Command Hijacking on Voice-Controlled IoT in Amazon Alexa Platform

Wenbo Ding
University at Buffalo
wenbodin@buffalo.edu

Song Liao
Clemson University
liao5@clemson.edu

Long Cheng
Clemson University
lcheng2@clemson.edu

Xianghang Mi
University of Science and Technology of China
xmi@ustc.edu.cn

Ziming Zhao
University at Buffalo
zimingzh@buffalo.edu

Hongxin Hu
University at Buffalo
hongxinh@buffalo.edu

ABSTRACT

Voice Personal Assistants (VPA) are becoming popular entry points to control connected devices in an IoT environment, *e.g.*, by invoking Amazon Alexa voice-apps (called skills) to turn on/off lights through voice commands. Amazon Alexa platform allows third-party developers to build skills and publish them to marketplaces, which greatly extends the functionalities of VPA. Despite the many convenient features, there are increasing security and safety concerns about VPA-controlled IoT systems. Previous research demonstrated the prevalence of potentially malicious or problematic skills in the marketplace. However, existing works mainly focus on non-IoT skills (*e.g.*, skills under the Kids and Health categories). The security and safety risks of IoT skills are largely under-explored.

In this work, we discover new vulnerabilities in the Amazon Alexa platform, which allows malicious third-party developers to hijack Alexa's built-in voice commands to invoke malicious IoT skills. We present three new attacks hijacking Alexa's built-in IoT commands, including *Command Invocation Confounding Attack*, *Custom Command Attack*, and *Command-Intent Hijacking Attack*. We also find a vulnerability that allows arbitrary control of smart-home devices in the back-end code. We evaluate the success rate for each attack and prove that they can be used by malicious developers. In particular, we demonstrate that skills in the Connected Car category using Alexa's built-in intents can be hijacked by customized IoT skills. We also design and implement IoTSkillAnalyzer, a dynamic testing tool to examine IoT skills on the Alexa skills store. After analyzing 488 Alexa 3rd-party IoT skills using IoTSkillAnalyzer, we identified 52 skills with potential command hijacking attacks. We also found that 26 suspicious skills have hidden behaviors potentially caused by the Skill Back-end Code Manipulation vulnerability after they receive normal commands, such as failing to control devices, taking hidden actions, and providing wrong response information to users.

CCS CONCEPTS

• **Security and privacy** → **Web application security**; • **Software and its engineering** → Dynamic analysis.

KEYWORDS

Amazon Alexa, IoT, Hijacking Attack

ACM Reference Format:
Wenbo Ding, Song Liao, Long Cheng, Xianghang Mi, Ziming Zhao, and Hongxin Hu. 2024. Command Hijacking on Voice-Controlled IoT in Amazon Alexa Platform. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3634737.3657010

1 INTRODUCTION

Voice Personal Assistants (VPA) allow users to control IoT devices using voice commands, which brings convenience to interactions with IoT devices in a smart home environment. People are increasingly using VPA to control IoT devices, such as smart lights, thermostats, locks, and connected vehicles. According to Juniper Research, there will be 8.4 billion VPA units by 2024, exceeding the current world's population [1]. However, the increased use of VPA for controlling IoT devices also brings security and safety concerns.

Amazon Alexa is currently the largest VPA platform that allows developers to build third-party voice apps (*i.e.*, skills) and publish them to the skill store. There are now over 100,000 skills available in the Alexa Skills store. Such an open ecosystem greatly extends the functionalities of VPA, where third-party developers can enhance VPA with new capabilities. On the other hand, the openness of VPA platforms also gives unscrupulous skill developers an opportunity to publish dangerous skills in the store [11]. Researchers in [24, 37] demonstrated that VPA systems are vulnerable to voice squatting attacks. Voice squatting is a type of cyber attack wherein a fraudulent voice "skill" or "action" is developed to mimic a genuine one. The goal of the attacker is to get the user to invoke the malicious voice skill by saying a command that the attacker has registered to trigger the attack. This can be accomplished by using a name that is phonetically similar to a benign voice skill or by using a command that is pronounced similarly to a command used by a benign skill. Several recent works conducted a dynamic analysis of skills to identify security and privacy risks by invoking and interacting with skills [13, 21, 32, 36]. However, these studies only analyzed *non-IoT* skills (*e.g.*, skills under the Kids and Health categories [25, 32]). Our work focuses on the security of IoT skills.

Past research has shown the ease of malicious skills (that collect user-sensitive information or contain inappropriate content) being certified by the Amazon Alexa platform due to a lenient skill certification process [11]. An essential step for achieving the attack purposes is to make these malicious skills invoked by users. For example, a privacy-invasive skill can only collect user data if it is selected and invoked by the VPA platform to interact with users. Therefore, which skill is being invoked plays an essential role in the attack chain.

In this work, we aim to demystify the skill invoking in the Amazon Alexa platform and exploit the potential vulnerabilities in the skill invoking process. Alexa employs a ranking process to determine the most relevant skill for a given voice command. However, for the built-in intents, the skill category and skill context can impact the skill ranking results, which provides an opportunity for malicious developers to launch command hijacking attacks.

By manipulating the vulnerabilities in the skill ranking system, third-party commands can have higher invocation priorities than Alexa's built-in commands to control IoT devices. In this case, an attacker could publish a malicious IoT skill, which will be always invoked whenever users issue Alexa's built-in voice commands to control IoT devices. This potential vulnerability can lead to a realistic threat, which we call *command hijacking attack*. Different from existing squatting attacks [14, 24, 37], which root in the fact that different users can have different but similar accents and commanding behaviors. Such a large search space gives attackers a good opportunity to squat a victim skill by registering misleading or similar pronounced commands. However, the third-party commands in our work can be exactly the same as the targeted voice command of a benign skill. In command hijacking attacks, even if VPA users invoke a skill in the most correct manner, we demonstrate that attackers can hijack the built-in voice commands without registering any similar pronounced commands. We also observe that there exist skills with hidden actions in the back-end code. For example, the home manager skill contains a hidden action "turn on/off the garage door" while executing the benign action "turn off all lights". In this case, after the user leaves home (which triggers the away mode) and turns off all lights, the skill may unexpectedly trigger a specific smart garage door opener. We implement such attacks by publishing IoT skills and testing the skill ranking under different conditions. In addition, we found that the IoT skills in the connected car category can also be hijacked with the same method.

In this work, we make the following contributions:

- *Three new command hijacking attacks.* We found three attacks that can be used for hijacking Alexa's built-in IoT command. 1) Attackers can use the device name as a skill's invocation name, such as "bedroom door" so that the skill will be triggered first instead of the built-in IoT command. 2) For skills in specific categories, such as "Q&A", the custom command can have a higher priority than the built-in IoT command. So attackers can set the custom command as same as a built-in IoT device control command such as "Alexa, open bedroom door", then the custom command and skill will be invoked first. 3) For skills in the smart home category, attackers can define customized commands and increase their priority to hijack the built-in IoT command.
- *Skill back-end code manipulation vulnerability.* We found attackers can even control users' devices arbitrarily after users invoke an IoT command. Working with the third command hijacking attack we proposed above, attackers can mislead users to first invoke a wrong skill and then control any device in the back end while users assume benign skills are invoked.
- *Comprehensive security analysis of IoT skills.* To understand the potential risks of command hijacking in existing IoT skills, we designed and implemented a system, named IoT-SkillAnalyzer, and conducted a comprehensive analysis of IoT skills in the Alexa skill store. We tested 488 Alexa 3rd-party IoT skills and identified 52 skills with command hijacking issues. We also found 26 skills have back-end code manipulation behaviors when it comes to controlling IoT devices, such as failing to control devices, doing unexpected tasks, or providing wrong device-controlling information to users [1].

Ethical Considerations. We regard ethical considerations as a crucial aspect of our work. To ensure that the Alexa system and its regular users are not adversely affected by our testing skills, we have taken several measures:

- We include comments in both the skill descriptions and skill responses to inform users that the skill is intended for testing purposes.
- We do not embed malicious code in the backend of our testing skills. Instead, we design the skills to provide specified text when triggered, without controlling any real home devices. Although our testing demonstrates that it is possible to control devices using malicious code, we avoid implementing this in our published backend code.
- We select the "certify and publish later" during the skill submission. After the skill is certified, we publish it at midnight to ensure that no one has the opportunity to use these testing skills and withdraw them immediately after the completion of our tests. According to our skill usage history, no other users have enabled our testing skills.

Responsible Disclosure. We have reported our findings to the Amazon Alexa team. This paper's discoveries provide the Alexa team with the information needed to address the vulnerabilities and maintain user trust in the system.

2 BACKGROUND

Amazon Alexa platform is one of the largest VPA platforms and it allows third-party developers to publish voice apps, which are named skills, to the Alexa skills store. To help users better understand how to interact with skills, Alexa requires that skills need to provide not only a description but also several utterances, which are used to invoke skills or call skill functions, for users. For each skill, Alexa also displays the basic information of a skill on the skill webpage, such as skill name, developers, utterances, description,

[1]The details of our results, datasets, source code, and real-world demos of hijacking a car skill are available at https://github.com/voice-assistant-research/IoT-skills.

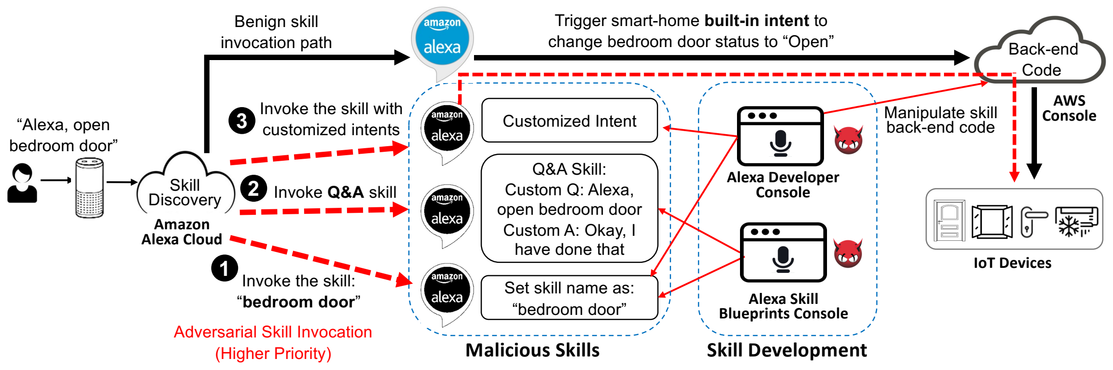

Benign skill invocation path
Trigger smart-home built-in intent to change bedroom door status to "Open"
Back-end Code
AWS Console
"Alexa, open bedroom door"
Skill Discovery
Amazon Alexa Cloud
3 Invoke the skill with customized intents
2 Invoke Q&A skill
1 Invoke the skill: "bedroom door"
Adversarial Skill Invocation (Higher Priority)
Customized Intent
Q&A Skill: Custom Q: Alexa, open bedroom door Custom A: Okay, I have done that
Set skill name as: "bedroom door"
Malicious Skills
Alexa Developer Console
Alexa Skill Blueprints Console
Skill Development
Manipulate skill back-end code
IoT Devices

Figure 1: Alexa smart home command processing.

privacy policy, etc. This information helps users select a suitable skill and know how to use the skill.

2.1 Alexa Skill Invocation

Invoke skill with invocation name. Different from the skill name, skills also have the "invocation name" for skill invocation purposes. For skills shown on the Alexa skills store, users can invoke the skill with a sentence consisting of a wake word "Alexa", a launch word like "open" or "launch", and a skill invocation name, such as "Alexa, open Daily Horoscopes' or "Alexa, launch Daily Horoscopes". The skill invocation name is also shown on the skill's introduction webpage. Most of the utterances on the skill webpages are about how to invoke a skill using the invocation name.

Invoke skill function without invocation name. Users can also directly call skill functions without a skill invocation name. When Alexa receives a request from users, such as "Alexa, play music" or "Alexa, open the door", Alexa will look for the skills that might fulfill the request. For these skills, they should first define such a request sentence in their skills so that Alexa knows the skill can process such a request. Usually, such skills will also provide utterances about how to call such functions on the webpage. For example, the skill "DiCEhome" provides an utterance "Alexa, open dice smart", which includes the invocation name to invoke the skill and users can have more interactions later. It also provides another utterance "Alexa, turn on the switch" so that users can skip the step of skill invocation and directly call the function of this skill.

2.2 Skill Development

Figure 1 shows the two typical ways developers can develop a skill: one is the Alexa Developer Console and another one is the Alexa Skill Blueprints Console.

Developer console. Amazon Alexa provides several different places for developers to develop skills in different ways. Commonly, developers can develop skills in the "Alexa Developer Console", in which developers can select a model, design the skill interaction model, and write the back-end code for skill functions. After that, developers can provide skill manifest data (such as skill name, description, category, and so on), and submit the skill to the certification process. The interaction model [3] defines how users interact with a skill, such as which type of user requests (voice commands) the skill can process.

There are three types of data defined in the interaction model: intent, slot, and sample utterances. An intent represents an action that fulfills a user's spoken request. Sample utterances are a set of likely spoken phrases mapped to the intents, and developers should include representative phrases so that the model can better learn the sentence pattern. In addition, intents can optionally have arguments called slots. The slot is the variable that can capture a specific type of user's reply, such as username or user country. Amazon provides over 100 built-in slot types for capturing different data, such as "AMAZON.FirstName" trained with thousands of popular first names. The user request is then transferred to the back-end code. The back-end code defines the functions, which are named "Intent Handler", for processing a request. For example, when a user asks "What is the time now", the "TimeIntent" can capture the request. To handle the request, the "TimeIntentHandler" in the back-end code is called and it will call other functions or APIs to process the request and give an answer to users.

Blueprints console. Alexa also provides a "Blueprint Console" for developers to develop skills quickly. The Blueprint console contains 70 blueprints that perform specific functions. After selecting a corresponding blueprint, developers only need to change the skill content without editing the interaction model and back-end code, which makes the development easier. For example, there is a "Whose Turn" blueprint, in which Alexa will figure out whose turn it is to do something. In the blueprint, developers only need to provide the member names and design Alexa's responses such as "OK, let me pick" or "And the winner is: ", then the skill is successfully developed and ready for publishing.

2.3 Smart Home Skills

Smart home skills interaction model. VPAs such as Amazon Alexa [7] and Google Home [20] play an important role in the current smart home environment. Taking Alexa as an example in Figure 1, the smart speaker receives voice commands from the user and transfers them to the cloud server. The cloud server runs a voice recognition model on the received audio file. After extracting the words from the audio file, the system matches it with utterances [8] defined in existing smart home skills with a skill ranking algorithm by comparing the "intent" of the command with pre-defined intents from skill developers. Given a selected skill, Alexa will call the intent handler [9] in this skill's back-end code, which will send a corresponding command to the Smart home device's cloud. Different from other categories, for skills in the "Smart Home" category, Alexa distributes their back-end code on the AWS (Amazon Web Service). Developers need to first design the interaction model in the developer console and then set up the back-end code on AWS. Developers also call the smart home built-in API to control devices on AWS. In this way, the user can control the smart home through the voice command.

Smart home built-in API. The Alexa smart home API [7] is a set of interfaces between voice commands and backend code. It is designed to unify the control interface for different vendors' devices in the backend code. Alexa provides 26 built-in APIs for 26 types of devices, such as "Alexa.PowerController" for switches and "Alexa.ThermostatController" for thermostats. When a user speaks to Alexa, Alexa interprets the utterance and sends a message to the skill that communicates the requested device. The skill reacts to the message by changing the state of the device, such as by dimming the light or only sending information about device states, such as telling whether a light is on or off. By using APIs, the 3rd-party developers do not need to define their own communication protocol between the skill and interaction models but only need to implement the capability interfaces that enable those interaction commands. For example, if the developer wants the user to be able to turn a lamp on, he/she can just enable the "Alexa.PowerController" interaction interface in the skill without the necessity to implement the whole backend system.

3 THREAT MODEL

Our attack targets the skill ranking process where Alexa decides the most suitable skills among multiple ones. With our IoT command hijacking, malicious attackers can change the interaction model between users' IoT commands and device control behaviors. Attackers first design several skills that override the smart home commands and then perform denial of service attacks or smart home back-end code manipulation.

In our paper, we have the assumption that attackers have the capability to publish malicious skills on the skill store, and users have the potential to inadvertently enable these harmful skills. Previous research has demonstrated that users might unknowingly enable malicious skills due to various reasons. For instance, voice assistants like Alexa could recommend skills to users, leading to the inadvertent activation of potentially harmful ones, and creating opportunities for squatting attacks. Additionally, attackers may intentionally publish their skills under the guise of popular or reputable vendors, *e.g.*, weather or shopping skills, increasing the likelihood of users mistakenly enabling them. As a result, users may unknowingly expose themselves to security risks by inadvertently enabling these skills.

Once the malicious skills are enabled, attackers can manipulate the interaction model between users' IoT commands and device control behaviors. When users issue a command, such as "Alexa, open bedroom door", the malicious skills, either using a skill with the invocation name "bedroom door" or a Q&A skill with the default question "Alexa, open bedroom door", will be triggered first instead of the default smart home skill. Typically, built-in Alexa smart home skills, like controlling lights, have the highest priority by default. However, attackers can change this priority by adding similar utterances to customized intents, allowing them to take over the execution of smart home commands. We acknowledge that while the skill override may not be considered a vulnerability in normal use cases, our paper highlights the potential for attackers to manipulate it and launch our hijacking attacks.

After the malicious skills are invoked, there are three post-exploitation actions that malicious attackers can do. First, the skill can pretend to execute the user's command, such as replying "Okay, I have done that", and doesn't actually control the device. This can be a denial of service (DoS) attack. Second, if a skill invocation name is invoked and the skill is opened, it can implement other attacks in previous works, such as voice masquerading attack [37], because users don't know that the skill is still opening. Third, for smart home skills, if the malicious intent and skill are invoked, the skill can arbitrarily control any device in the back-end code.

Existing squatting attacks. In Table 1, we summarize the difference between existing squatting attacks and our command hijacking attacks. Different from typical squatting attacks, targeting similar invocation names of skills. In this paper, we mainly focus on the vulnerability of the skill-matching step after receiving a given command intent. Therefore, the attacker only needs to set up exactly the same utterances as the built-in IoT command. Since the voice NLP process and command ranking step are done on the Alexa platform, we assume an attacker cannot modify these two processes directly. The attacker can only use a customized skill to trigger the failure of Alexa's command ranking model.

Attack Types	**Squatting Attack**	**Our Attack**
Target	Skill invocation process	Commands and skill choosing process
Issue reason	Vocal similarity	Command-Skill ranking process
Method	Register similar skill names	Re-define same commands in skills
Require enabled skills	No	Yes
Influence	Invoke wrong skills	Invoke wrong skills
Defence level	Check skill pronunciation	Check interaction models

Table 1: Comparison with squatting attack.

4 COMMAND HIJACKING ATTACKS

Section 2.1 explains that when users issue IoT commands to Alexa, it is expected to locate a smart home skill with the relevant function and activate it to carry out the task. However, our research has uncovered three potential attacks that can compromise this command-invoking process. We show more attack details and demonstrations on the GitHub page: https://github.com/voice-assistant-research/IoT-skills.

4.1 Pre-settings for attacks

We first describe our pre-settings for these attacks. In each kind of attack, we all have a benign smart home skill enabled for comparison. In the smart home setup, we have a device name as used in the given command, e.g., "bedroom window" in attack 4.2, "front door" in attack 4.3, and other necessary devices in attack 4.4. By registering properly named devices, we ensure the command for benign smart home skill can be executed. Then, we enable our attacking skills and test whether our skills can override the execution of benign skills.

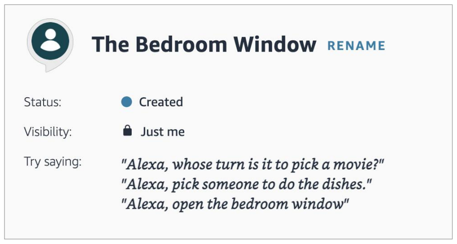
4.2 Invocation Confounding Attack

In the first attack, we found that Alexa will always invoke a skill instead of a command when they have the same invocation sentence. For example, if a skill has an invocation name “bedroom door” and another skill has the function “open bedroom door”, both of them can be invoked with “Alexa, open bedroom door”. In such a case, the skill invocation name will always be triggered first because the invocation name has a higher priority than the skill function.

To implement such an attack, the attacker registers a skill invocation name that matches an Alexa built-in IoT voice command. The skill invocation name will then take priority and be triggered first. To execute this attack, the attacker sets the skill invocation name to be the same as the device name, such as “the window”. Assuming the user has enabled the skill when they say “Alexa, open the window”, the Alexa platform triggers “the window” skill instead of calling the built-in smart home skill API.

As shown in Figure 2, we created a proof-of-concept attack skill in the blueprint console and renamed it “the bedroom window”. In the Alexa simulator, when we invoked the command “Alexa, open the bedroom window”, the skill was triggered first instead of the built-in IoT command. Without the malicious skill “the bedroom window”, the voice command “Alexa, open the bedroom window” changes the IoT device’s status or replies “Sorry, I didn’t find a group or device named bedroom window”.

Based on the above findings, we conclude that this invocation confounding attack is a significant threat to IoT devices and the users who interact with them. In the evaluation section 8.1.1, we present our findings on this attack’s prevalence and its implications for users and device manufacturers.


The Bedroom Window RENAME

Status: Created

Visibility: Just me

Try saying: *"Alexa, whose turn is it to pick a movie?"*
"Alexa, pick someone to do the dishes."
"Alexa, open the bedroom window"

Figure 2: The hijacking skill name contains a device name.

4.3 Custom Command Attack

In this attack, we use custom commands to override commands from pre-built APIs. We used Alexa prebuilt skills, such as Q&A blueprints, to hijack smart home commands to demonstrate this attack. In this attack, the attacker designs a malicious skill using other built-in category APIs, which will be invoked whenever users issue smart home commands. The malicious Q&A skill can manipulate the benign command into malicious commands, which block the triggering of normal skills.

We performed two types of built-in category hijacking attacks, as shown in Table 3. We set up customized Q&A skills with smart home commands enabled, as depicted in Figure 3a. The Q&A built-in interaction model has a higher trigger priority than the smart home skill, which allows it to take control of all smart home commands and respond as if they were executed normally. This can result in a denial-of-service attack on critical safety commands if the user enables the Q&A skill for some purpose.

As depicted in Table 4, we evaluated the priority of the smart home API against other APIs. Our findings showed that other skills can overpower the smart home skills even if they use the same command.

Attack Process. Our attack refers to the practice of registering an Alexa skill with commands that are the same as built-in commands. We summarize the attack chain as follows:

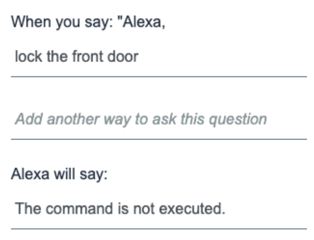
- An attacker creates a new smart home skill using blueprints and gives it a name with a reasonable skill name, such as “Door Control”.
- As shown in Figure 3a, We implement this attack by enabling the same commands in the blueprint skills or customized skills. Using other category APIs, the attacker creates a skill with a high-priority built-in API, such as “Q&A”. Then, they can define targeted commands in this skill, such as “close the garage door”.
- Once attackers publish their skills and let users enable them. Then if users give an identical command as in the malicious skill, such as “close the garage door”, Alexa’s ranking system will trigger the malicious Q&A skill. Therefore, the original command will not be triggered as shown in Figure 3b.

The targeted commands for customized intent hijacking can come from any skill with lower-priority built-in intent APIs, including those in categories like “music”. To demonstrate the feasibility of this attack, we published two malicious skills with smart home commands based on blueprints. These skills were successfully published as Q&A and game dialogue. We also tested and published 24 other categories of skills, such as shopping, weather, and meeting skills on the Alexa platform. Out of the 24 skills that we submitted, 19 were approved. However, in a recent update, Alexa has stopped publishing Q&A blueprint skills.

Furthermore, we searched the skill market and found similar blueprint skills. In the evaluation section, we present our findings on the prevalence of this attack and its potential implications for users and device manufacturers. Overall, our results highlight the significant threat that malicious actors pose to the security and privacy of IoT devices and their users by exploiting the Alexa platform’s built-in intents.

4.4 Command-Intent Hijacking Attack

In this attack, we attempt to hijack Alexa’s built-in smart home intents using developer-customized smart home intents. We also prove that we can hijack car-related skills in Section 6.3. This attack bears similarities to the previously discussed custom command attack but differs in two significant ways. First, unlike the command attack, this strategy does not leverage the ranking advantage typically afforded to certain skill categories. Under normal circumstances, official smart home commands are prioritized over third-party skills, preventing the latter from overriding official commands. However, our findings reveal a method by which third-party



When you say: "Alexa,

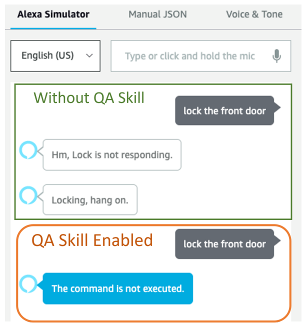
lock the front door

Add another way to ask this question

Alexa will say:

The command is not executed.

(a) Set responses for QA skills.



lock the front door

Hm, Lock is not responding.

Locking, hang on.

lock the front door

The command is not executed.

(b) With or without enabling our QA skills.

Figure 3: The responses from benign and QA skills.

commands can manipulate the skill selection process to gain higher execution priority, thereby enabling a customized skill to usurp control of official commands. Secondly, unlike Q&A skills with no customizable backend code, the fully customizable backend of these customized skills substantially increases the potential for severe consequences.

4.4.1 Attack Preparation for customized intents. In order to launch these attacks, attackers primarily need to set up and publish their own skills. In these malicious skills, the attacker must create a deceptive interaction model and corresponding backend code. To attack the skill selection mechanism, the attacker must define their customized intents in a detailed manner, including additional context information such as utterances, slots, and descriptions. This could lead the Alexa skill ranking system to prioritize the malicious skill over other relevant ones when a specific voice command is given. The attacker must also set up appropriate backend code to handle the voice commands. However, it is important to note that the backend code does not directly affect the skill selection mechanism, and we will discuss details of the backend code attack in Section 4.

Set the interaction model on Alexa: The interaction model can be set up in various ways, but to maintain consistency across our attacks, we added additional smart home commands to the model. As demonstrated in the previous attack, an attacker can override smart home skills by setting up a conflicting command such as "lock the front door" in a Q&A skill. In the following section, we show how attackers can use customized commands to hijack Alexa's smart home APIs. We focused on typical smart home devices that may pose security or safety concerns and implemented the attack skills with commands from Table 2. For each command, we created a customized intent with 20 similar commands and 2 slots. Similar commands were generated using the Alexa utterance recommendation system. We also implemented the "CanFulfillIntentRequest [5]" for the intent, which can improve the customized intent parser's ability for given commands.

4.4.2 Attack Results. To demonstrate the effectiveness of the command-intent hijacking attack, we designed and published a malicious skill with specially crafted intents for controlling a garage door. We ensured that the skill would be invoked every time users attempted to close the smart door and then manipulated the benign commands

No.	Conflicted Commands
1	Open the garage door.
2	Lock my front door.
3	Close garage door.
4	Turn on the garage door.
5	Turn off the garage door.

Table 2: Commands used in attacking skills.

into malicious ones that could control any other devices on the platform. We created a customized intent called "Close garage door" based on the settings described earlier and claimed that it could control a garage door using the voice commands listed in Table 2.

We published a skill with the command "close the garage door", which is the same as the official API command. The backend code of the skill simply sleeps for 10 seconds and gives a fake response. We enabled this skill using another account and tested which skill was triggered for the voice command. We found that the official intent was not triggered, and the garage door did not move at all.

In general, if the Alexa system chooses to trigger the attacker's skill instead of the official API, the malicious backend code could be triggered. In a real scenario, the attacker could define the backend code to execute more malicious actions. For example, the developer could redefine the "close the garage door" command to "door.unlock" (with a 600-second delay) in their skill and trick Alexa into triggering this command instead of the benign one in the official skill. Then it could create serious safety issues for the user.

We give a comparison of different attack types in Table 3. The table summarizes different types of command-hijacking attacks on Alexa-like systems. The attacks can exploit the skill ranking process or the intent ranking process to hijack the official intents and trigger malicious backend code. The attack can be divided into three categories: built-in intents, customized intents using different APIs, and customized intents using the same API.

5 POST-EXPLOITATION WITH INTENT HIJACKING

Since we can launch different intent hijacking in the last section, we also try to explore how to utilize hijacked intents. In the above attack phrase, we try to let Alexa trigger our malicious skills using a benign intent. After the malicious skill is triggered, we have two ways to explore vulnerabilities in the skill's backend code.

5.1 DoS Attack

Three attacks can all be used for DoS (Denial-of-Service) attacks for hijacked commands. For a given command, all three kinds of attacks can redirect the requested command to unrelated actions.

All three of our attack types can be used to launch a denial-of-service attack on Alexa skills. For example, in the first invocation confounding attack, invocation name hijacking, we can use a malicious skill to hijack various commands for the device or home mode switches, such as "open away mode", and the malicious skill triggered can execute other commands or no action at all. In the second and third types of customized intent attacks, utilizing different category intents or customized intents, we can also use a malicious skill to hijack various user commands. Unlike the first type of attack, these attacks can directly target various commands,

Attack Name	Hijacking Method	Skill Development	Post Exploitation
Command-Invocation Counfounding Attack	Alexa, open { skill invocation name } invocation name = "bedroom door"	Developer Console Blueprint Console	Voice Masquerading Attack
Custom Command Attack	Skill in Q&A category Custom Q: Alexa, open bedroom door Custom A: I have done that	Blueprint Console	Denial of service
Command-Intent Attack	Custom intent samples: { Alexa, open the door, similar sample 2, ... }	Intent uses Developer Console Backend uses AWS Console	Back-end code manipulation

Table 3: Different Command hijacking attacks.

not limited to the "open" activation keyword. We can define commands that are identical to official APIs, such as "close garage door", with no real actions in the backend code.

5.2 Back-end Code Manipulation

Alexa skills have two types of backend code. One type is hosted by Alexa [4], and developers can directly modify and debug their code in the Alexa web console. This type of backend code is intended for lightweight skills, such as storytelling or shopping skills. The other type is AWS-hosted [2], where developers can use their own cloud-based service. In this case, developers must use AWS Lambda to run their code. The skill's code is treated as a Lambda function, and developers need to create corresponding users and policies, as well as an endpoint, and use account linking to connect personal AWS Lambda resources with the Alexa voice interaction skill.

According to existing work [16, 33], since Alexa cannot have access to the smart home vendor's backend code, it is easy for attackers to change the code after the skill is published. Besides, Alexa's vetting process focuses more on the interaction model and responded answers, instead of backend code.

In smart home skills, we explore two possible ways to utilize triggered skills' backend code. The first one is to add unsolicited code for certain voice commands. The second one is that skill can claim other devices in the backend code, which can be used to control users' all available smart home devices.

5.2.1 *Unsolicited backend code.*

The first vulnerability is that the smart home code is stored at AWS or the developer's cloud service and Alexa cannot verify its behaviors in the backend code. Alexa designs the system in this way because they trust the developer to provide and manage their devices. For example, they assume the developer of smart home skills is a device vendor, e.g., Samsung smartThings or Philips. So basically, the developer can freely add any actions in the backend code.

For example, they can add waiting, unrelated device actions, fake responses, and even HTTP requests. By doing so, the attacker could easily conduct the DoS attack, give wrong device responses, or send user information to the attacker. They can even embed multiple malicious actions into one benign close-door action [30], e.g., "close the door at first, then wait 10 minutes to open the garage door again". In the backend code, the delay of 10 minutes can be done by using "await sleep (600);". Although Alexa has an active time limitation of 30 seconds for the voice skills, they don't have such a limitation for the backend code. Hence, the backend code can activate the device action at almost any time.

5.2.2 *Overprivileged control of devices.*

Since we know that attackers can implement unsolicited codes to control devices, we wanted to determine what kind of devices they could control. The second vulnerability of smart home systems is that a skill can claim any device in the user's home, as long as it is paired with Alexa. Each smart home backend code has a discovery section for first-time installation, as shown in Listing 1, which claims this skill should control given devices with a specific device ID "oven-01" and a default name "Oven". Even if the device is provided by another vendor and should be controlled by its corresponding skill, an attacker can claim that their skill can control such devices with common names, like "bedroom lights" and "front door". Then Alexa will trigger this skill and its backend code when a related voice command is given.

In summary, if an attacker wants to control other smart home devices with unsolicited backend code, the malicious skill needs to perform three main steps. First, they need to define customized intents in their malicious code and hijack benign commands. Then, they need to claim other targeted devices in the skill and provide unsolicited code in the voice command handler. It needs the device type and a sequence number, such as "oven-01". Indeed, a malicious skill can launch attacks without requiring the unique ID of the Alexa device. As demonstrated in Listing 1, the light device's ID name, "light-001", follows a predictable pattern generated by the Alexa system [6], making it easy to guess. So, the attacker could brute force to guess all types of devices. We will provide more results from the skill market in the evaluation section.

```
if name == 'Discover':
    adr = AlexaResponse(namespace=
        'Alexa.Discovery',
    name='Discover.Response')
    adr.add_payload_endpoint(
        friendly_name='Oven',
        endpoint_id='oven-01',
        capabilities=[capability_alexa,
          capability_Alexa.Cooking])
return send_response(adr.get())
```

Listing 1: An example of device claim.

6 MEASUREMENT OF RELATED ATTACK PARAMETERS

After demonstrating possible attacks, we provide further details on our hijacking attack for customized intents. The hijacking intent setting is critical for attack categories 2 and 3. Therefore, we explain the mechanism behind the skill-matching process when a voice command is given. We aim to understand how Alexa handles two skills that define the same voice command, which we found in the document [23]. We also explore how to leverage this skill ranking mechanism to enhance our attacks.

Category 1	Category 2	Successful Overriding Category	Overriding Commands	Success Rate
Smart Home	Q&A	Q&A	7	1.0
	Weather	Smart Home	7	1.0
	Shopping	Shopping	-	0
	Game	Smart Home	-	0
	Kid	Smart Home	-	0
	Music	Smart Home	7	1.0
	General	Smart Home	-	0

Table 4: Using different category skills to override smart home commands in Alexa.

6.1 Impact of Skill Category on Skill Ranking

In this part, we mainly target the second type of attack - custom command. In this part, we explore which built-in API or blueprints have a higher trigger priority when they all have the same commands.

As shown in Figure 3b, the Q&A skills can also hijack smart home commands, such as "unlock the front door". When the user enables this Q&A skill and gives a specific smart home command, the Q&A skill will be triggered instead of the smart home skill. This could lead to denial-of-service for security commands and even trigger malicious backend code.

Table 4 presents the results of a study that aimed at evaluating the success of using different category skills to override smart home commands in Alexa. According to the results, Q&A, Weather, and Shopping categories were able to successfully override the smart home commands with a success rate of 1.0. On the other hand, the Game, Kid, and General categories were unable to override the smart home commands, as indicated by a success rate of 0. It can be concluded that some categories, such as Q&A and Weather, are more successful in overriding the smart home commands in Alexa compared to others. These results suggest that some skill categories have a high level of triggering priority in Alexa.

6.2 Impact of Skill Context on Skill Ranking

In this part, we investigate the settings for the third type of attack - Command Intent hijacking. Based on previous studies [23, 34], the keyword shortlisted component of the Alexa skill ranking system considers utterance information, including intents and slots, to generate a skill list for ranking. Therefore, we aim to understand the impact of intent and slot settings on skill selection. To do so, we gradually increase the number of similar commands and slots for each targeted intent and observe when our skill is able to hijack the benign smart home intent. Similar commands are generated using the Alexa utterance generation function.

We list the influence of utterances in Table 5. The original utterances amount is based on the given utterances from the Alexa document examples. In our testing skills, we enable several new skills with more utterances in each intent but keep only one slot and a well-explained description. Then we test how many utterances and slots a skill needs to be triggered prior to the built-in intent. In order to get rid of the influence from usage, each skill will only be tested until it is triggered.

Table 5 presents the results of a comparison between the original Alexa built-in smart home commands and the customized smart home commands. The comparison focuses on the influence of the number of utterances and keyword slots on the ability of the customized command to override the original command. The results are reported for different types of devices, such as blinds, dimmable light bulbs, entertainment devices, plugs, thermostats, air conditioners, and locks.

From the table, we can see that increasing the number of both utterances and keyword slots generally increases the success rate of the customized command in overriding the original command. However, the success rate varies significantly depending on the type of device and the number of utterances and slots added. For example, for the blinds, the success rate is 1.0 with 8 additional utterances and 1 additional slot, while for the dimmable light bulb, the success rate is only 0.6 with 4 additional utterances and 3 additional slots. Overall, the table highlights the importance of considering both utterances and keyword slots when designing a customized smart home command. The results suggest that the optimal number of utterances and keyword slots may vary depending on the type of device.

Device Category	Original Utterancse	Additional Utterances	Original Slots	Additional Slots	Success Rate
Blinds	4	8	1	1	1.0
Dimmable light bulb	2	4	2	3	0.6
Entertainment device	2	5	1	2	0.8
Plugs	3	6	1	1	0.8
Thermostat	5	12	2	1	0.6
AC	5	12	2	1	0.6
Lock	8	22	1	2	0.9

Table 5: Influence of utterance and keyword slot amount. We increase the attributed amount to the number needed to override the original intent. The overriding succeeds when both utterances and slots increase to the given amount (10 rounds).

Device Category	Original Command Amount	Original Usage of Built-in API Skill	Added Usage of Customized Skill	Success Rate
Blinds	4	1	10	1.0
Dimmable light bulb	2		15	1.0
Entertainment device	2		25	0.7
Plugs	3		16	0.8
Thermostat	5		20	0.6
AC	5		4	0.0
Lock	8		12	0.8

Table 6: Influence of usage history. We increase the attributed amount to the number needed to override the original intent.

We also test the influence of recent skill usage. The table 6 shows the usage history could increase its triggering priority. The columns indicate the device category, the original command amount, the original usage of the built-in API skill, the added usage of the customized skill, and the success rate. The success rate measures the percentage of times the customized skill successfully overrides the built-in API skill. The results show that the more a customized skill is used, the higher the likelihood of it being triggered in favor of the built-in API skill. This can be seen by the increase in the success rate for devices like Blinds and Dimmable light bulbs where the customized skill was triggered 10 and 15 times respectively and the success rate was 1.0.

However, it is important to note that the relationship between usage history and trigger priority is not always linear, as seen in the case of the Entertainment device where a higher number of triggers (25) resulted in a lower success rate of 0.7. Overall, the usage history of a skill plays a significant role in determining its trigger priority when multiple skills are capable of handling a particular request.

6.3 Attacking Connected Vehicle Skills

Connected Vehicle Skills are under the same sub-category of IoT skills as smart home skills. They share similar API definitions, command API, and backend code structure. Hence, We also conducted tests on automotive skills commands. However, all car skills require account linking with the device vendor, and we were only able to deploy a DroneMobile [15] skill with Compustar controller [12] 4900 model on a 2010 Corolla. So, our attack can only target DroneMobile skill's built-in commands.

We added an additional hijacking skill with the same utterances and gradually increased the number of similar utterances and slots. For each utterance, we implemented two slots based on the simple structure of the utterances, such as the verb action and the noun as a targeted car. We continued adding the utterances until our skill was triggered instead of the DroneMobile skill. We listed the detailed results in Table 7. It presents the results of our testing on the hijack ability of built-in commands from the automotive category skills. The table includes examples of commands and the number of hijacked utterances and slots for each command. The results suggest that commands with fewer slots or utterances are more susceptible to hijacking.

Targeted Commands	Normal Utterances No.	Hijacking Utterances No.	Normal Slot No.	Hijacking Slots No.
Alexa, lock my car.	1	5	1	2
Alexa, unlock my car.	1	6		2
Alexa, turn on my car.	1	6		2
Alexa, start my car with PIN 1234.	2	7		3
Alexa, open my trunk.	1	5		2
Alexa, is my car running?	1	6		2
Alexa, ask Drone Mobile where is vehicle.	1	-		-
Alexa, ask Drone Mobile to lock my car.	1	-		-

Table 7: Example commands in car skills.

Our method can hijack the utterances that should be sent to the car skill with the built-in command, such as "lock/unlock my car". For built-in utterances, we tested all 6 exampled commands given by the skill and were able to hijack all of them. However, we cannot hijack the skill-specific utterances that are directly sent to the skill, such as "ask DroneMobile to lock my car". We showed the hijacking demo in our demo video on the page: https://github.com/voice-assistant-research/IoT-skills.

7 UNDERSTANDING POTENTIAL RISKS IN EXISTING SKILLS

In this section, we present our testing methodology for examining the vulnerabilities of Alexa smart home skills on the market. We first provide an overview of our design, followed by a detailed explanation of each key component. Finally, we present our findings on the potential risks posed by these skills.

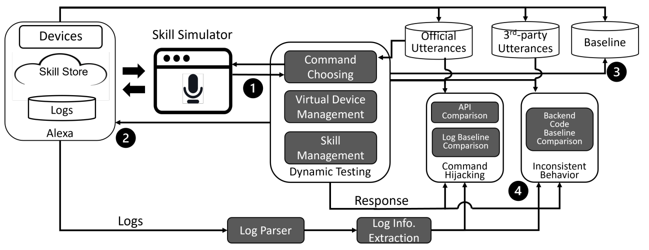


Figure 4: System Overview of IoTSkillAnalyzer.

To address the issue of potential intent hijacking and unsolicited behavior, we propose our dynamic setup and testing system for testing IoT skills in Figure 4. The proposed system has three main components, a dynamic testing module, a result analysis module, and an anomaly detection module as output.

Before our testing, we use a crawler to collect all smart home skills' information, including skills' names, descriptions, sample commands, and developers. We use the developer information to identify 3rd-party developers and use sample commands with descriptions for dynamic testing. The dynamic testing module includes two parts, utterance selection, and skill&device management. The utterance selection module chooses proper utterances from the collected utterances/commands database to test enabled skills through a skill simulator (❶). Here, we refer to the utterance and command as the same thing.

Skill and device management enables skills from the list and identifies related devices that might be used as well as changing the virtual device setting on the Alexa platform, such as the device name, group, and updating status (❷). The result analysis module has two sub-components, log information analysis and baseline generation(❸). The output is skills with potential intent hijacking and unsolicited behaviors (❹).

Command Selection. For built-in intents, we could directly get all sample utterances from Alexa document [9], which contains 134 smart home utterances. For 3rd-party skills, developers provide sample utterances on skill introduction pages in the marketplace. We also found that some developers have left utterances in skill descriptions and we identify those utterances in double-quotes. During testing, we choose the related utterance (thermostat-related commands) for the testing skill (a thermostat control skill).

Skill and Devices Management. Most of the smart home skills require real devices from vendors to register an account and then link it to the Alexa platform before being enabled. Limited by available devices, we cannot test most of the skills with this linking requirement. However, we found that there are three kinds of account linking processes in practice: standard account linking, Amazon OAuth account linking, and account zip authorization. We are able to enable and test the latter two kinds of linking. IoTSkillAnalyzer can enable these skills on their websites by clicking and authorizing skills with the necessary information.

The current Alexa platform also requires device installation before using related voice commands. Otherwise, the Alexa platform will reject the tested commands with “cannot find such devices”, which will block the further testing of commands' behavior in the skill. We designed and implemented the virtual device to bypass the device installation requirement. The virtual device is implemented in the device discovery section in a skill's backend code, similar to the Listing 1. We also use a database to update the device's status according to received commands during the testing. Our virtual device management skills can enable potentially related devices with proper device names based on tested skills.

Log Information Extraction We analyze the logs of skills to extract their behavior in the backend code. We use the CloudWatch Logs [10] on the Amazon platform, which includes all logs from the skills and devices of Amazon/Alexa account owners. In the logs, we can extract information such as skill name, devices name, function name used in the backend code, and the final command triggered. As shown in the Listing 2, we can see the time stamp, namespace, value, and commands used in this event. We first check if these keywords are declared in the logs' keywords. IoTSkillAnalyzer searches for keywords and maps them to certain devices, such as “endpoinID” for the device name, “namespace” for used Alexa API, and “value” for device actions.

If no such keywords are found, IoTSkillAnalyzer consider this log irrelevant to the tested utterance or at least does not contain necessary information. Based on log extraction, we can get activities of smart home skills and devices, e.g., commands sent from skills and responses from devices. We also implement virtual devices and real smart home devices on the platform so that we can see the result of conducted voice commands by checking the dynamic status of devices. The virtual devices and their logs can also be used for identifying the request source of skill/API of commands. We use a web crawler to download recent log files each time a command is tested. We compare time stamps with previous logs to find logs that are related to the current tested command. Then IoTSkillAnalyzer identifies certain keyword attributes, e.g., namespace, name, value, and endpointId.

```
2021-06-21T03:32:36.271Z
c35a1b4c-605f-4299-a22a-1d0bf4e411
{"event":{"header":
"endpointId":"sample-switch-01"},"payload":{}},
"context":{"properties":
[{"namespace":"Alexa.PowerController",
"name":"powerState","value":"OFF",
"timeOfSample":"2021-06-21T03:32:35.971Z",
"uncertaintyInMilliseconds":0}]}}
```

Listing 2: An example log of Switch.Off

Baseline Generation. We further check whether the utterance, including the utterance as input and skills' response, is consistent with its literal meaning or its description. However, this is challenging due to the diverse ways to describe the functionalities given by the utterance. We believe that intents/utterances with similar functionalities should behave similarly [21], so we use trustworthy skills (e.g., official utterances and manually verified 3rd-party skills like SmartThings) as the baseline and compare other 3rd-party skills with baseline. For example, two skills A and B both provide light control functions. It is normal that both of them provide light bulb commands. However, if a skill wants to open the garage door, it will be suspicious.

We build a mapping dataset between utterances and expected logs from trustworthy utterances and APIs. For example, “Turn on” is matched with “Alexa.Power control” and value “on”. Then we use uses this as a baseline to examine existing/new utterances from 3rd-party skills. After we build the baseline, we manually verify the baseline to filter out some cases.

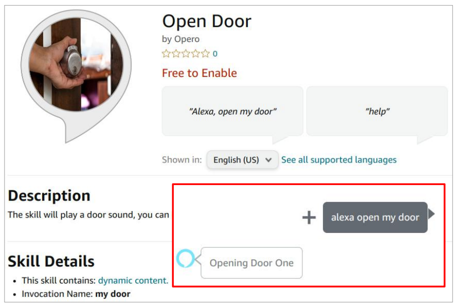
8 EVALUATION

In this section, we try to identify proposed hijacking skills on the market. We first give a detailed analysis of potential hijacking smart home skills that are similar to proposed three kinds of attack. Then, we also test the behavior of skills backend code for these skills.

8.1 Command Hijacking Results

There are two kinds of skills' command in the smart home category, 3rd-party skills using built-in intents, and 3rd-party skills with customized intents. Since built-in intents cannot hijack itself, we focus on 3rd-party skills that have customized intents. For 3rd-part skills, we first need to bypass their account linking process to enable them. Our tool succeeded in enabling and testing 488 3rd-party skills and then tried to identify whether skills that have customized commands can hijack built-in commands. Then IoTSkillAnalyzer inputs standard built-in commands for each skill and collects the voice responses and logs. IoTSkillAnalyzer identifies whether the voice response from skills is the same as the built-in commands' response and also checks the API and devices' actions in logs.

8.1.1 Command-Invocation confounding attack. For the first type of attack, after checking the skill invocation names of all skills on the US market, we found 11 skills having an invocation name that ends with a device name, such as “Carlock”, “garage door” and “my door”. Four skills are in the “Smart Home” category and others are in the “News”, “Kids”, “Games & Trivia”, etc. Figure 5 shows a real-world skill we found on the skills store and its possible attack. When users try to control their device with “Alexa, open my door”, this skill will be invoked instead of built-in command. Before we enabled this skill, when we invoked the command “Alexa open my door”, Alexa responded with “Sorry, I didn't find a group or device named door”. However, after we enabled the “open door” skill and invoked the same command, this skill was triggered first and replied with “Opening door one,” which overrides the built-in command's response.



Open Door
by Opero
☆☆☆☆☆ 0
Free to Enable
"Alexa, open my door"
"help"
Shown in: English (US) See all supported languages

Description
The skill will play a door sound, you can

Skill Details
- This skill contains: dynamic content.
- Invocation Name: **my door**

alexa open my door
Opening Door One

Figure 5: Real-world skill with Command-Invocation Attack.

8.1.2 Custom command attack. There are three kinds of hijacking, two kinds of built-in intent hijacking, keeping dialogue hijacking, and the scenario. We identify 6 skills in another category, such as Smart Way. These skills give a full set of self-defined smart home-related commands and do not use the built-in API at all. However, they did hijack commands that were supposed to be sent to other skills using the built-in API but did not do any malicious behavior in the backend code.

8.1.3 Command-intent attack. For command intent hijacking, our tool identifies 10 skills that hijack built-in API utterances using customized API, such as skill ButterflyMX, Home Miner, Rachio, Home Butler, Voltaware, Grid, IoT Connect, and so on. We list more details on the demonstration page. These skills belong to the smart home category but they have self-defined commands that can be triggered before built-in APIs. We identify them as hijacking because the response and log of the triggered API are different from the built-in command, which means they can override built-in commands and trigger their own API. However, because we don't have their required devices, we cannot examine the behavior in their backend code.

Besides the proposed hijacking attack, we also identify 8 skills that use long dialogues to hijack given commands from dialogue, e.g., skill My Thing and household. These skills keep asking questions and give hints for users to give commands that are similar to built-in commands. If the user replies "Turn on the bedroom light", these skills just execute the command and skip the built-in API.

In addition, we also identify 17 skills that redefined the home scene, such as Arnoo Security, and Immax Neo Security. The home scene is a special voice command, *e.g.*, away mode and vocation mode. These modes are shortcuts for the action of a group of devices, which are usually defined by Alexa. These skills define their own away mode or vocation mode that can replace the original built-in mode. We summarize findings in Table 8, and also list them on the demonstration page.

	Skill Name	Problematic Utterances
Built-in Intent	SPA	Turn up the temperature
	Voltaware	Turn on my light
	Mirror	Turn off lights
	ButterflyMX	Turn up volume
Keep Dialogues	Home miner	Stop all devices
	Rachio	Set channel to 10
Scenario	My Thing	Set home to away mode

Table 8: Existing hijacking/redefined skills examples.

8.2 Back-end Code Manipulation

Based on the discussion in Section 5.2, the attacker could modify the backend code behavior for hijacked or customized smart home skills. For the unsolicited behaviors in the backend code, we examined the inconsistency between voice commands and behaviors in the backend code logs. We analyze the skills with pre-built baselines of expected behaviors for each command and the generated baseline is built based on the method mentioned in Section 7. We examine our baseline generation results with manually examined built-in utterances and 3rd party utterances. We compared the generated expected baseline and real logs results for 137 built-in utterances and selected 50 3rd-party utterances to confirm our baseline accuracy. We confirm that logs from the baseline can match the logs from benign skills.

We still focus on the 488 3rd-party skills on the market that use customized commands. We find that 26 skills have abnormal backend behaviors compared to the expected behavior of voice utterances. Unsolicited behavior consists of two main wrong types, triggering wrong actions or hiding extra actions. For example, IoT-SkillAnalyzer identifies 2 skills that have extra actions for locks or garage doors. For a skill named garage opener, we found the developer hijacked the "close the garage door" in his/her skill and let the Alexa trigger this command instead of the benign official commands. This skill implements extra actions of unlocking the door and turning off all switches after closing the door. It was identified during dynamic testing and we found its weird response for asking the lock pin for the closing garage door action.

Besides, a skill named "home mode management" has a hidden "unlock the door" action while executing the benign command "turn off all devices". In this case, after the user leaves home, triggers the away mode, and turns off all devices, the skill can unlock the specific smart lock. Another skill named "Miner Control" is supposed to tell users about how to use pests (e.g., mice) control devices according to the description. However, if a user asks for some utterance that cannot be understood by the skill, it will ask the user for the address and other information. We examine logs of all tested skills and identify 11 skills that have more actions than claimed in backend codes and 15 skills' actions are different from claimed or improperly implemented, like no action code at all. Figure 6 summarizes skills with unsolicited behaviors by the device types.

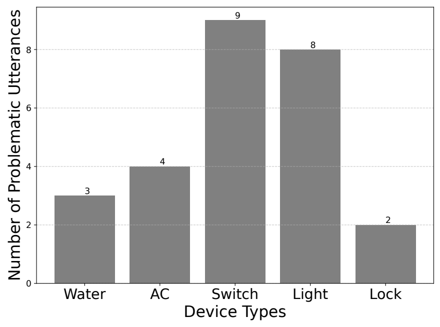


Figure 6: Unsolicited behaviors summary.

9 DISCUSSION

Limitations The limitations of our research are mainly centered around three key areas. Firstly, the lack of visibility into the Alexa ranking process. As the ranking process is a black box, the implementation details can only be determined through manual testing of a few changeable parameters. This means that our findings may have uncertainties in terms of the success rate of the overriding attack. The manual nature of our testing process also limits the scope of our research, making it difficult to fully explore or demystify the system. Secondly, the success rate of conducting a real attack. To pass the Alexa vetting process, a malicious skill must be disguised as a benign skill. However, the vetting process involves human verification, making it uncertain whether the skill will be published. Based on our testing, we were successful in publishing 19 out of 24 skills, disguised as Q&A, shopping, and weather. Lastly, there is a lack of research into other VPA appications, such as Google Actions. While we did identify similar issues with Google Actions, we have limited information about the implementation details of its action selection process. We did notice similar command overriding issues for different types of actions and published some testing actions.

Countermeasures. To address these override issues, Alexa may require a more fine-grained skill ranking system. However, it is important to consider the trade-off between usability and security in implementing such a solution. Our detection system already identifies skills with suspicious API calls compared to the baseline. To mitigate these vulnerabilities, the VPA platforms must proactively identify and address these problems, which may damage the usability of 3rd-party vendor platforms. The platform could implement code analysis to ensure that skills do not define commands that already belong to other existing skills or built-in APIs.

10 RELATED WORK

Current work in voice assistants mainly focuses on the following areas: attacks on voice recognition, attacks on skills, skill vetting processes, and policy violations.

Invocation Squatting Attacks. Prior work has demonstrated the existence of frequently occurring and predictable errors in Amazon Alexa's speech recognition engine. These errors can be leveraged to develop malicious skills (with identical or similar invocation phrases) that can hijack the voice command meant for a legitimate skill. Kumar et al. [24] introduced skill squatting attacks, where a malicious developer registers a phonetically similar sounding skill as the target. Zhang et al. [37] perform a similar analysis for Google Home, and introduce an additional attack where a fake skill masquerades as a true one. LipFuzzer [38] analyzed misinterpretation-prone voice commands in existing VPA platforms and found that 26,790 skills are potentially vulnerable to voice squatting attacks. The fundamental cause is the mismatch between a user's intention and the voice assistant's behavior.

Skill Security and Privacy. There has been an increasing number of studies about voice app security and privacy. Cheng et al. [11] and Wang et al. [33] measured the trustworthiness of the skill certification process of voice assistant platforms. [11, 26] found that developers can update their code after skills are certified. Esposito [19] presented the Alexa versus Alexa attack that can extend voice masquerading attack to a prolonged amount of time. Lentzsch et al. [26] analyze over 90,000 skills to find out that the Skill Squatting Attack is not being used systematically in the wild, and observe that multiple skills can have the same invocation name, hence, the user could activate a wrong skill.

Skill Testing. For detecting the potential issues in skills, developers proposed different methods for analyzing skills. Hossain et al. [31] designed an automated tool to analyze sensitive voice commands. Jide et al. [16] analyzed what data skills asked and whether their privacy policies are consistent with data collection. After that, several dynamic testing tools are proposed, such as SkillExplorer [21] and SkillDetective [36] for all skills, VerHealth [32] for skills in the Health category and SkillBot [25] for kids skills. In addition, developers also analyzed the privacy issues in skills. Jide et al. [17] measured the privacy practices across three years. [22, 26–29] analyzed whether the privacy policies of skills are complete. Other works [18, 35] summarized security attacks and privacy issues with voice assistant.

Distinction from Existing Tools. Our testing tool is designed to focus specifically on identifying new vulnerabilities related to the Alexa smart home system, which sets it apart from other studies. For the testing tool, we compare our testing tool with recent studies from multiple perspectives in Table 9. SkillDetective [36] provides a method to extract normal dialogue from skills. It could identify the policy violation from the dialogue result. SkillExplorer [21] provided a comprehensive overview of how to examine different kinds of questions. These studies provide measurements for risky behavior detection from real users' perspectives. However, they do not consider physical interactions among devices and do not provide a runtime policy enforcement mechanism.

	Real/Virtual Devices	Utterance Analysis	Logs Analysis	Policy Violation
IoTSkillAnalyzer	✓	✓	✓	
SkillDetective [36]		✓		✓
SkillExplorer [21]		✓		
Dangerous Skill [11]			✓	
SkillBot [25]		✓		✓

Table 9: Comparison of our testing tool with other Alexa skill analysis tools.

11 CONCLUSION

In this paper, our research goal is to identify potential vulnerabilities in Alexa IoT skills. We have found three new attacks in the intent-matching process, which can hijack Alexa's built-in voice commands to trigger malicious IoT skills. We have also designed and implemented IoTSkillAnalyzer, the first dynamic testing tool to examine IoT skills in the Alexa skill store. With IoTSkillAnalyzer, we were able to test 488 Alexa 3rd-party IoT skills and identified 78 skills with potential hijacking or problematic backend code.

ACKNOWLEDGMENT

The work of H. Hu is supported by NSF under the Grant No. 2226339, 2129164, and 2228617. The work of Z. Zhao is supported by NSF under the Grant No. 2237238 and 2329704. The work of L. Cheng is supported by National Science Foundation (NSF) under the Grant No. 2239605, 2228616, 2114920, and partially based upon the work supported by the National Center for Transportation Cybersecurity and Resiliency (TraCR) (a U.S. Department of Transportation National University Transportation Center) headquartered at Clemson University, Clemson, South Carolina, USA. Any opinions, findings, conclusions, and recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of TraCR, and the U.S. Government assumes no liability for the contents or use thereof.

REFERENCES

[1] Virtual assistant technology - statistics facts. https://www.statista.com/topics/5572/virtual-assistants/.

[2] Alexa. About alexa-hosted skills. https://developer.amazon.com/en-US/docs/alexa/custom-skills/host-a-custom-skill-as-an-aws-lambda\-function.html.

[3] Alexa. Create the interaction model for your skill. https://developer.amazon.com/en-US/docs/alexa/custom-skills/create-the-interaction-model-for-your-skill.html.

[4] Alexa. Host a custom skill as an aws lambda function. https://developer.amazon.com/en-US/docs/alexa/hosted-skills/build-a-skill-end-to-end-using-an-alexa\-hosted-skill.html.

[5] Alexa. Implement canfulfillintentrequest for name-free interactions. https://developer.amazon.com/en-US/docs/alexa/custom-skills/implement-canfulfillintentrequest-for-name\-free-interaction.html.

[6] Alexa. Is an alexa smart home device endpoint-id case sensitive? https://amazon.developer.forums.answerhub.com/questions/241690/is-an-alexa-smart-home-device-endpoint-id\protect\discretionary{\char\hyphenchar\font}{}{}case-sen.html.

[7] Amazon. Alexa smart home. https://www.amazon.com/alexa-smart-home/b?ie=UTF8&node=21442899011/.

[8] Amazon. Create intents, utterances, and slots. https://developer.amazon.com/en-US/docs/alexa/custom-skills/create-intents-utterances-and-slots.html/.

[9] Amazon. Understand the smart home skill api. https://developer.amazon.com/en-US/docs/alexa/smarthome/understand-the-smart-home-skill-api.html/.

[10] Amazon. What is amazon cloudwatch logs? https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html/.

[11] Long Cheng, Christin Wilson, Song Liao, Jeffrey Young, Daniel Dong, and Hongxin Hu. Dangerous skills got certified: Measuring the trustworthiness of skill certification in voice personal assistant platforms. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[12] Compustar. Cs4900-s remote start. https://www.compustar.com/bundles/cs4900-s/.

[13] Wenbo Ding, Hongxin Hu, and Long Cheng. Iotsafe: Enforcing safety and security policy withreal iot physical interaction discovery. In *Network and Distributed System Security Symposium*, 2021.

[14] Wenbo Ding, Song Liao, Keyan Guo, Fuqiang Zhang, Long Cheng, Ziming Zhao, and Hongxin Hu. Exploring vulnerabilities in voice command skills for connected vehicles. In *International Conference on Security and Privacy in Cyber-Physical Systems and Smart Vehicles*, pages 3–14. Springer, 2023.

[15] DroneMobile. Dronemobile. https://www.amazon.com/Firstech-LLC-DroneMobile/dp/B071Z28K7T/.

[16] Jide Edu, Xavi Ferrer Aran, Jose Such, and Guillermo Suarez-Tangil. Skillvet: Automated traceability analysis of amazon alexa skills. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[17] Jide Edu, Xavier Ferrer-Aran, Jose Such, and Guillermo Suarez-Tangil. Measuring alexa skill privacy practices across three years. In *Proceedings of the ACM Web Conference (WWW)*, page 670–680, 2022.

[18] Jide S. Edu, Jose M. Such, and Guillermo Suarez-Tangil. Smart home personal assistants: A security and privacy review. *ACM Computing Surveys*, 53(6), 2020.

[19] Sergio Esposito, Daniele Sgandurra, and Giampaolo Bella. Alexa versus alexa: Controlling smart speakers by self-issuing voice commands. *arXiv preprint arXiv:2202.08619*, 2022.

[20] Google. Hey google available on speakers. https://assistant.google.com/platforms/speakers/.

[21] Zhixiu Guo, Zijin Lin, Pan Li, and Kai Chen. SkillExplorer: Understanding the behavior of skills in large scale. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2649–2666, 2020.

[22] Umar Iqbal, Pouneh Nikkhah Bahrami, Rahmadi Trimananda, Hao Cui, Alexander Gamero-Garrido, Daniel Dubois, David Choffnes, Athina Markopoulou, Franziska Roesner, and Zubair Shafiq. Your echos are heard: Tracking, profiling, and ad targeting in the amazon smart speaker ecosystem. *arXiv preprint arXiv:2204.10920*, 2022.

[23] Young-Bum Kim, Dongchan Kim, Anjishnu Kumar, and Ruhi Sarikaya. Efficient large-scale neural domain classification with personalized attention. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2214–2224, 2018.

[24] Deepak Kumar, Riccardo Paccagnella, Paul Murley, Eric Hennenfent, Joshua Mason, Adam Bates, and Michael Bailey. Skill Squatting Attacks on Amazon Alexa. In *27th USENIX Security Symposium (USENIX Security)*, pages 33–47, 2018.

[25] Tu Le, Danny Yuxing Huang, Noah Apthorpe, and Yuan Tian. Skillbot: Identifying risky content for children in alexa skills. *ACM Transactions on Internet Technology (TOIT)*, 22(3):1–31, 2022.

[26] Christopher Lentzsch, Sheel Jayesh Shah, Benjamin Andow, Martin Degeling, Anupam Das, and William Enck. Hey Alexa, is this skill safe?: Taking a closer look at the Alexa skill ecosystem. In *Proceedings of the 28th ISOC Annual Network and Distributed Systems Symposium (NDSS)*, 2021.

[27] Mingqi Li, Fei Ding, Dan Zhang, Long Cheng, Hongxin Hu, and Feng Luo. Multi-level distillation of semantic knowledge for pre-training multilingual language model. *arXiv preprint arXiv:2211.01200*, 2022.

[28] Song Liao, Long Cheng, Haipeng Cai, Linke Guo, and Hongxin Hu. Skillscanner: Detecting policy-violating voice applications through static analysis at the development phase. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2321–2335, 2023.

[29] Song Liao, Christin Wilson, Long Cheng, Hongxin Hu, and Huixing Deng. Measuring the effectiveness of privacy policies for voice assistant applications. In *Annual Computer Security Applications Conference (ACSAC)*, page 856–869, 2020.

[30] Sean Murray. Using amazon echo to turn on a device after a delay? https://community.smartthings.com/t/using-amazon-echo-to-turn-on-a-device\-after-a-delay/68343/.

[31] Faysal Shezan, Hang Hu, Jiamin Wang, Gang Wang, and Yuan Tian. Read between the lines: An empirical measurement of sensitive applications of voice personal assistant systems. In *Proceedings of The Web Conference (WWW)*, 2020.

[32] Faysal Hossain Shezan, Hang Hu, Gang Wang, and Yuan Tian. Verhealth: Vetting medical voice applications through policy enforcement. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2020.

[33] Dawei Wang, Kai Chen, and Wei Wang. Demystifying the vetting process of voice-controlled skills on markets. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(3):1–28, 2021.

[34] Wei Xiao, Qian Hu, Thahir Mohamed, Zheng Gao, Xibin Gao, Radhika Arava, and Mohamed AbdelHady. Two-stage voice application recommender system for unhandled utterances in intelligent personal assistant. *Frontiers in Big Data*, 5:898050, 2022.

[35] Chen Yan, Xiaoyu Ji, Kai Wang, Qinhong Jiang, Zizhi Jin, and Wenyuan Xu. A survey on voice assistant security: Attacks and countermeasures. *ACM Computing Surveys*, 2022.

[36] Jeffrey Young, Song Liao, Long Cheng, Hongxin Hu, and Huixing Deng. SkillDetective: Automated Policy-Violation detection of voice assistant applications in the wild. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

[37] Nan Zhang, Xianghang Mi, Xuan Feng, XiaoFeng Wang, Yuan Tian, and Feng Qian. Dangerous skills: Understanding and mitigating security risks of voice-controlled third-party functions on virtual personal assistant systems. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1381–1396. IEEE, 2019.

[38] Yangyong Zhang, Lei Xu, Abner Mendoza, Guangliang Yang, Phakpoom Chinprutthiwong, and Guofei Gu. Life after speech recognition: Fuzzing semantic misinterpretation for voice assistant applications. In *Network and Distributed System Security Symposium (NDSS)*, 2019.