "CUSTOM.TransferIntent":
7  {"Alexa, ask True Bank to transfer money to Alice.",
8  "I want to send money to someone."}
```

Listing 2: Example Template with Custom Intent.

Then, as shown in Listing 2, a developer can also define customized intents with associated voice command templates. For example, a developer can define a custom intent `CUSTOM.BalanceQueryIntent` (at Line 2) with corresponding voice command templates (at Line 3-5).

[4]Although low-level implementations of the classification process can be different, based on our study, both Amazon Alexa and Google Assistant's high-level architectures are almost the same.

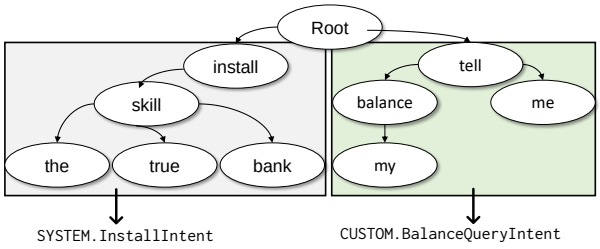


Fig. 2: Intent Classification Tree Example

In Figure 2, we show a simplified classification tree based on the dependency relationship of NLP-processed voice commands. The left nodes are generated from a built-in installation voice command, i.e., "install the True Bank Skill", defined at Line 7-9 in Listing 1. The right nodes are generated from a custom voice command listed in Listing 2. With more vApp Interaction Models aggregated into the tree, additional nodes will be added. Any voice command input in a VA platform will be processed with this tree.

Slot Values. A slot is the value associated with an intent. Slots can be associated with intents by using tags. For example, with the "transfer money" voice command mentioned above, a developer can define a voice command with a slot type. For example, "Alexa, ask True Bank to transfer Money to *Alice*." (at Line 6-8 in Listing 2). "Alice" can be defined as `Contacts` slot value that is associated with `CUSTOM.TransferIntent`. This `Contacts` uses default slot type `SYSTEM.US_FIRST_NAME` which will force the NLU to automatically match common people names used in the US, and hook the name value with intents. Slot values are usually important in processing voice commands as they contain the key data for deciding the next step. For example, in `SYSTEM.InstallationIntent`, a unique ID related to the installation name is associated with the slot value to decide which vApp to install.

Fuzzy Matching. Another important feature enabled in NLU is the fuzzy matching of users' intent. It can tolerate minor LAPSUS. For example, a voice intent for launching "True Bank" can be interpreted correctly with the voice command "open True Banks" or "opening True Bank". We note that Google Assistant always applies fuzzy matching, but Amazon Alexa only enables this feature after a vApp is installed. Hence, in Amazon Alexa, a vApp Installation Name has to be spoken exactly with minimum tolerance for any LAPSUS.

B. *Speech Errors and* LAPSUS

People misspeak words, and this has long been studied as speech errors in the psycholinguistic field [30] [22]. One study shows that most people make somewhere between 7 and 22 verbal slips everyday [13]. Usually, misspeaking is considered to occur more often for children, non-native speakers, and people who are nervous, tired, anxious, or intoxicated. Speech errors are classified into three categories: mispronunciation, vocabulary errors, and grammatical [31] errors. For example, in English, one could misuse the "-ed" form, and this falls into grammatical errors.

In this paper, we motivate LAPSUS from daily speech errors that may cause machines' misunderstanding problems. However, we consider that LAPSUS is not limited to speech errors. Other types of linguistic issues can also cause LAPSUS, such as combined words, regional vocabulary, and other malformed speech.

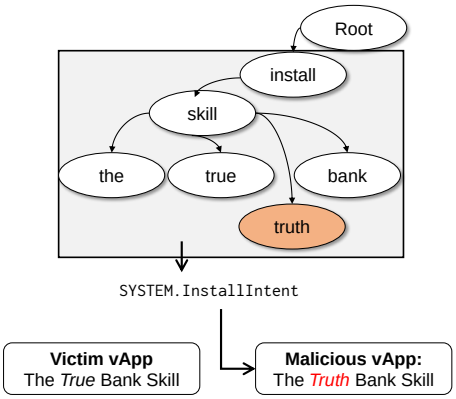


Fig. 3: Example vApp Squatting

C. *Motivating Example*

To better understand how Intent Classifier relates to the speech misinterpretation, we show an example of vApp squatting attacks using the vApp illustrated in Listing 1 and Listing 2. In this example, a maliciously uploaded vApp aims to hijack the victim vApp when users try to install the victim vApp. The correct form of installation voice command should be "Alexa, install The True Bank Skill". An adversary intentionally crafted his/her vApp with a confusing name of "The Truth Bank Skill". As a result, as shown in Figure 3, a new leaf node of "truth" is injected into the intent classification tree. Thus, if a user accidentally speaks "Alexa, install The Truth Bank Skill", the malicious vApp will be installed and executed (with few or no indication of which vApp is running). Similarly, as mentioned above, any LAPSUS can take place in any words/expressions when users are speaking to a VA.

Thus, it is crucial to evaluate the current Intent Classifier design, which is closely related to the recently proposed vApp squatting attack [28].

D. Threat Model

We assume that an adversary does not need to access any internal implementations/configurations of vApp including ASR, NLU components. We consider the vApp-enabled devices are safe. Moreover, we argue that a third-party vApp can be malicious in the vApp store.

Attack consequences. We summarize a list of example consequences of interacting with malicous vApps as follows:

Denial of Service: This would occur if a device-controlling vApp is hijacked. For example, when a user says "Alexa, close the garage door" to a mismatched malicious vApp instead of a legitimate one, the garage door may be left open.

Privacy Leakage: vApps intimately connect users' daily life to a more exposed digital world. By interacting with improper vApps, private data could be handled unexpectedly. For example, as reported in 2018, a Portland family's Alexa device accidentally captured a private conversation, and then sent the audio recording to someone whose number was stored in the family's contact list [4].

Phishing: Third-party vApps' back-end processing is fully controlled by the developers. To the best of our knowledge, no proper monitoring (e.g. runtime verification of third-party back-end processing) is deployed. Audio can thus be manipulated by an attacker at runtime, substituting a benign audio file with a "malicious" audio file. For example, if an audio file, "*Hello, Welcome*", is played when a vApp is opened, it can be substituted, requiring no permission or notification, with "*Sorry, a critical problem occurred in your Amazon Pay, please answer with your account username and password to know more.*"

Other Consequences: It is clear that current template-based intent classification is not able to enforce accurate control of intended functionalities. As a result, voice-controllable services are currently limited to mostly non-critical applications. However, as VUI-based systems are quickly evolving to enable richer functionalities, e.g., Amazon Alexa's recently announced third-party in-vApp purchasing [3], more destructive and never-before-seen consequences could be expected in the future.

III. Fuzzing Speech Misinterpretation

In this section, we first present detailed challenges of fuzzing speech misinterpretation. Then, we briefly illustrate our LipFuzzer design which addresses the challenges accordingly.

A. Fuzzing Challenges

As mentioned in Section I, we aim to design a mutation-based scheme to fuzz the problematic Intent Classifier. However, it is challenging because of VA's unique features:

(i) Mutation Field: Deciding the mutation field is not straight-forward since natural languages (e.g. voice commands) are complicated in terms of the linguistic structure. A voice command can be segmented into different levels of linguistic units: from low-level phonetic features to high-level phrases consisting of a few words. Therefore, we need to first decide what are the most LAPSUS-relevant linguistic units. Additionally, we need to select the proper tool to extract computational linguistic data for mutation.

(ii) Space Explosion: A naive fuzzing solution is to use random fuzzing to generate LAPSUS, e.g., at the character level, we may mutate each character of a voice command. However, such random fuzzing is impractical as it would trigger a large number of resulting voice command variations. For example, for character-based mutation, a simple voice "open the truck" has 12 characters and each character of a word can have at least 25 possible ways of mutation (e.g. "a" to "b"). Moreover, a voice command can also be mutated with phonemes, words, etc., which makes the potential fuzzing space infinitely large. As a result, we need to find a strategy to reduce the search space of finding effective LAPSUS.

B. Our solution: LipFuzzer

To solve the fuzzing challenges, we design LipFuzzer shown in Figure 4.

Linguistic Modeling. To overcome the difficulty of locating mutation fields, we convert vague voice commands (textual data) into computational fine-grained linguistic data using NLP. Specifically, the textual data are processed into a three-level linguistic structure that is commonly used in studying linguistic speech errors [21]: pronunciation, vocabulary, and grammar. Thus, we design the Linguistic Modeling component of LipFuzzer. As the input of linguistic modeling, we collect relevant linguistic knowledge from both academic materials and English teaching websites. Second, our tool abstracts logic representation (i.e., predicate logic) of these pieces of knowledge by manual parsing, or automatic logic generation (if examples are provided). Third, we construct linguistic models, namely LAPSUS Models, by formulating Bayesian Networks (BNs) from predicate logic entities we collected. Lastly, we train BNs with statistical weights which measure how likely

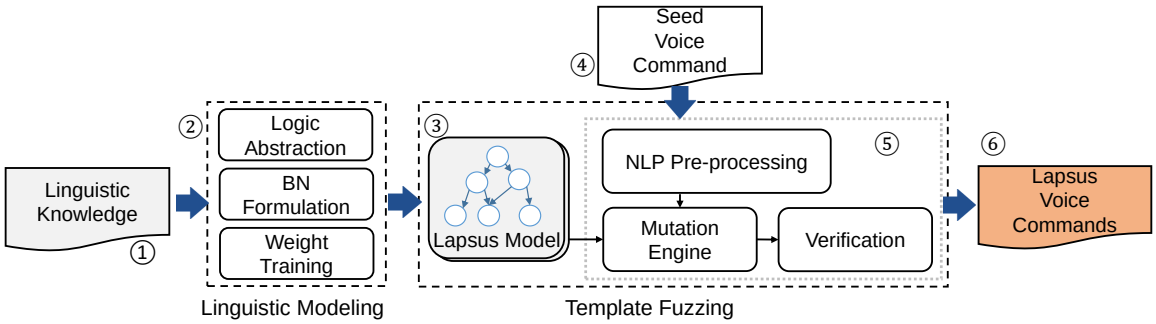


Fig. 4: LipFuzzer Architecture

it is that a state transition (i.e., mutating a field to result in another state) would take place. Our LAPSUS Models can be used to answer both how a LAPSUS will be generated, based on seed voice commands (i.e., collected templates used for matching vApp intents), as well as the weights of these mutations. Lastly, The LAPSUS Models are loaded with Template Fuzzing to guide the Mutation Engine for generating LAPSUS voice commands.

Template Fuzzing. To address the space explosion problem when fuzzing voice commands, we leverage LAPSUS-related linguistic knowledge to reduce the search space to find effective LAPSUS. Thus, we design our template fuzzing component which takes the input of seed voice commands and generates LAPSUS voice commands. Three basic modules are shown in Figure 4: NLP Pre-processing, Mutation Engine, and Verification. First, the NLP Pre-processing module extracts linguistic information for mutating. Second, the Mutation Engine mutates the fields such as phonemes, words, or grammar entities based on LAPSUS Models. Third, the Verification module synthesizes speech voice command to verify whether a LAPSUS is effective.

IV. LINGUISTIC-MODEL-GUIDED FUZZING

In this section, we detail LipFuzzer's design. First, we define the input and output of LipFuzzer. Second, we present the linguistic modeling component for producing LAPSUS Models. Lastly, we illustrate Template Fuzzing which is guided with LAPSUS Models.

A. Fuzzing Input & Output

Linguistic Knowledge. The input of linguistic modeling is linguistic knowledge data shown in ① of Figure 4. We choose LAPSUS-related linguistic knowledge from multiple sources [12] [26] [1] [20] [24]. They are part of many linguistic concepts (examples shown in Table I) related to LAPSUS. With various descriptions and discrete features, we have to manually select them. As a result, we have a total of 498 pieces of knowledge collected (which are expected to expand over time), including 53, 264, 181 for sound-level, word-level, and grammar-level, respectively. In addition, 223 of them come with examples (e.g., target and LAPSUS in Table I).

LAPSUS Models. The output of the linguistic modeling component is shown in ③. Multiple linguistic models are generated after the mathematical modeling. In this work, we use graph-based data structures to represent linguistic knowledge statistically. Moreover, a LAPSUS can be queried with proper query inputs (we show details in Section IV-C).

Seed Input. The input of Template Fuzzing is seed voice command templates (or seed templates) shown in ④. We collect seed templates from Alexa Skill Store through web crawling. These seed templates are example voice commands posted by vApp developers. Although only a few developers would display all voice commands in the store, most of these example voice commands are essential ones. Thus, we believe these voice commands are enough to demonstrate the design flaw in Intent Classifier.

Fuzzing Output. In ⑥, LipFuzzer generates modified voice commands based on seed inputs. A modification is done through a linguistic knowledge guided mutation.

B. Linguistic Modeling

We build LAPSUS Models based on existing LAPSUS knowledge. As shown in ② of Figure 4, LAPSUS Models are generated with three modules: Logic Abstraction, BN Formulation, and Weight Training.

Logic Abstraction. As the first step of linguistic modeling, we use predicate logic to represent collected LAPSUS knowledge. The reason is that predicate logic is widely used in traditional linguistic studies with extensible representation. It is capable of using quantified variables over non-logic LAPSUS, which then

TABLE I: Example LAPSUS with Logic Abstraction

Lapsus	Description	Examples	Example Logic Abstraction
Blends†	Two intended items fuse together when being considered.	Target: person/people LAPSUS: perple	∀ x,y,phoneme(END,"S-N",x), phoneme(END,"P-L",y) → phoneme_exch("S-N","P-L",-)
Morpheme∗ -Exchange	Morphemes changes places.	Target: He packed two trunks. LAPSUS: He packs two trunked.	∀ x,y,suffix("ed",x), suffix("s",y) → suffix_exch("ed","s",-)
Regional Vocabulary ‡	Everyday words and expressions used in different dialect areas	Target: Entree Lapsus: Hotdish (esp. Minnesota)	∀ x,word("entree",x),→ word_exch("entree","hotdish",-)
Category Approximation ‡	Word substitution due to the lack of vocabulary knowledge.	Target: Show my yard camera. Lapsus: Turn on my yard camera.	∀ x,word("show",x), → word_exch("show","turn on",-)
Portmanteaux‡	Combined words that are used.	Target: Eat the (late) brekfast Lapsus: Eat the brunch	∀ x,word("late breakfast",x), → word_exch("late breakfast", "brunch",-)

†: Pronunciation, ‡: Vocabulary, ∗: Grammar.

allows succeeding modeling and computation of LAPSUS knowledge. As our fuzzing scheme works with pronunciation, vocabulary, and grammar level of linguistic units, our predicate logic representations are also defined with these three types.

However, as linguistic knowledge is typically defined in a discrete manner, many are vague and difficult to translate. Thus, we transform collected linguistic knowledge into predicate logic representation in two ways: manual abstraction and automated example-based abstraction. A manual abstraction process works for linguistic concepts that lack proper examples. We used a structured approach for manual abstraction that aligns well with our automated abstraction. For example, in Table I, an example logic abstraction of blends indicates that: for any two words, x and y, if x ends with phoneme combination "S-N" (e.g., "son") and y ends with "P-L" (e.g., "ple"), then these two phoneme combinations may be fused with each other.

For an automated logic abstraction, a differential analysis is performed to extract the difference of a pair of textual data to generate logic function used to describe the difference. For example, we show the target and LAPSUS form of linguistic knowledge in Table I which can be used to perform a differential analysis. In Morpheme Exchange [16], the correlation results of examples will be the suffixes of "pack" and "trunk" which are at the grammar level.

Bayesian Network (BN) Formulation. Bhme et al. [17] showed that state transition graphs such as Markov Chain can be used in modeling fuzzing processes. We use BNs [27] to model LAPSUS statistically because BNs are widely utilized in Statistical Relational Learning (SRL), which solves many AI problems including Natural Language Understanding [25] [32].

In our model, we define BN as a joint probability

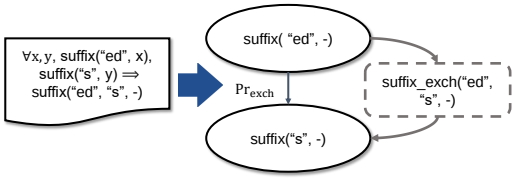


Fig. 5: BN Formulation Example

distribution over a finite set of states

$$BN : G = (E, V), P \tag{1}$$

Our proposed BN data structure has two components: a directed acyclic graph $G = (E, V)$ with each vertex representing a state of a possible form of linguistic expressions; P, a set of probability variable P_e which is indexed by E . Each state is defined as a logic clause such as functions or variables (example functions shown in Table II). The transition probabilities $p_{i,j}$ define the processing from state v_i to state v_j. The density of the stationary distribution formally describes the likelihood that a certain LAPSUS event is exercised by the fuzzer (for example, a state with no prefix to the state with the prefix "mal-"). In fact, any $p_{i,j}$ in our model is the quantified result of a transition state (i.e., by counting how many times the specific LAPSUS is observed). For example, we count how many times a `suffix_exch("ed","s",-)` occurs and calculate a corresponding probability.

We show how to build BNs with the example in Figure 5. Initially, three nodes are created: the starting state of a word with a suffix "ed", the transition state showing a suffix exchange, and an ending state indicating the word with suffix "s" as the result of the exchange. The transition state, however, will not be directly aggregated into a BN. Instead, it will be calculated and shown as the

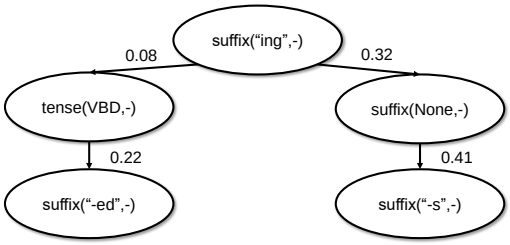


Fig. 6: BN Example with Weight Trained

probability weight from the starting node to the ending node. With multiple paired nodes input to formulate a BN, a multi-hop BN graph will be constructed. We show an example grammar-level BN in Figure 6. (Note that transition states are not shown.) In this study, we have three different levels of LAPSUS Models based on the above formulation process.

TABLE II: Logic Functions in BN Modeling

Function	Examples
Pronunciation	
`phoneme(Op,Var ,Cons)`	`phoneme(END,"S" "time"),` e.g.'T-AY-M-S' (”times”)
Vocabulary	
`word(Op,Pos,Var Cons)`	`word(AFTER,- ,"please","enable"),` e.g.”enable please”
Grammar	
`suffix(Var, Cons)`	`suffix("-s","wait"),` e.g. ”waits”
`prefix(Var, Cons)`	`prefix("mal-", "function"),` e.g. ”malfunction”
`tense(Var, Cons)`	`tense(VBD, eat),` e.g.”ate”

Weight Training. After we have the initial BN (i.e., a graph with all edge weights set as 1) ready, we further train the weight (i.e., the probabilities of successful transition states) through a user study with audio recording. In this user study, we find sentences or short expressions which contains the states in the models. Then, we ask users in the study to repeat these sentences or expressions (detailed setting is shown in Section V-B). Next, we calculate how many times these transitions are observed. Then the probabilities of the transitions are calculated accordingly.

Further Refinement. Using only initially trained models is not scalable and accurate. Thus, we refine the model based on two rules. First, while fuzzing is in process, when there are any unformulated states observed, we add these states to the BN. Second, when a state transition is observed, the observed edge's weight will have a hit added. Meanwhile, any other egress edges from the starting state will be counted with a miss.

C. Template Fuzzing

Once LAPSUS Models are prepared, as shown in ⑤ of Figure 4, we then conduct automated template fuzzing based on seed templates. The Template Fuzzing component takes inputs of seed templates in the form of natural language. Then, seed templates are pre-processed using NLP tools. Next, the mutation engine performs guided fuzzing by querying LAPSUS Models. Thereafter, our tool verifies if the derived LAPSUS is effective by testing them with VA platforms.

NLP Pre-processing. Natural language such as voice commands do not have enough information for fuzzing tasks mentioned earlier. We leverage NLP techniques to retrieve computational linguistic information to build LAPSUS Models.

Pronunciation-level Information. We choose phonemes [18] as sound-level linguistic information since it is the basic sound unit. We extract phonemes from each word by leveraging CMU Pronouncing Dictionary in the NLTK package [29].

Vocabulary-level and Grammar-level Information. For vocabulary linguistic information, we leverage basic string metric (e.g., *Word_Match*, *Word_Count*). In order to tackle the ambiguity of the natural language, we also use grammar-level linguistic information, i.e., PoS Tagging, Stemming and Lemmatization [10]. In particular, PoS Tagging processes grammar information by tagging tenses and other word contexts. The stemming and the lemmatization are similar regarding functionality. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form [10].

Mutation Engine. After NLP pre-processing, seed templates are now in the form of three-level linguistic structure. We query each LAPSUS Model accordingly using these data. Each query is a path exploration process which returns all reachable states after a starting state. However, there are still problems that need to be solved for feasible fuzzing without path explosion.

How to guide the fuzzing? In our design, two strategies can be used in guiding template fuzzing so that we can avoid the large search space. First, we can guide the fuzzing process based on the weight of each query result. Another method is to guide the fuzzing based on the distance, which refers to the number of hops a path contains. We test both strategies as well as a randomized approach in our evaluation.

Algorithm 1: LAPSUS Model Query Algorithm

output: V_{result}
input : BN: $G = (E, V), P$
query: starting state S, cutoff C;
initialization:
$pr_v \leftarrow 0$;
$Visited \leftarrow \{\}$;
$v_{current} \leftarrow S$;
if S *not in* V **then**
  $Output$: None;
end
while $Size(Visited) < Size(V)$ **do**
  $v_{current} \leftarrow$ next state $v \in V$ & $v \notin Vistied$;
  calculate $pr_{current}$ based on P indexed by E;
  if $pr_i > C$ **then**
    $v_{current} \rightarrow Visited$;
  else
    truncate succeeding states of $v_{current}$ from V;
  end
end
$Output : V_{result} \leftarrow V$;

Here we show a weight-based model query algorithm in Algorithm 1. First, as defined by BN, a query input consists of both a starting state and a cutoff value. Second, we check if the linguistic data exists in the model. If there is no corresponding state, we return a $None$ message to indicate that no LAPSUS will be be generated. Third, the algorithm traverses the graph, vertex by vertex (both depth-first or breadth-first works). If a visited vertex has a resulting weight (e.g., a probability value pr) that does not satisfy the cutoff value, we truncate all the succeeding states as they will never be reached. Fourth, we remember all the states we visited until there are no more state left. Lastly, we return the remaining V as a set of all possible states that have a probability weight within the cutoff value. For a hop-based query, we simply substitute pr to the hop values.

How to decide a model query input? We decide query input based on BN state definition. Both pronunciation and grammar levels are queried for each word. For the vocabulary level, we process different combination of adjacent words based on the word count. In other words, we only process word combinations that exist in the models. As shown in Algorithm 1, a query consists of three parts: the BNs (i.e., the LAPSUS Models), the starting state for the input words, and the cutoff value.

Verification. We verify if a LAPSUS is effective by testing the synthesis audio with Alexa. To do that, we first synthesize speech from generated LAPSUS voice commands. Then we monitor the response of the VA. For installation-related voice commands, we check if the correct vApp is installed. For invocation voice commands, we first install the testing vApp (if needed), and then we test the LAPSUS. We define that a LAPSUS is verified to be effective when it is *incorrectly* interpreted by the VA system. For example, if the vApp is not installed after an installation-related LAPSUS is played to VA devices, then this LAPSUS is effective.

V. IMPLEMENTATION

A. LipFuzzer

We implement a prototype LipFuzzer using Python. BNs are constructed with two components: DAG graphs with weights (a unique ID is assigned to each vertex/node) and corresponding logic functions to these nodes. A model query specifies the starting point of a path search, then unrepeated paths are returned as query results. All the query results from different models are aggregated to remove repeated results. Then the results can be sent to the verification step where voice commands are converted to the corresponding audio format.

Speech Synthesis. In order to work with real VA platforms, We generate voice command records to verify the effectiveness of fuzzing results. For machine-generated voice commands, we use two types of speech synthesis methods to generate LAPSUS. The first speech synthesis method (for Phoneme-level) is phoneme-to-speech that is used for phoneme based fuzzing results. In the NLP preprocessing, ARPABET phonetic code representation [8] is used (with CMU Dict). However, this code is not found in use for speech synthesis. We still need to translate it into IPA (International Phonetic Alphabet) phonetic representation for speech output with tools such as ESpeak [15]. We use each vApp platforms' native speech synthesis tools: Amazon Polly [6] and Google Cloud TTS [9]. The second speech synthesis method is for vocabulary and grammar levels. We direct input fuzzing results generated with LipFuzzer to the above mentioned services and perform a Text-to-Speech conversion.

B. User Study

We are interested in templates used in real vApps, thus our evaluation should have real-world vApp developers and users involved. However, it is difficult to directly acquire private data from VA providers such as Amazon Alexa, Google Assistant. Thus, we need to recruit volunteers to study how developers define voice commands templates and how actual VA users interact with VA devices. We use Amazon Mechanical Turk (MTurk) crowdsourcing platform [5] for collecting data related to both using and developing vApps.

TABLE III: LAPSUS Examples Collected from Real Users

	Correct Form	LAPSUS	LAPSUS Type
Installation Name	"Airport Security Line Wait Times"	"Airport Security Wait for Line"	Grammar
		"Airport Security Line **Waiting** Time"	Grammar
		"Airport Line Wait Times"	Vocabulary
	"Thirty Two Money Tip with Nick True"	"Thirty Two Money Tip with Nick **Truth**"	Pronunciation
		"Thirty Two Money Tip with Nick **Drew**"	Pronunciation
		"Thirty Two Money **Trip** with Nick **Truth**"	Pronunciation
	"Elon - Tesla Car"	"Elon Tesla Car"	Pronunciation
Invocation Voice Command	"Alexa, ask Elon to turn on the climate control"	"Alexa, ask Elon **Musk** to turn on the climate control"	Vocabulary
	"Alexa, ask massage manager begin session for number five"	"Alexa, ask massage **messenger** begin session for number five"	Pronunciation

Remarks: 1) The red, bold mark indicates the words where errors exist. 2) The dash symbol "-" in "Elon -Tesla Car" is treated as an unnaturally long pause between "Elon" and "Tesla" when matching the voice commands.

In using MTurk, we create survey assignments to emulate different scenarios in a vApp lifecycle. We have two groups of users: vApp developer group and vApp user group. For a vApp developer, we collect how many LAPSUS a normal developer can consider when developing a vApp. We emulate the process of vApp development to let a normal developer (with some background in computer science) create a template in our survey. There are 60 MTurk workers involved in our developer-group user study.

For a vApp user, we collect three types of data. First, we collect training data for initializing LAPSUS Models. Second, we record voice commands to see what users would say to install Alexa vApps using Skill Names. Third, we ask users to repeat full voice command usages (to emulate the real vApp usage they cannot access the shown/played example commands while speaking). For a solely speech-hearing setting as mentioned earlier, participants are asked to remember Skill Names or voice commands shown (or audio sound) to them and repeat these Skill Names and voice commands with real Alexa vApps. Their voices are recorded and verified with the actual Alexa device. In total we have 150 MTurk workers involved in our user-group user study.

Disclaimer. We conduct the user study under the permission of the University Institutional Review Board (IRB). Also for ethical reasons, we do not include any harmful code in the uploaded testing vApps when demonstrating our work.

VI. EVALUATION

Prior work already mentioned that pronunciation errors (we regard this as a type of LAPSUS) exist in using vApps, and could be used to launch vApp squatting attacks [28]. In previous sections, we illustrated that vApp squatting attack is mainly caused by the vulnerable Intent Classifier. In this section, we empirically verify whether this is the case, and whether our LipFuzzer can systematically discover those vulnerable templates. More specifically, our evaluation has three goals:

(i) We empirically verify that the problematic Intent Classifier can lead to speech misinterpretation related to LAPSUS.

(ii) We show LipFuzzer's performance in terms of the LAPSUS Models' accuracy and effectiveness.

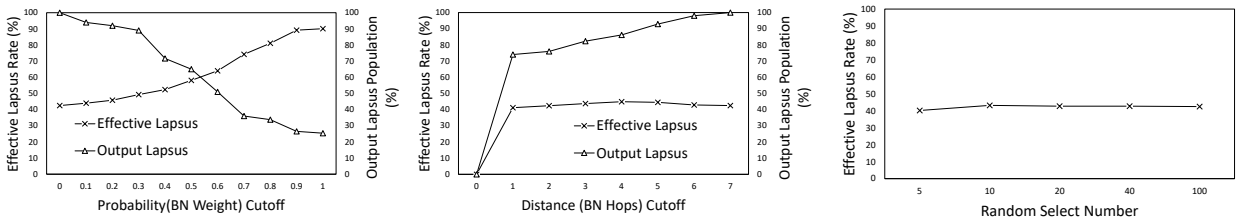
(iii) We use LipFuzzer to reveal that problematic templates widely exist in both Amazon Alexa and Google Assistant platforms.

A. Intent Classifier Evaluation

First of all, we want to verify that the vulnerable Intent Classifier, rather than ASR, should be mainly blamed for incurring semantic misinterpretation. We first leverage user-group data from the user study to locate LAPSUS. Then, with these LAPSUS voice commands, we input synthesized audios (so that ASR processing is guaranteed to be correct) to the Amazon Echo device to check if the semantic inconsistency still exists.

1) Experiment Setup: For the first 40 users in the user study, we randomly select 30 Alexa vApps (from the pool of top 20 Skills in each category) with example voice commands provided for them to emulate the vApp usage. As a result, we collected 521 audio records. These audios are collected and played to the Alexa Echo device in our lab.

2) What are the real LAPSUS*?* The goal of this experiment is to confirm the existence of LAPSUS in the real world. From the collected audio records, we first remove unqualified audio samples: low-quality recording (usually too much noise and not recognizable) and incomplete voice commands. After that, we play filtered audio (312/521) to the device and make sure they will not interfere with each other (by ending a skill



(a) Probability Cutoff (b) Distance Cutoff (c) Random Select

Fig. 7: LAPSUS Cutoff Selection Strategies

session after each play). As a result, 61.86% (193/312) of them are processed with intended skills and functionality. This means that 38.14% of the recorded voice commands are LAPSUS. We showcase examples of these LAPSUS in Table III. In this table, we can observe all three types of LAPSUS in this initial experiment. Note that for the installation stage voice commands, we use standard installation commands, "Alexa, enable `[SKILL INSTALLATION NAME]`."

3) How much do templates contribute to vApp squatting? We use the 119 identified LAPSUS and try our best to remove the effect of ASR issues. Then, the remaining LAPSUS will be confirmed to be caused by improper semantic interpretation of the Intent Classifier.

We first manually transcribe those LAPSUS audios into text. We still use MTurk with each audio transcribed by 2 workers (the results are correlated). We use 109 successfully transcribed results that are agreed by 2 workers who work on the same LAPSUS audio. Next, the transcribed texts are processed by speech synthesis tools. We play them to the Amazon Echo again to check if they can be processed with intended skills and functionality.[5] Our result shows that 77% (84/109) still incur the semantic inconsistency, which means LAPSUS still exists. We notice that most of the remaining 25 ASR-induced LAPSUS are caused by accent. In conclusion, we empirically verified that the Intent Classifier contributes most in the problem of semantic misinterpretation.

B. LipFuzzer Evaluation

We now evaluate LipFuzzer's accuracy and effectiveness. First, we present how different cutoff strategies perform in generating LAPSUS. As a result, the probability cutoff strategy is selected because it conducts the fuzzing more accurately. Second, we show that our refinement can further improve the accuracy. Specifically, we showcase the fuzzing accuracy in terms of the effective LAPSUS number (described in Section IV) over the total number of generated test cases (i.e., output LAPSUS). Third, we present the fuzzing effectiveness with average LAPSUS produced from a seed template (i.e., a voice command).

1) Experiment Setup: We choose to test our tool with the top 20 vApps in each category (based on page ranking). Thus, we have a total of 460 templates with 1104 voice commands. For the LAPSUS Models (all three levels: pronunciation, vocabulary, and grammar), we chose both refined and initial models for experimenting. In refined models, collected LAPSUS results from Section VI-A are used to improve LAPSUS Models similar to the process of model building (shown in Figure 5).

2) Cutoffs for Different Query Strategies: Using cutoff is important in guiding linguistic-model-guided fuzzing because there can be many ineffective fuzzing results (i.e., LAPSUS which only have a low possibility of being spoken by users). Thus, we examine how different cutoffs will affect the effective LAPSUS rate. Our goal is to find a cutoff criterion that is best for producing useful LAPSUS. Note that, during the fuzzing, we only get one mutation for each voice command.

We present evaluation (in Figure 7) for different strategies: two strategies mentioned in Section IV-C and a randomized approach. We sample the results with 0.1 probability. Figure 7(a) shows that probability cutoff can be used to produce more LAPSUS with lower percentage of ineffective LAPSUS. In other words, the probabilities (BN weights for edges) can be used to guide the fuzzing effectively. Also, both distance-based and random selection strategies, shown in Figure 7(b)&(c), are not able to reduce the searching space successfully.

The Cutoff Value. In the rest of our evaluation, we empirically choose a probability cutoff value of 0.483 to be a threshold for model query. That means any query results within the cutoff can be used to generate LAPSUS. We use this value for our prototype demonstration. By applying this cutoff value, we cut off 49.9% of generated LAPSUS population with the effectiveness rate increased to 59.52%.

[5]We also verified the ASR audio-to-text transcription results (in Alexa web portal Setting/History) with synthesis text input, and they are the same.

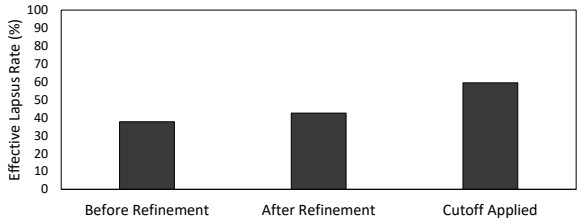


Fig. 8: Fuzzing Accuracy

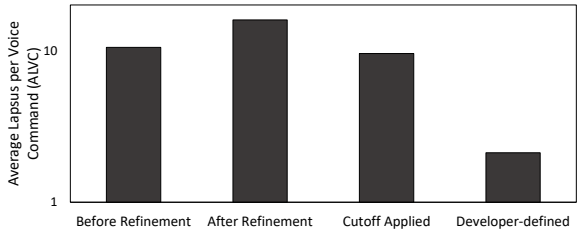


Fig. 9: Fuzzing Effectiveness

3) Accuracy: Then, in Figure 8, we show the rates of effective LAPSUS with refinement and cutoff used. After the refinement, the effective LAPSUS rate is increased from 37.7% to 42.5%. Then, with a cutoff applied, the rate is further increased to 59.52%. This indicates that the refinement and the cutoff can indeed improve the linguistic models' accuracy.

4) Effectiveness: Next, we show that the LipFuzzer can effectively locate problematic templates. We use the metric of Average LAPSUS per Voice Command (or ALVC) to evaluate the effectiveness of locating LAPSUS. In Figure 9, we show that, the refined LAPSUS Models provide more LAPSUS (i.e., from 10.5 to 15.92 ALVC) with 1,104 seed templates. Applying the cutoff, will then reduce the ALVC to 9.57 because fewer states in the models are involved.

Furthermore, we show that LipFuzzer can identify the semantic inconsistency in the real world. We compare templates defined by developers (from the user study) and LAPSUS Model guided fuzzing results to check if LAPSUS Model is effective in finding the semantic inconsistency caused by the misinterpretation in the Intent Classifier. Specifically, to find ALVC data from mock vApp development, we gathered 60 MTurk workers who have basic engine ring background (i.e., using MTurk's filtering function). We show them example voice commands, and let them think of what possible variations a vApp user could speak (note that we intentionally let these developers think of LAPSUS). In total, we evaluated 300 voice commands with an ALVC of only 2.12. Thus, we can empirically confirm that our proposed linguistic-model-guided fuzzing approach is significantly better in finding more LAPSUS than manual effort (which is less efficient and scalable).

5) Time Overhead: The current LipFuzzer implementation takes an average 11.4 seconds per seed template fuzzing (excluding the verification). This large time overhead is mainly caused by the slow local NLP pre-processing (based on our implementation and regular PC setting). More specifically, it is due to the time-consuming dictionary-based searching. However, the NLP pre-processing is only used for new seed inputs. Thus, it is a one-time processing overhead. We consider it reasonable for an offline fuzzing tool. Moreover, it could be optimized with cloud outsourcing and NLP pipeline optimization in the future.

C. vApp Store Evaluation

In this section, we further apply LipFuzzer to evaluate real-world vApp stores.

1) Experiment Setup: We evaluate LipFuzzer by using templates crawled from the Amazon Alexa Store and Google Assistant Store. For Amazon Alexa Store, we acquired a seed template dataset of 98,261 voice commands from 32,892 vApps. For Google Assistant Store, we gathered 2,328 vApps with 9,044 voice commands. Moreover, we apply the same LAPSUS Models used in LipFuzzer evaluation.

2) Potentially vulnerable vApps: We define a potentially vulnerable vApp as one that uses voice command template(s) that have corresponding LAPSUS[6] generated by LipFuzzer. We note that they may not be actually vulnerable because some LAPSUS may be ineffective for a vApp (e.g., LAPSUS trigger nothing unintended). As shown in Table IV, for Alexa vApps, LipFuzzer generated 497,059 LAPSUS with 32,892 crawled vApps. Among them, 26,790 vApps are potentially vulnerable to hijacking attacks. Similarly, our tool finds that 2,295 Google Assistant vApps are potentially vulnerable.

3) Verified vulnerable vApps: We further evaluate what percentage of potentially vulnerable vApps can be actually vulnerable. It is worth noting that it takes a very long time to verify all related LAPSUS (e.g., 30 seconds to 1 minute for just one LAPSUS verification using a real vApp device). Thus, we use a sampling approach in this work. To be specific, we randomly pick 1,000 Amazon Alexa vApps and 200 Google Action vApps for real device verification. In the verification, we choose installation- (or invocation-) related LAPSUS from selected vApp's fuzzing result.[7] Then, we automat-

[6]We exclude those LAPSUS originated from key words or wake-up words.

[7]Note that we do not use non-installation/invocation LAPSUS because, without vApps' internal context, responses are difficult to judge whether successful or not. For example, it is not straightforward to find the difference between the functionalities in the same vApp using an automatic and scalabe way.

TABLE IV: vApp Store-wide Fuzzing Results

Store Name	Crawled vApp #	LipFuzzer-generated LAPSUS #	Potentially vulnerable vApp #	Verified vulnerable vApp % (Sampled)	Potentially vulnerable vApps # Zhang et al. [37]	Vulnerable vApps # Kumar et al. [28]
Amazon Alexa	32,892	497,059	26,790	71.5%	531	25
Google Assistant	2,328	11,390	2,295	29.5%	4	N/A

Remark: N/A means not applicable because Google Assistant was not evaluated in the work.

ically play voice commands and transcribe the response. Next, we determine if a vApp is triggered based on a few conservative heuristics, e.g., if Amazon Alexa returns "I can't find that skill", then it is an effective LAPSUS and thus the corresponding vApp is vulnerable. As a result, a total of 715 (71.5%) Amazon Alexa vApps and 59 (29.5%) Google Assistant are verified to be vulnerable. We note that these percentages only represent *lower* bounds for the actual vulnerable vApps, because we only use very conservative heuristics in the automatic verification. For example, our verification will not report vulnerable when there is no response (silence), because both vulnerable and not vulnerable vApps can have no response, and without detailed understanding of the context/function of the vApp, it is hard to tell. We found that this occurred a lot in both platforms, particularly in Google Assistant vApps.

4) Result Comparison: We also compare our fuzzing results with existing [28] or concurrent work [37] and show that LipFuzzer can find much more vApps that can potentially lead to speech misinterpretation. Compared with our automated and systematic approach, existing works are limited. For example, in [28], the authors manually found 25 vulnerable skills, and in [37], only a small number of vApps were found to be potentially vulnerable.

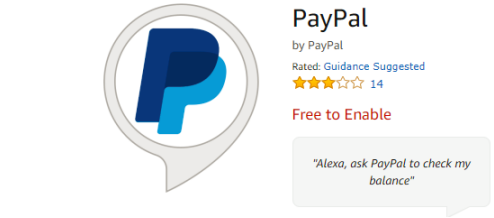
TABLE V: LAPSUS for Example vApps

Intended Voice Command	LAPSUS	Effective LAPSUS?
"Paypal" (installation)	"Pay-ple"	✓
	"Pay-ples"	✓
"ask PayPal to check my balance"	"ask PayPal to check my balances"	✗
	"ask PayPal to check my balancing"	✗
	"ask PayPal to check my balancing"	✗
	"ask PayPal to checks my balance"	✗
	"ask PayPal to checking my balance"	✗
"Skyrim Very Special Edition" (installation)	"Skyrim Very Special Edit"	✓
	"Skyrim Special Edition"	✓
	"Skyrim Very Specially Edition"	✗
	"Sky-ram Special Edition"	❍
	"Sky-im Special Edition"	❍

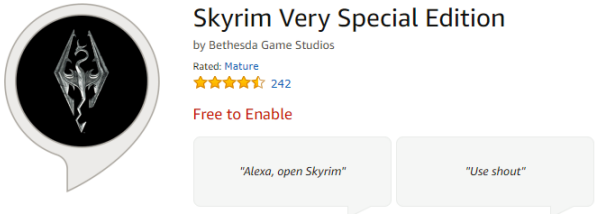
✓: Effective, ✗: Ineffective, ❍: Maybe Effective

D. Case Study

1) Using LipFuzzer on real vApps: We demonstrate how LipFuzzer works on two popular Amazon Alexa vApps, shown in Figure 10. The results are presented in Table V. First, we observe that Paypal uses a simple, short name as its vApp installation name. As a result, only two pronunciation-based LAPSUS are generated and verified from LipFuzzer. Similarly, Paypal uses straightforward voice commands after installed, and LipFuzzer only reports simple LAPSUS which are already protected by the system's default fuzzy matching. The Skyrim game vApp, on the other hand, is shown to be more vulnerable. Its name is long, and many effective LAPSUS are generated using LipFuzzer. Note that the ❍ mark means that, when verifying the LAPSUS, the VA responds with a guess rather than "not found".



(a) Paypal Alexa vApp



(b) Skyrim Alexa vApp

Fig. 10: Example Alexa Skills

Security Implication: The aforementioned results show that simple and unique names are more difficult to be misspoken according to our linguistic models. With the current template-based Intent Classifier design, using simple and unique voice commands is the most effective way of preventing vApp squatting attacks. However, it is difficult to achieve given the increasingly growing vApp population. LipFuzzer, in an adversarial setting, provides an automatic and scalable way to find these unsafe voice commands.

2) Attacking "True Bank": To demonstrate the practicality of our linguistic-model-guided fuzzing, we show a real attack case study. In Figure 11, we present an

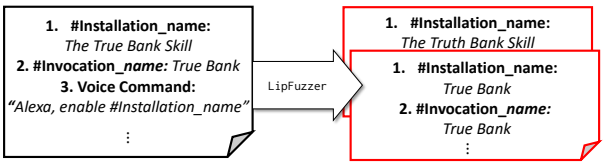


Fig. 11: Attacking vApp with LAPSUS

example Interaction Model (i.e., a set of voice command templates) of a victim Alexa vApp which we created. To attack it, we use LipFuzzer-generated LAPSUS to create 4 malicious vApps which were uploaded to the Alexa Skill Store.

To evaluate whether users would actually speak those LAPSUS, we conducted a simple user study. We collected 30 users' audio recordings from the user-group by showing them example voice commands (from victim vApp), and allowing them to speak to the VA. Thereafter, we verified the recorded voice commands with Amazon Alexa. As a result, a total number of 3 malicious vApps are invoked, which clearly demonstrate the practicability of the attack. Note that, after this evaluation, all experimental vApps have been removed from the store.

VII. RELATED WORK

Attacking ASR through Acoustic Channels. For audible voice command attacks, Diao et al. [23] proposed to play prepared audio files using non-hidden channels that are understandable by a human listener. Tavish et al. [34] then presented Cocaine Noodles that exploits the difference between synthetic and natural sound to launch attacks that can be recognized by a computer speech recognition system but not easily understandable by humans. To achieve a better result, a white box method [19] was used based on knowledge of speech recognition procedures. With complete knowledge of the algorithms used in the speech recognition system, this improved attack guarantees that the synthetic voice commands are not understandable by a human. For the threat model, Audible Voice Command Attacks require the attackers' speakers to be placed at a physical place near victims' devices ([19] fails with distances over 3.5 meters). Recently, CommandSong [35] was proposed to embed a set of commands into a song, to spread to a large amount of audience.

More powerful attacks using inaudible voice commands were proposed in [36] [33]. They leverage non-linearity of microphone hardware used in almost all modern electric devices. Humans are not able to recognize sound with frequency over 20 kHz, and microphone's upper bound of recordable range is 24 kHz. For this threat model, Inaudible Voice Command Attacks also require the attackers' devices to be physically close to the victims' devices (in feet range).

Attacking ASR with Misinterpretation. Kumar et al. [28] presented an empirical study of vApp squatting attacks based on speech misinterpretation. Moreover, as a concurrent work, [37] also showcases a similar approach to exploit the way a skill is invoked, using a malicious skill with similarly pronounced name or paraphrased name to hijack the voice command meant for a different skill. Different from them, our work is the first to systematically explore the root-cause in NLU and Intent Classifier, and create the first linguistic-model-guided fuzzing tool to uncover significantly more vulnerable vApps in existing VA platforms.

VIII. DISCUSSION

We acknowledge that BNs could be computationally expensive and very data dependent. Different methods such as Neural Networks and other machine learning techniques may be applied in the future to improve LAPSUS Models. However, in this work, we think that BN is sufficient to demonstrate the effectiveness.

In this work, we only collect publicly accessible voice commands for templates, and there are more defined in templates but inaccessible. In the future, we plan to collect a much larger dataset of LAPSUS in real-world usage. With a more complete dataset, we could produce LAPSUS more effectively.

Our future work will also improve the model with more complicated logical representations so that we can cover more LAPSUS. For example, we could use predicate logic to represent hybrid LAPSUS across different models. Improving the training and mutation process of LipFuzzer is another direction of our future work.

IX. CONCLUSION

In this paper, we systematically study how Intent Classifier affects the security of popular vApps. We first find that the currently used vApp templates can incur dangerous semantic inconsistencies. Then, we design the first linguistic-guided fuzzing tool to systematically discover the the speech misinterpretations that lead to such inconsistencies. We plan to publish our source code and dataset used in this work to help not only vApp developers but also VA system designers to create better templates with fewer speech misinterpretations. This will also benefit the research community to work on this important IoT security area.

ACKNOWLEDGMENT

This material is based upon work supported in part by the National Science Foundation (NSF) under Grant

no. 1816497 and 1700544. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] 30 common grammar mistakes to check for in your writing. https://blog.hubspot.com/marketing/common-grammar-mistakes-list.

[2] Alexa skill interaction model. https://developer.amazon.com/docs/alexa-voice-service/interaction-model.html.

[3] Amazon alexa in-skill purchasing. https://developer.amazon.com/blogs/alexa/post/b8101123-f1b9-494c-8bbb-53e3850a1123/in-skill-purchasing-offers-jeff-bolton-s-voice-business-a-new-level-of-monetization.

[4] Amazon echo recorded and sent couple's conversation all without their knowledge. https://www.npr.org/sections/thetwo-way/2018/05/25/614470096/amazon-echo-recorded-and-sent-couples-conversation-all-without-their-knowledge.

[5] Amazon mechanical turk crowdsourcing platform. https://www.mturk.com/.

[6] Amazon polly: Turn text into lifelike speech. https://www.mturk.com/.

[7] Amazon prize: The socialbot challenge. https://developer.amazon.com/alexaprize/2017-alexa-prize.

[8] Arpabet phonetic transcription codes. https://en.wikipedia.org/wiki/ARPABET.

[9] Google cloud text-to-speech. https://cloud.google.com/text-to-speech/.

[10] Introduction to information retrieval: Stemming and lemmatization. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.

[11] Natural language understanding google ai. https://ai.google/research/teams/nlu/.

[12] Oxford living dictionary - confusable words. https://www.merriam-webster.com/dictionary/confusable.

[13] Slips of the tongue. https://www.psychologytoday.com/articles/201203/slips-the-tongue.

[14] Stanford corenlp. https://stanfordnlp.github.io/CoreNLP/index.html.

[15] Text-to-speech tools. https://help.ubuntu.com/community/TextToSpeech.

[16] Wikipedia speech error. https://en.wikipedia.org/wiki/Speech_error.

[17] M. Böhme, V.-T. Pham, M.-D. Nguyen, and A. Roychoudhury, "Directed greybox fuzzing," in *Proc. of the 2017 ACM CCS*, 2017.

[18] C. Boitet and M. Seligman, "The whiteboard architecture: A way to integrate heterogeneous components of nlp systems," in *Proceedings of the 15th conference on Computational linguistics-Volume 1*, 1994.

[19] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, "Hidden voice commands." in *USENIX Security Symposium*, 2016, pp. 513–530.

[20] G. S. Dell, "Representation of serial order in speech: Evidence from the repeated phoneme effect in speech errors." *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 10, no. 2, p. 222, 1984.

[21] G. S. Dell, C. Juliano, and A. Govindjee, "Structure and content in language production: A theory of frame constraints in phonological speech errors," *Cognitive Science*, vol. 17, no. 2, pp. 149–195, 1993.

[22] G. S. Dell, K. D. Reed, D. R. Adams, and A. S. Meyer, "Speech errors, phonotactic constraints, and implicit learning: A study of the role of experience in language production." *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 26, no. 6, p. 1355, 2000.

[23] W. Diao, X. Liu, Z. Zhou, and K. Zhang, "Your voice assistant is mine: How to abuse speakers to steal information and control your phone," in *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. ACM, 2014, pp. 63–74.

[24] A. W. Ellis, "Errors in speech and short-term memory: The effects of phonemic similarity and syllable position," *Journal of Verbal Learning and Verbal Behavior*, vol. 19, no. 5, pp. 624–634, 1980.

[25] L. Getoor and B. Taskar, *Introduction to statistical relational learning*. MIT press Cambridge, 2007, vol. 1.

[26] J. H. Hulstijn and W. Hulstijn, "Grammatical errors as a function of processing constraints and explicit knowledge," *Language learning*, vol. 34, no. 1, pp. 23–43, 1984.

[27] K. Kersting, L. De Raedt, and S. Kramer, "Interpreting bayesian logic programs," in *Proceedings of the AAAI-2000 workshop on learning statistical models from relational data*, 2000, pp. 29–35.

[28] D. Kumar, R. Paccagnella, P. Murley, E. Hennenfent, J. Mason, A. Bates, and M. Bailey, "Skill squatting attacks on amazon alexa," in *27th USENIX Security Symposium*, 2018.

[29] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proc. of the ACL Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics*, 2002.

[30] A. S. Meyer, "Investigation of phonological encoding through speech error analyses: Achievements, limitations, and alternatives," *Cognition*, vol. 42, no. 1-3, pp. 181–211, 1992.

[31] R. Pfau, *Grammar as processor: A Distributed Morphology account of spontaneous speech errors*. John Benjamins Publishing, 2009, vol. 137.

[32] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[33] N. Roy, H. Hassanieh, and R. Roy Choudhury, "Backdoor: Making microphones hear inaudible sounds," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 2–14.

[34] T. Vaidya, Y. Zhang, M. Sherr, and C. Shields, "Cocaine noodles: exploiting the gap between human and machine speech recognition," *WOOT*, vol. 15, pp. 10–11, 2015.

[35] X. Yuan, Y. Chen, Y. Zhao, Y. Long, X. Liu, K. Chen, S. Zhang, H. Huang, X. Wang, and C. A. Gunter, "Commandersong: A systematic approach for practical adversarial voice recognition," in *27th USENIX Security Symposium*, 2018.

[36] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "Dolphinattack: Inaudible voice commands," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 103–117.

[37] N. Zhang, X. Mi, X. Feng, X. Wang, Y. Tian, and F. Qian, "Dangerous skills: Understanding and mitigating security risks of voice-controlled third-party functions on virtual personal assistant systems," *IEEE Security & Privacy*, 2019.