MegaMind: A Platform for Security & Privacy Extensions for Voice Assistants

Seyed Mohammadjavad Seyed Talebi[*]
Ardalan Amiri Sani
University of California, Irvine

Stefan Saroiu
Alec Wolman
Microsoft

ABSTRACT

Voice assistants raise serious security and privacy concerns because they use always-on microphones in sensitive locations (e.g., inside a home) and send audio recordings to the cloud for processing. The cloud transcribes these recordings and interprets them as user requests, and sometimes even shares these requests with third-party services. These steps may result in unintended or malicious voice data leaks and in unauthorized actions, such as a purchase. This paper presents MegaMind, a novel extensible platform that lets a user deploy security and privacy extensions locally on their voice assistant. MegaMind's extensions interpose on requests before sending them to the cloud and on responses before delivering them to the user. MegaMind's programming model enables writing powerful extensions with ease, such as one for secure conversations. Additionally, MegaMind protects against malicious extensions by providing two important guarantees, namely permission enforcement and non-interference. We implement MegaMind and integrate it with Amazon Alexa Service SDK. Our evaluation shows that MegaMind achieves a small conversation latency on platforms with adequate compute power, such as a Raspberry Pi 4 and an x86-based laptop.

CCS CONCEPTS

• **Security and privacy** → **Systems security**; **Mobile platform security**; **Privacy protections**; **Security services**; **Domain-specific security and privacy architectures**.

KEYWORDS

voice assistants, smart speakers, extensibility, security and privacy

ACM Reference Format:
Seyed Mohammadjavad Seyed Talebi, Ardalan Amiri Sani, Stefan Saroiu, and Alec Wolman. 2021. MegaMind: A Platform for Security & Privacy Extensions for Voice Assistants. In *The 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '21), June 24–July 2, 2021, Virtual, WI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458864.3467962

[*]This research started when the first author was an intern at Microsoft Research in the summer of 2018.

MobiSys '21, June 24–July 2, 2021, Virtual, WI, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8443-8/21/06...$15.00
https://doi.org/10.1145/3458864.3467962

1 INTRODUCTION

Voice assistants, such as Amazon Alexa [20], Google Assistant [26], Apple Siri [37], and Microsoft Cortana [35], are becoming ingrained in our personal lives. Beyond their prevalent integration into smartphones and tablets, they are now increasingly found in home speakers [21, 28–30, 32], cars [3, 11, 12], children's toys [9], light bulbs [6], TV sets [4, 10], and other appliances. Several companies have even released device SDKs to simplify adding voice assistance to any hardware device [22, 23, 27].

Voice assistants provide a convenient user interface: natural language. However, this convenience comes with serious security and privacy risks. A voice assistant uses an always-on microphone and operates by capturing audio and sending it to the manufacturer's cloud service for processing. The cloud service transcribes the audio and interprets it as user requests. Audio recordings can have private and sensitive content, such as medical or sexual information [57]. Moreover, interpreted requests may result in unintended or unapproved actions, such as a purchase or a phone call. These unintended actions can be either due to "mistakes" by the assistant, or attacks [7, 8]. Moreover, the assistants' responses might contain inappropriate content, such as content not suitable for children [1].

To make matters worse, voice assistants incorporate many third-party applications, i.e., *skills*[1], which enhance the assistant functionality [39]. Unlike mobile apps, skills do not run on the voice assistant hardware. Instead, they are cloud services invoked by the manufacturer's cloud service. Researchers have shown a plethora of additional security and privacy concerns surrounding third-party skills [49, 53, 63, 69], including malicious skills [60] and unintended voice data leaks [5, 54].

This paper presents *MegaMind*, a security and privacy extensibility platform for voice assistants. MegaMind extensions execute locally on the assistant itself. They intercept the recorded audio before sending it to the manufacturer's cloud service, and the response audio before delivering it to the user. Extensions can thus inspect, modify, or discard unwanted content to meet a user's security and privacy goals. For example, a redaction extension removes any mentions of a user's personal information from the recorded audio.

MegaMind's design enables novel extensions that bring a level of unprecedented security to users. For example, we implement *secure conversation*, an extension that provides end-to-end encryption, integrity, and rollback protection to let a user conduct a secure conversation with a trusted skill such as a bank, without the voice assistant manufacturer having unrestricted access to the conversation. As another example, we implement *anonymous query*, an extension that employs a mixer cloud service to enable a user to

[1]*Skill* is Amazon's Alexa service terminology for voice assistant apps, but we use it broadly for all assistants.

remain anonymous (to the voice assistant manufacturer and to third-party skills) when issuing sensitive queries such as medical queries.

MegaMind does not blindly trust extensions and assumes they can be malicious. To protect users from such extensions, MegaMind provides two security guarantees, *permission enforcement*, and *non-interference*. Permission enforcement limits the conversations each extension can access (i.e., *access permissions*) and modifications it can perform on them (i.e., *modification permissions*). Moreover, *non-interference* guarantees that a malicious extension cannot modify the conversation in a way that disrupts the execution of other extensions.

Given the richness of natural language, providing such guarantees is challenging and requires a careful design and a comprehensive security analysis. To do so, MegaMind provides a novel programming model for extensions. Each extension consists of a *manifest* and an *action function*. In the *manifest*, an extension declares its needed permissions using MegaMind's easy-to-review and straightforward permission description language. *Action functions* are generic Python scripts that process the phrases. We enforce several limitations on what an extension can do, and we demonstrate, with a careful analysis, that our design provides the aforementioned security guarantees.

Any third-party can develop MegaMind extensions. In addition, MegaMind can provide an extension market (similar to the application market for smartphones) in which developers publish their extensions. Similar to application markets, a MegaMind extension market can audit all extensions prior to publication, and reject those with malicious manifests. In addition, the extension market can authenticate the extension developers. For example, if a companion extension for a third-party skill implements secure conversation, then MegaMind can easily check that this skill and its companion extension are published by the same entity.

We build MegaMind and integrate it with the Amazon Alexa Service SDK. Thus, it is potentially deployable on many commercial devices that use the Alexa SDK, such as the Acer Spin 5 Convertible Notebook [18] and the Fitbit Versa 2 smart-watch [41]. MegaMind is also compatible with all Alexa skills. To minimize conversation latency, we optimize MegaMind's implementation in several ways, such as using a sandbox pool to reduce startup latency. Our prototype is mature, allowing users to have multi-turn conversations with Alexa Voice Service (AVS) or third-party skills. We open source the prototype for the benefit of users and researchers and provide a video demo showing MegaMind's performance and novel extensions.[2]

We evaluate MegaMind's implementation on three hardware platforms: two ARM SoC platforms, Raspberry Pi 4 (RPi 4) and Raspberry Pi 3 (RPi 3), and an x86-based laptop. Our ARM prototypes represent lower-end mobile devices such as smartphones, modestly-powered standalone assistants, and embedded ones. Our x86 prototype, on the other hand, represents higher-end and more powerful assistants. MegaMind achieves good performance on the RPi 4 and on the laptop, but suffers from high performance overhead on the weaker RPi 3. This performance discrepancy is expected; MegaMind has moderate local processing needs, including speech-to-text conversion and NLP processing, and the RPi 3 processor is not powerful enough to meet these needs [16]. Nevertheless, our evaluation shows that MegaMind's processing requirements can still be met by an inexpensive platform such as the RPi 4. We also perform extensive testing to evaluate MegaMind's ability to deliver on its security and privacy goals using a large corpus of sample conversations. Our results show that MegaMind achieves high accuracy (less than 10% false positive and false negative rates) in many cases, although it can sometimes experience lower accuracy. Our investigation shows that local speech-to-text conversion is an important contributor to MegaMind's inaccuracy. We expect that future conversion engines will further improve MegaMind's effectiveness.

We make the following contributions.

- We present the first extensible platform for enhancing the security and privacy of voice assistants.
- We design a programming model for extensions that enables ease of development and high expressibility.
- We design an extension execution framework that provides permission enforcement and non-interference guarantees for the extensions.
- We demonstrate novel extensions for voice assistants, including extensions for secure conversation and anonymous query. These extensions provide security guarantees not possible in today's voice assistants.
- We perform several optimizations in our prototype to achieve a low conversation latency overhead, critical for the adoption of MegaMind in practice.
- We perform an extensive evaluation of MegaMind and show that it incurs a small conversation latency overhead, has modest CPU utilization, and is effective in achieving its security and privacy goals.

2 MOTIVATING EXTENSIONS

MegaMind enables many security and privacy extensions. In this section, we describe extensions we have developed and tested.

Secure conversation. A user may want to converse in a secure manner with a trusted third-party skill, such as a bank or a health provider skill. The user may want to protect the conversation detail both from other third-party skills and from the voice assistant cloud service, e.g., AVS. In §7.1, we discuss how an extension can provide end-to-end encryption, integrity, and rollback protection for such conversations. MegaMind lets a user send and receive ciphertext over AVS, a novel functionality not demonstrated before.

Anonymous queries. A user may want to issue a sensitive query without revealing their identity. The user does not want the assistant cloud service or any third-party skills to associate the query with them. For example, a user may want to issue a medical query anonymously to protect the user's underlying medical conditions. In §7.2, we show how a MegaMind extension along with a mixer skill can realize this novel security feature.

Redaction. This extension's goal is to protect sensitive user information, such as family members' names, phone numbers, or credit card numbers, from being revealed to AVS or third-party skills.

[2]https://trusslab.github.io/megamind/

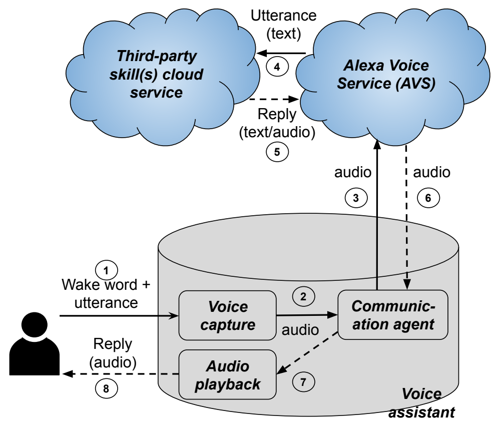


Figure 1: ***Amazon Alexa voice assistant architecture.***

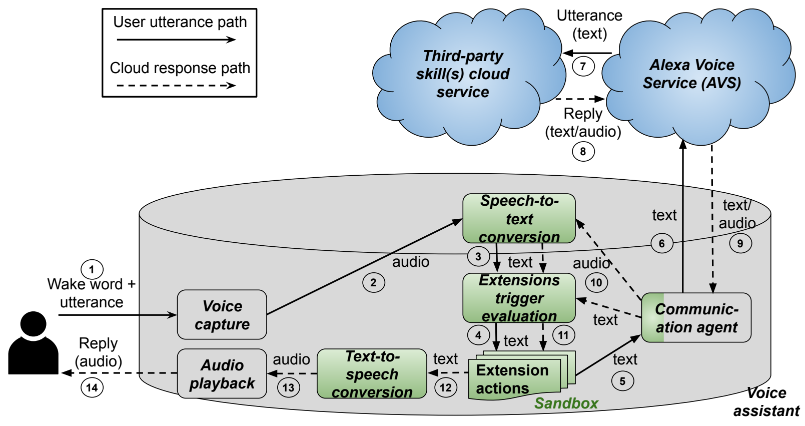


Figure 2: ***Adding the MegaMind extensibility platform to Amazon Alexa. MegaMind's functionality is shown in green.***

Night mode. A user may want to disable an assistant placed in a certain location (e.g., bedroom) during a certain time period (e.g., 10 PM to 7 AM).

Parental control. A user may want to enforce access control policies when a voice assistant is used by their children. For example, they might want to limit access to certain skills or limit usage time periods. Moreover, they may want to block any form of purchases and prevent assistant's responses from including adult content, violent content, or profanity.

Phone call control. A user might want to block the assistant from making calls to phone numbers outside a contact list. This can prevent unintended and malicious calls triggered by the voice assistant [8, 13, 14].

Third-party skill limiter. A user might want to limit the set of third-party skills their assistant can communicate with. Such an extension would help mitigate voice squatting attacks [60].

Please note that these are only a few examples of extensions that MegaMind enables. MegaMind's programming model and permission system allow the development of various extensions capable of performing edge computing in a controlled and secure manner.

3 ARCHITECTURAL OVERVIEW

This section presents an overview of MegaMind's architecture. For simplicity and without loss of generality, our description of MegaMind is based on Amazon Alexa voice assistants.

Figure 1 illustrates the interactions among users, their voice assistant, AVS, and third-party skills in existing commodity voice assistants. A wake word, such as *"Alexa"*, invokes the assistant to start recording a user's *utterances* and send them to AVS. AVS parses the audio and interprets it as a user request. Note that some captured utterances could be the result of accidental [7, 8, 14] or malicious [49, 65, 74] wakings of the assistant rather than intended user requests. AVS handles the request internally (i.e., using *built-in skills*) or sends it to a third-party skill for processing.

In addition to one-shot queries (i.e., a question and an answer), the voice assistant may enter a *dialog mode* that consists of multiple questions and answers, forming a *multi-turn conversation* [46]. Dialog mode's goal is to gather and confirm all the information needed for servicing a user request. For example, when ordering an Uber, AVS may ask about the type of the ride, the number of passengers, and the departure time. Hereafter, we refer to all requests and responses in a conversation as one *session*. In each session, the user interacts with either a built-in AVS skill or with a third-party one.

Figure 2 shows the voice assistant architecture when incorporating MegaMind. MegaMind interposes on communications from the voice assistant device to the voice assistant cloud service. It converts the user's utterance to text, evaluates it against the *trigger rules* of deployed extensions, and invokes the extensions' *action functions* on a trigger rule match. To protect against malicious extensions, MegaMind provides permission enforcement (§5) and non-interference (§6) to limit what an extension can do. Once processed, the text-based utterance is sent to AVS.

On receiving a response, MegaMind's deployed extensions process it before sending the possibly modified response for audio playback. The response from AVS can be in audio or text formats. If in text format, the communication agent directly sends it to be evaluated by the extensions. If not, the audio is first converted to text. Finally, the response needs to be converted back to audio for playback for the user. MegaMind achieves this using a text-to-speech converter.

The figure also shows that MegaMind executes the action functions of extensions in sandboxes (§8.1).

4 TRUST & THREAT MODEL

There are seven participants in MegaMind's ecosystem: 1) the owner of the device, 2) the user of the voice assistant (which might be the owner or someone else in owner's household such as their children) 3) voice assistant cloud service provider and its vendor (AVS/Amazon in case of Alexa), 4) the vendor of the voice assistant hardware, 5) third-party skills, 6) MegaMind, including all its software components, and 7) extensions, including (7A) extensions' manifest and (7B) extensions' action function. We note that (4) can be different from (3). For example, Amazon allows third-party hardware developers such as Sonos to build voice assistant devices that use its voice assistant cloud service. Even users can deploy Amazon's open-source Alexa SDK on their personal computers. Therefore, (4) provides the voice assistance hardware and the system software (OS and firmware) running on it.

We develop MegaMind's threat model from the perspective of the device owner (1). We assume that the owner always trusts the voice assistant hardware vendor (4) and MegaMind (6). The owner does not trust the MegaMind third-party extensions' action functions (7B). Moreover, we assume the owner reviews the installed extensions' manifest (7A) and ensures it does not ask for malicious permissions. Hence, we suggest that extension developers make the manifests publicly auditable. We also suggest that extensions developers sign extensions' manifest and action functions. This signature can be checked before installing the extension to verify its authenticity. Unless otherwise specified, the owner does not trust users (2). Users may accidentally share their private information or intentionally perform actions that the owner does not authorize.

In addition to the mentioned trust model, which applies to all the extensions, we discuss the trust model specific to different goals each extension tries to achieve.

1. Privacy protection. These extensions prevent users from accidentally sharing their private information with AVS and/or skills. For these extensions, the owner does not trust AVS (3) and third-party skills (5).

2. Content control. These extensions prevent AVS and skills to send sensitive responses to users. For these extensions, the owner does not trust AVS (3) and third-party skills (5).

3. Action limiting. These extensions prevent users from conducting some actions. For example, the owner might deploy one extension to block all purchases for the device installed in their child's room. For these extensions, the owner needs to trust AVS (3) or third-party skills (5) as they can conduct such action without the device sending them the request anyway.

4. Skill allow-listing and deny-listing. The owner uses these extensions to selectively trust a subset of skills. Using these extensions, the owner can either block the usage of a subset of skills or selectively apply privacy preserving or content control on them. For these extensions, the owner trusts AVS (3), and trusts/distrusts a subset of third party skills (5). Please note the owner needs to trust AVS since it is in charge of routing the commands to the third-party skills.

5. Skill security enhancement. The owner uses these extensions to provide a security feature for one specific third party skill. These extensions use secure channels to protect the content of the conversation from AVS. Using this extension, the owner trusts only one third party skill and its companion extension on its assistant.

Adversarial model. We assume that the adversary has full control over untrusted participants other than the user. For example, when we assume that AVS is untrusted, we assume that it is controlled by the adversary. When the user is untrusted, we assume they can make mistakes, but they do not act maliciously. Side-channels attacks, physical attacks, and any other attacks that allows the adversary to compromise a trusted participant are out of our scope.

Defense against attacks. Several important attacks have been demonstrated on voice assistants. In §10, we describe these attacks and mention how MegaMind extensions help protect against them.

5 PERMISSION ENFORCEMENT

Since MegaMind extensions are developed by untrusted third parties, we need to enforce limitations on these extensions. MegaMind provides a permission system similar to the Android permission system for applications. The owner reviews permissions an extension asks for and installs it only if she approves the permissions.

Despite similarities, MegaMind's permission model is fundamentally different from Android's. Android's permissions help limit the application in accessing different systems resources such as I/O peripherals. In contrast, MegaMind's permission system divides the permissions of each extension in two categories, *access permissions* and *modification permissions*. Access permissions limit the conversations an extension can mediate, and modification permissions limit how they can modify them.

Limiting the extensions' permissions is not trivial and requires special care. Some extensions, such as security enhancement extensions require to arbitrary change the phrases. However, it is not secure to give all extensions this permission. Also, it is not secure to allow a security extension to access and modify all the communications. As a result, we devise a permission model that strikes a balance between access and modification permissions. In our permission model, extensions that have more access permissions have more restricted modification permissions. Based on this model, we divide extensions in three different types: discarders, sanitizers, and companions. We define each of these types and their permission model in the §5.2.

In the rest of the section, we first describe how an extension expresses its access permissions through trigger rules. We then discuss how extension types bring a balance between access and modification permissions.

5.1 Access Permissions

Every extension declares its required permissions in its manifest, a JSON formatted file consisting of an extension type and a rule-set (i.e., an array of *trigger rules*) (Figure 3). Trigger rules indicate an extension's access permission by specifying the utterances or responses (i.e., jointly referred to phrases hereafter) the extension needs to process. MegaMind provides a description language that makes it easier to declare trigger rules in a generic and transparent fashion. It facilitates reviewing the manifest to find malicious permissions. Besides, in §5.2, we show how MegaMind helps in preventing malicious access permissions by limiting the trigger rules an extension can use based on the extension's type.

Trigger Rules Description Language Each extension expresses its access permissions in a rule-set. MegaMind evaluates the rules of the rule-set against each phrase and, *if any of the rules evaluates to true*, MegaMind executes the action function on all subsequent phrases of the current *session*. A rule itself is a set of *conditions* on *keywords*, *time*, or *third-party skill ID*. These conditions are grouped in two sets: (1) an inclusive disjunction (shown with the predicate *include_or* in Figure 3), and (2) an exclusive conjunction (shown with the predicate *exclude_and*). A rule evaluates to true *if all of its conditions are true.*

Our trigger rules description language is expressive because it allows arbitrary trigger logic using its language constructs. Using this language, an extension can request access permissions to

```
{
 "type": "Extension Type", // discarder, sanitizer, or companion
 "rule_set": [
    // rule1
    {
      "keywords": {
        "include_or": [
          "contain(word 1)",
          "adult_word()", ... // can add more included words
        ],
        "exclude_and": [
          "synonym(word 2)", ... // can add more excluded words
        ]
      },
      "time": {
        "include_or": [
          { // time range 1
            "start": "start time 1",
            "end": "end time 1"
          }, ... // can add more included time ranges
        ]
      },
      "Skill_ID": {
        "exclude_and": [
          "skill ID 1", ... // can add more excluded skill IDs
        ]
      },
    }, ... // can add more rules
  ]
}
```

Figure 3: ***Trigger rules description language.***

phrases containing or not containing certain words, occurring in certain time periods, and belonging to conversations with certain skills. A MegaMind rule-set is the sum of products (SoP) of the conditions on keywords, time and skill ID, and any arbitrary logic can be expressed as an SoP.

NLP Helper Functions Although MegaMind's trigger rule description language is expressive, when using a natural language, constructing a set of conditions using keywords alone is challenging. Consider an extension that blocks adult content from responses returned by third-party skills. It is difficult to construct a comprehensive corpus of adult content using keywords alone. To improve MegaMind's expressiveness and ease of use, we provide *Natural Language Processing Helper (NLP Helper) functions*. Examples of functions provided in our prototype look for synonyms, antonyms, first and last names, phone numbers, addresses, violent content, adult content, and profanity. Third party extensions can only use the NLP helper functions and cannot modify or train them. Thus, they cannot use them as an attack surface to increase their access permissions.

Secure Skill ID Detection Skill ID is a crucial factor in determining access permissions of an extension. Hence, MegaMind needs to associate each phrase to a target skill in a secure manner. MegaMind uses two methods for skill ID detection: an *AVS-dependent method* and a *local method*. We use the former when AVS is trusted and the latter when it is not (e.g., for extensions that provide skill security enhancement, i.e. companions; see §4).

For the AVS-dependent method, we leverage the fact that AVS tags each response with the ID of the skill that provided it. Therefore, we this AVS's metadata to associate a skill ID to a phrase.

For the local method, we try to detect skill invocations locally by analyzing the user's utterances. AVS establishes a conversation session between a user and a third-party skill in two ways: explicit invocation and implicit invocation. An *explicit* invocation is when the user deliberately invokes a skill using a specific grammar, e.g., "Open Uber". This grammar is deterministic and known by users and AVS [40]. An *implicit* invocation occurs when AVS delegates handling a request to a skill without the user asking. The implicit invocation only occurs with skills that implement the *name-free skill invocation* feature [40]. Our local method can only detect explicit invocations. This is because the grammar of an explicit skill invocation is known but there is no specific grammar for implicit ones. As a result, third-party skills with companion extensions (that need to rely on MegaMind's local method for their security) cannot (and should not) implement the name-free invocation feature.

5.2 Modification Permissions

When a phrase triggers a rule, MegaMind invokes the extension's action function to process the phrase. It is not secure to permit extensions to modify the phrases arbitrarily. The set of modifications MegaMind allows an extension to make depends on the extension type, as described next.

Discarder. A discarder's action function can drop a phrase from the conversation but cannot modify it. Whenever a discarder extension drops a phrase, MegaMind notifies the user by saying: "The last phrase is dropped by [discarder's name] extension." This notification prevents a malicious discarder extension from stealthily dropping a sensitive phrase.

Given the limited modification permissions the discarders have, we give them broad access permission by setting no limitation on their trigger rules. Discarders are useful for action limiting and skill deny-listing goals.

Sanitizer. A sanitizer's action function is allowed to modify the phrase, however, not arbitrarily. MegaMind enforces two constraints on sanitizers' modification permissions: (1) it can change only the words within a phrase that its triggered rule specifies under the *keywords* label (including those detected by NLP helpers), and (2) its replacement words cannot be arbitrary but must be drawn randomly by MegaMind from a specific category, such as a first or last name, a phone number, a city name, a country name, a random N-digit number, or a bleep censor (i.e., the word *bleep*). In §6, we discuss how this second limitation is also critical to MegaMind's guarantee of non-interference.

As sanitizers have more modification permissions, we limit their access permissions. Sanitizers cannot use *exclude_and* in the keyword section of their rule-set. This prevents sanitizers from excluding a rare category of words and getting triggered on many generic phrases. Sanitizers are useful for privacy protection and content control goals.

Companion. A companion extension is paired with a specific third-party skill. Therefore, its rules must list only one third-party skill ID that the extension accompanies. A companion's action function is allowed to make arbitrary changes to phrases. A third-party skill can have a single companion extension only. Companion extensions have strong modification permission, but their access permission is heavily limited. They are suitable for implementing extensions for the skill security enhancement goal.

	User's requests						Assistant's responses					
E2 E1	Discarder		Sanitizer		Companion		Discarder		Sanitizer		Companion	
	UP	OP	UP	OP	UP	OP	UP	OP	UP	OP	UP	OP
Discarder	✓(E)	✓(E)	✓(E)	✓(E)	✓(E)	✓(E)	✓(E)	✓(E)	✓(E)	✓(E)	✓(E)	✓(E)
Sanitizer	✓(O)	✓(L)	✓(E)	✓(E)	✓(O)	✓(E)	✓(O)	✓(L)	✓(E)	✓(E)	✓(O)	✓(E)
Companion	✓(T)	✓(O)	✓(T)	✓(O)	✓(E)	✓(E)	✓(O)	✓(T)	✓(O)	✓(T)	✓(E)	✓(E)

Table 1: ***This table summarizes all possible types of interference extension E1 can cause on extension E2's execution. "✓" means MegaMind can prevent interference. In each case, interference is avoided by: Extensions' definition (E), Order of execution (O), Limitations on extensions (L), and Trust model (T).***

6 NON-INTERFERENCE

In some cases, the same phrase needs to be processed by more than one extension. For example, consider two extensions, one that redacts the names of people living in a household in all conversations, and one that implements secure conversation with a third-party skill. Both extensions require processing the conversations between the user and that third-party skill.

In MegaMind, the extensions operate on phrases sequentially. This is because parallel processing would undoubtedly result in conflicts in the output that cannot be trivially resolved. For example, merging the encrypted output of one extension with the redacted output of another is impossible.

Since the extensions process phrases one by one, their execution's order can affect the final output. We develop a specific ordering for extension execution, which prevents interference. Our proposed extension execution orderings (which we justify next) are:

User's requests: sanitizers execute first, followed by discarders, and then followed by companions.

Assistant responses: companions execute first, followed by sanitizers, and then followed by discarders.

6.1 Non-interference definition

We show that our proposed orderings guarantee non-interference defined by the following two criteria: 1) No *under-protection (UP)* and 2) No *over-protection (OP)*. Below, we define UP and OP based on the notion of *protection actions*. Protection actions are: discarding the whole phrase, sanitizing words in the phrase, or securing the phrase (which might involve any arbitrary changes by the companion.)

UP occurs when an extension: 1) undoes the effect of a previously executed protection action, or 2) prevents the later extensions from performing their protection actions. As an example for (1), consider two extensions, a companion and a sanitizer that redacts the adult content from the assistant responses. If the companion executes after the sanitizer, it can add the adult content back to the phrase. As an example of (2), consider a companion that encrypts the user requests. If the companion executes prior to a privacy protecting extension, which redacts the user's personal information, the encryption hides the private information and makes the privacy protecting extension useless.

No OP means if an extension performs any protection action on a phrase, it would have performed the same action to the original phrase. As an example of an OP, assume a scenario that a sanitizer extension runs before a discarder extension. If the sanitizer modifies a word in a phrase to a word forbidden by the discarder, the discarder will block that phrase. However, the discarder would not have blocked the original phrase.

Please note that the term "non-interference" might have been used with other meanings in other contexts and research fields. Any reference to non-interference in this paper refers to the above definition.

6.2 Non-interference guarantee

We list all possible interference types between extensions in Table 1. This section discusses how MegaMind provides a non-interference guarantee by preventing all these types. MegaMind prevents interference in four different ways: (1) access/modification permission limitations each **E**xtension has by definition, (2) **O**rder of execution, (3) extra **L**imitations MegaMind enforces on extensions to guarantee non-interference, and (4) the MegaMind's **T**rust model.

Below we discuss how MegaMind successfully prevents interference for every entry in Table 1 using one of the four ways mentioned above. In following paragraphs, we use `E1-E2-{req,resp}-{UP,OP}` naming convention to point to each entry in Table 1. We use '`*`', and '`{}`' to point to more than one entry. For example, `{dis,san}-*-*-UP` means UP interference a discarder or a sanitizer can cause on all extensions while processing user's requests or assistant's responses.

Extension Definition (1) Discarder extensions do not modify the phrases. As a result, they cannot cause any interference to other extensions (i.e. no `dis-*-*-*`). Please note that MegaMind's guarantee of non-interference does not protect against denial-of-service. A malicious discarder can drop all the messages. However, MegaMind will notify the user every time the discarder drops a message, and the user will uninstall the malicious discarder. (2) MegaMind only allows one companion extension per skill. As a result, the `comp-comp-*-*` interference never happens. (3) No other extension can modify the skill ID of a session. Hence, they cannot cause the companion to process a message with wrong skill ID (i.e., no `*-comp-*-OP`). (4) As discussed in 5.2, sanitizers cannot use the exclusion of keywords in their trigger rules. It means a specific word (or word category) should be present in the phrase to trigger a sanitizer. Also, they can only change those words to a random word drawn by MegaMind. Together, these two limitations prevent a sanitizer from re-changing a word previously changed by another sanitizer (i.e., no `san-san-*-UP`). The same limitations prevent a sanitizer from changing a word to cause triggering of another sanitizer (i.e., no `san-san-*-OP`).

Order of Execution (1) Sanitizers run before discarders; thus, they cannot undo the protections discarders provide and cause UP (i.e. no `san-dis-*-UP`). (2) Companions run immediately before sending users' requests to AVS and immediately after receiving AVS responses. As a result, sanitizers cannot compromise the integrity or confidentiality of phrases going to/from AVS. (i.e., `san-comp-*-UP`) (3) Companions run after discarders and sanitizers for user requests. Hence, there is no way for them to cause OP on discarders or sanitizers (i.e. no `comp-{san,dis}-req-OP`). (4) Companions run before discarders and sanitizers for assistant responses. Hence, they cannot undo the protection actions provided by discarders or sanitizers. (i.e. no `comp-{san,dis}-resp-UP`)

Extra Limitations (1) Without further considerations, a sanitizer can cause OP interference with a discarder (i.e. `san-dis-*-OP`). Consider a sanitizer that redacts personal phone numbers and a discarder that blocks all the calls to phone numbers outside of a contact list. In this case, when the user tries to call a number on his contact list, first the sanitizer changes it to calling a random number, then the discarder blocks the call. However, if we passed the original phrase to the same discarder, it would not block it. Hence, it is a case of OP.

In the mentioned case, the discarder and the sanitizer have an inherent conflict, and no order can result in their interference-free execution. MegaMind resolves this issue by identifying and preventing these conflicts at the time of extension installation. The rule we enforce is that a sanitizer and a discarder can have overlapping inclusion and exclusion keyword lists only if they work on a non-overlapping set of skill IDs. In the above example, the conflict would be resolved if the sanitizer extension makes an **exception** for a phone call skill and the discarder blocks phone calls **only** for that skill.

Trust Model (1) Companion extensions run after discarders and sanitizers for user's request with unlimited modification permission. Therefore, they can potentially undo the protection actions of discarders and sanitizers and result in UP (i.e., `comp-{san,dis}-req-UP`). (2) Because companions run before discarders and sanitizers for assistant responses, they can add keywords to the phrase which provoke discarders and/or sanitizers and cause OP (i.e., `comp-{san,dis}-resp-OP`).

However, a companion extension only runs when the user is conversing with its accompanied skill, and based on our trust model; it is as trusted as the skill. Companion is the last extension that processes the user's request before sending them to the skill, and it is the first extension that processes the incoming responses. The companion executes in an isolated sandbox; the only data it can access is the phrases in the ongoing conversation. Hence, whatever action the companion extension does could have been done by the skill itself. Thus, we do not consider these actions as interference with other extensions.

7 NOVEL SECURITY FEATURES

MegaMind enables novel security features for voice assistants. In this section, we provide more details on two such features.

7.1 Secure Conversation

This extension lets a user conduct a secure conversation with a trusted third-party skill. The whole conversation is encrypted, integrity- and rollback-protected, ensuring no intermediary including AVS have access to the conversation's plaintext nor can they tamper with the conversation's contents.

Workflow. Assume a user intends to converse securely with a bank skill named *Great Bank*. The user invokes the skill by a phrase like "Alexa, Open Great Bank". At this point, the extension gets executed on the voice assistant. Its first task is to share a symmetric *session* key with the skill. To do so, it generates the key, encrypts it with the skill's public key and waits for the response from the skill. The skill responds to the "Open Great Bank" message with a welcome message and asks the user if they want to establish secure communication to this skill. If the user answers "yes", the extension replaces the user's answer with "key is [encrypted key]" and sends it to the skill. AVS delivers the encrypted key to the companion skill. Since the key is encrypted with the public key of the skill, skill decrypts it with its private key. Now both sides have the same symmetric key. The session key is then used for encryption and integrity protection (HMAC). The skill sends all the messages back to the user encrypted using this symmetric key. The extension also, encrypts all the user's utterances with the session key and sends a message as follows for every utterance: "search for [ciphertext]", where "search for" is a carrier phrase, described later in the skill support subsection. In addition, messages include a monotonic counter value to prevent rollback. The endpoints also check that the counter value is incremented with no gaps to ensure no messages are dropped. Finally, session tear-down is done explicitly using an *end-of-session* exchange to ensure the endpoints received all messages. If any of these checks fail, the session terminates with an error and MegaMind instructs the user to try again.

Encoding. Voice assistants cannot send arbitrary data over AVS to a skill because AVS restricts messages to include lower-case letters and numbers only. This creates a challenge for sending ciphertext to a skill. To address this challenge, we encode the ciphertext using RFC 4648's base32 encoding that converts 5-bit data chunks to a code comprised of upper-case letters and the numbers between 2 and 7. With base32 encoding, each 5-bytes of ciphertext is converted to 8 characters, padded with '=' characters in case the encoded message's length is not a multiple of 8. Finally, we convert upper-case letters in base32 to lower-case and remove all the trailing '='. Decoding is done in a similar manner.

Skill support. A third-party skill can offer this functionality by developing a skill-specific companion extension. This skill and its companion extension only communicate through AVS. The extension sends the encryption key and encrypted messages to the skill in the same way as regular messages (i.e., using AVS). Consequently, the skill must register specific *intents*, *sample utterances*, and *slots* with AVS to let the extension achieve this goal. An intent represents an action that fulfills a user's request. The sample utterances indicate the pattern of the words users can say to invoke intents. And slots are the optional arguments of intents.

Slots are defined with different types. Amazon provides several built-in slot types to capture first names, phone numbers, city names, etc. In addition to the built-in slot types, users can leverage a specific slot type for capturing users' generic queries. This slot type is `AMAZON.SearchQuery`, which is designed to be used in search engine skills or any skill that needs to capture complete phrases from users. We use this slot type to receive the key from the assistant.

In the case of the secure conversation extension, the skill should register the following intents with AVS:

- Intent1: KeyIntent
  Sample utterance: key is {KEY}
  Slots: KEY ; Slot type: AMAZON.SearchQuery
- Intent2: SearchIntent
  Sample utterance: Search for {PHRASE}
  Slots: PHRASE ; Slot type: AMAZON.SearchQuery

This, however, raises a challenge. To protect user's privacy, AVS does not allow the first intent that launches the skill to contain a slot with `AMAZON.SearchQuery` type. This limitation prevents us from converting the user's first request to a phrase such as *Alexa, open secret health with the {KEY}*. Therefore, we share the encrypted symmetric key at the second request to the skill, as described earlier in the workflow. Besides, Amazon also enforces another limitation on the usage of a slot type used in this skill (i.e., AMAZON.SearchQuery) for privacy purposes. Based on Amazon's rules, any `AMAZON.SearchQuery` slot should be accompanied by carrier phrases and cannot be used alone in an utterance [38]. This limitation is why we added the phrase "search for" to the beginning of the utterances of our SearchIntent intent.

7.2 Anonymous Query

This extension lets a user issue sensitive queries anonymously. The query is relayed indirectly through a *mixer skill* that prevents AVS or any other skill (including the mixer itself) from correlating the user's identity with the query's content.

This extension forms a secure channel with the mixer. Upon receiving messages from a user, the skill cannot identify the user because AVS never shares any of the user's identity information with third-party skills. Thus, the mixer skill receives the user's query but does not know the user's identity. On the other hand, AVS knows the user's identity but does not have access to the query's contents. This separation ensures that the sensitive query remains anonymous. Note that we assume that AVS and the mixer skill do not collude.

Workflow. First, the user must invoke the mixer skill explicitly by saying "Alexa, Open Query Mixer". The mixer skill asks the user to submit the query to be anonymized, and the extension sends the query to the mixer over the secure channel. The mixer skill decrypts the query and submits it in plaintext format to AVS. Upon receiving the response, the skill forwards it back to the user over the secure channel.

We implement a prototype mixer for demonstration purposes. Our mixer reorders the requests, adds randomized latency to each request, and submits them to AVS. Moreover, we assume that the mixer skill has access to a small network of Alexa-enabled devices to submit the queries. The mixer fends off several possible attacks by AVS. We do not further discuss these defenses due to space constraints.

8 IMPLEMENTATION

We implement MegaMind on top of Amazon's Alexa Voice Service SDK. Therefore, MegaMind is compatible with all built-in and third-party skills deployed on the Alexa ecosystem.

8.1 Key Implementation Components

MegaMind engine. The MegaMind engine is the conductor orchestrating all other components. The Alexa SDK sends an IPC signal to the MegaMind engine whenever it detects the wake word. It then waits until it receives the processed user's command from the MegaMind engine using another IPC channel. Upon receiving the wake-word detection signal, the MegaMind engine uses the speech-to-text engine to obtain the transcribed text of the user's command. MegaMind engine similarly processes the response from AVS. Besides, it forwards the altered response to a text-to-speech engine to read it for the user.

Runtime sandbox. We use the Firejail sandbox [25], which uses Linux namespaces and seccomp, to completely isolate the execution of the action function from the rest of the system.

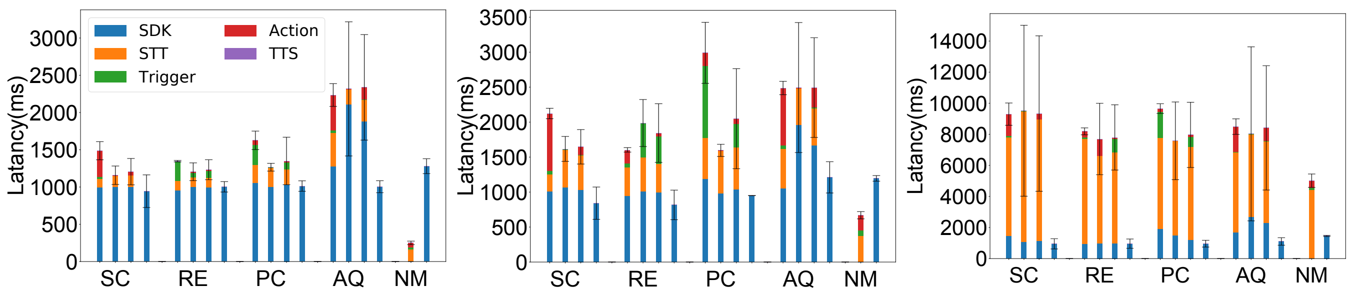
Using the sandbox, we enforce the following restrictions. First, we limit file system accesses. We configure Firejail to only allow access to the random number generator, libraries, and binary programs that are essential for execution of action functions (which are written in Python) including Python packages on cryptography and natural language processing. Moreover, for performance purposes, we allocate a temporary subfolder in the RAM-based `/tmp` directory to be used as a home directory for the sandbox, which is needed for temporary storage and communication with MegaMind. Second, we disallow sandboxes to communicate with the outside world by disabling network access. Third, using seccomp filters, we limit the syscalls that can be executed. We only allow `open`, `read`, and `write` syscalls. Finally, we configure Firejail to limit the resources such as memory, number of files and size of files, and the CPU time available to a sandbox.

Speech-to-Text conversion. We use Mozilla DeepSpeech as our speech-to-text engine for the conversion. We choose DeepSpeech because it outperforms all of the open source speech-to-text conversion implementations that we have tested, including Kaldi, CMUSphinx, Julius, and Simon. According to Mozilla, DeepSpeech achieves a 7.5% word error rate on LibriSpeech clean database, and can convert speech to text faster than real-time even on a single core of a RPi 4 board [15]. Moreover, our engine uses the Voice Activity Detection (VAD) algorithm to detect the end of utterances and responses based on the silence detection.

Text-to-speech conversion. For our text-to-speech conversion, we use pico2wave from SVOX [36]. pico2wave is fast; it converts the text to speech in less than 1 ms on a normal laptop. The output voice of pico2wave sounds less artificial than its other alternatives such as eSpeak engine on Linux.

NLP helper functions. We implement MegaMind NLP helper functions using Python natural language processing toolkit (NLTK). MegaMind helper functions lie within three main categories.

The first category includes helper functions that search words in a database. Examples are functions that look for first/last names and profanity. For the former, we use a database including the top 5000 of all the first/last names registered for a Social Security card



(a) x86 laptop (b) RPi 4 board (c) RPi 3 board

Figure 4: *Latency breakdown for different extensions for three platforms. For each extension, first, second and third bars, respectively, show the average latency for first, middle, and overall commands in a session. The last bar shows the baseline latency for that extension.*

in the United States since 1879 [2]. For the latter, we use a list of 1383 profane words in English from CMU [34]. The second category includes helper functions that look for a predefined structure in the sentence. Examples are helper functions that find phone numbers or U.S. addresses in sentences. The third category includes helper functions that use information about a word's meaning. Examples are helper functions that find synonyms and similar words in the sentences, or functions that find a specific type of content such as violent content or adult content. We implement these helper functions with the help of NLTK and the WordNet database [42].

8.2 Performance Optimizations

Sandbox pool. One source of overhead in our earlier prototypes was the sandbox initialization time. To avoid this high latency, we use a *sandbox pool*, which MegaMind initializes at boot time. This optimization reduces the latency by 420 ± 50 ms on a laptop, and 295 ± 20 ms on a RPi 4 board.

Text submission to AVS. Another source of overhead in our earlier prototypes was the time needed to covert the modified user utterance to audio in order to submit to AVS. To eliminate this overhead, we used another API of AVS that allows submission of requests in text format. This API is used in Alexa Developer Console for testing third-party skills. Moreover, the response from AVS, which is normally in audio format, includes the corresponding text of the response as well, when we submit the requests in text format. This eliminates the need for converting the response to text before passing it to the extensions, further reducing the latency. This optimization reduces the latency by about 310 ± 20 ms on a laptop and 630 ± 90 ms on a RPi 4.

Stream processing. If we wait until the end of the utterance and then start converting the recorded audio to text, it adds several seconds of latency. Therefore, we use stream processing for the conversion. In its recent versions (> 0.6), DeepSpeech provides a full streaming API. We use webRTC's VAD to collect 300 ms of audio from the microphone and pass it to the DeepSpeech streaming API. This way, regardless of the utterance length, we can have the converted text after around 300 ms. This optimization reduces the latency by 2000 ± 1500 ms.

9 EVALUATION

We evaluate the performance and effectiveness of MegaMind. We deploy MegaMind on three platforms, a laptop, a RPi 4 board, and a RPi 3 board. On the laptop, we use a VMware virtual machine with 4 CPU cores and 2 GB of RAM. The laptop uses a 2.6 GHz Intel Core i7 x86 CPU with 4 cores, 8 GB of RAM, and Intel hardware virtualization (VT-x). RPi 4 uses a 1.5 GHz ARM Cortex-72 with 4 CPU cores and has 2GB of RAM. Finally, RPi 3 uses a 1.2 GHz ARM Cortex-53 with 4 CPU cores and has 1 GB of RAM. Our ARM prototypes represent lower-end mobile devices such as smartphones, modestly-powered standalone assistants, and embedded one. Our x86 prototype, on the other hand, represents higher-end and more powerful assistants.

9.1 Performance

Conversation latency. The latency of responses is one of the critical factors in user's satisfaction with a voice assistant. We measure the latency for several extensions and report them in Figure 4. The figure shows the breakdown of the latency, showing contributions from local speech-to-text conversion, NLP helper function evaluation, and execution of computationally-heavy action functions. In addition, the figure shows the latency for the *first utterance in a session*, *all utterances after the first one*, and *all utterances including the first one*. We show the results as such since the first utterance in a session suffers from higher latency than the other utterances in the same session, for several reasons. First, the action function execution initializes at this utterance. Second, extensions, specifically companion extensions, perform heavy computations at the first utterance. The latency also depends on the length of an utterance. As a result, we test each extension with a session consisting of five utterances with different lengths. To further reduce the measurements' noise, we repeat each session five times. The final latency for each extension is the average latency of twenty-five measurements in five different sessions.

As the figure shows, different types of extensions have different latency profiles. First, secure conversation (SC) and Anonymous query (AQ), have heavy initialization and impose higher latency on the first utterance. As we can see in the figure, most of this latency comes from the execution of the action function. Second,

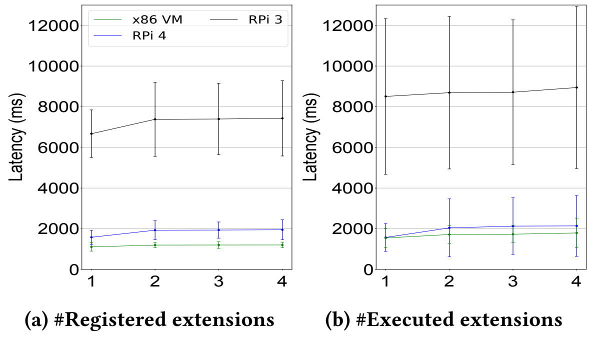


Figure 5: *Impact of number of extensions on latency.*

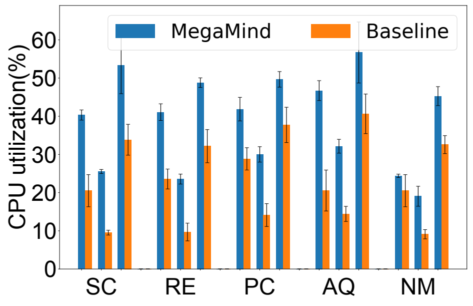


Figure 6: *Extensions CPU utilization. For each extension, first, second, and third bar groups, respectively, represent laptop, RPi 4, and RPi 3.*

for redaction (RE) and parental control (PC) extensions, evaluating the trigger functions imposes the highest latency. This is because of the usage of NLP helper functions in the trigger rules of these two extensions. Finally, night mode (NM) and skill limiter (SL) experience a small latency since neither their action functions nor their rule evaluations are computationally intensive. Please note since these two extensions discard the utterances before submission to the SDK, there is no reported SDK latency for them in the figure.

Speech-to-text conversion, on average, imposes similar latency to each extension in each platform. On the laptop and RPi 4, speech-to-text performs near real-time and imposes less than 400 ms of latency on average. However, for the weaker platform, RPi 3, speech-to-text conversion imposes notable latency. This explains poor performance results on this board. As MegaMind relies heavily on local computation, it requires adequate compute power on the voice assistant. However, RPi 3 has much less computation power compared to RPi 4 [17]. Fortunately, our results show that a device as inexpensive as RPi 4 can provide adequate compute power. Most of the latency on weaker devices is for speech-to-text conversion. Hence, such devices (e.g., smartwatches) can offload the speech-to-text conversion to another edge device like a smartphone and still be able to deploy MegaMind.

Impact of the number of extensions. We next evaluate the effect of the number of extensions on latency, in two steps. First, we evaluate the impact of *the number of active yet not triggered extensions*. In this experiment, we measure the average latency for a sequence of utterances that do not trigger any of extensions. This way, we measure the overhead of evaluation of the trigger rules for these extensions, but not their action functions. To increase the number of extensions, we add the following in order: (1) secure conversation, (2) redaction, (3) night mode, and (4) skill limiter. Figure 5a shows the results. It shows that increasing the number of enabled extensions does not have a notable impact on the overall latency. Only adding the redactor, which uses NLP helper functions slightly increases the overall latency of MegaMind.

Second, we evaluate the effect of *the number of triggered extensions*. This experiment captures not only the impact of evaluation of trigger rules, but also execution of actions functions. For this experiment, we use utterances that trigger all of the extensions. Please note that we modify the discarder's action function for this experiment to avoid discarding the utterances when they get triggered. Figure 5b shows the results. It shows that even if an utterance triggers four MegaMind extensions, the overall latency is only slightly higher than the latency of only one extension. This is because most of MegaMind latency comes from speech-to-text conversion (which executes once per command).

CPU utilization. We also measure the CPU utilization of each extension using the same experimental setup. Figure 6 shows the results. Since the unmodified SDK delegates almost all the computation to AVS, it is not surprising that running MegaMind increases the CPU utilization. For all platforms and all extensions, CPU remains idle most of the time. Thus, this increase in CPU utilization does not disrupt the normal execution of the assistant.

9.2 Effectiveness

MegaMind extensions can effectively provide security and privacy features for voice assistants. As mentioned in the introduction, we developed a few extensions and demonstrated their usage in a video demo. We have developed a simple banking skill that supports MegaMind's secure communication alongside its MegaMind companion extension. In our experiments, the user securely communicates with this banking skill, logs in to his account, queries for his balance, and issues a transaction. We recorded the usage of this skill-extension pair and published it in our demo. In another instance, we showed the usage of an anonymous query extension. We showed how a user uses this extension to anonymously query for a medical condition.

Besides, we evaluate the impact of inaccuracies in speech-to-text conversion and NLP helper function components on the effectiveness of MegaMind. We note that these inaccuracies mainly impact sanitizers and discarders in MegaMind. They have, otherwise, minimal impact on companion extensions, such as secure conversation and anonymous query, for two reasons. First, companion extensions do not use NLP helpers in their trigger rules. Second, inaccuracies in the transcription can be easily mitigated by additional authentication methods employed by the companion skill.

We evaluate the effectiveness of MegaMind in four tasks: (1) detecting sessions, (2) redacting profanity, (3) redacting private information, and (4) preventing purchases. Table 2 summarizes the results. For each of the above tasks, we report results from two sets of experiment, one where we submit the test utterances in audio format hence requiring speech-to-text conversion, and

	Text		Voice	
Detection	FN	FP	FN	FP
New session	0%	8%	20%	8%
Profanity	0%	5%	10%	6%
Private info	0%	5%	15%	8%
Purchase	13%	1%	20%	2%

Table 2: *MegaMind's detection errors. FN stands for false negatives, and FP for false positives.*

one where we bypass the speech-to-text conversation and feed the accurate text of the utterances to MegaMind. These results help us understand the effectiveness of MegaMind in the presence of a highly accurate speech-to-text converter. Below, we discuss the effectiveness experiment results.

Effectiveness of skill ID detection.

To measure the accuracy of MegaMind in detecting explicit invocation of a skill, we test MegaMind with a combination of 100 standard built-in commands randomly chosen out of 190 Alexa built-in commands reported in [24] and twenty commands that we generate to ask Alexa to start a new session with a third-party skill. In generating these commands, we randomly chose the grammar to open the skill, and we chose skill names randomly from Alexa skill market.

Table 2 shows that MegaMind could find all of the commands aiming to start new session accurately. However, in a few cases MegaMind detects a false session start for a normal command. This is because the AVS grammar for starting a new skill has overlap with some of the Alexa's built-in commands. For example, a user can launch a third-party skill using the following grammar: *"[a request] from [skill invocation name]"* (e.g. *"Order pizza from great pizza shop".*) A built-in command such as *"Disconnect bedroom's echo device from John's phone"* follows the exact same grammar. MegaMind can potentially filter out all of these false new session detections by having a database of Alexa's published skills names.

Effectiveness of profanity redaction. For this experiment, we develop a custom skill, which tells jokes. We combine ten jokes containing profanity with one hundred clean jokes in a database, all randomly chosen from Laugh Factory [31]. Our result shows that MegaMind redacts all the profane words. However, since the database we used for bad words is conservative and contain dual-used words as well, MegaMind filters a few words in clean jokes as well.

Effectiveness of private information redaction. In this experiment, we mix twenty utterances containing private information such as first and last names, phone numbers, Social Security Numbers, with 100 Alexa's standard commands randomly chosen out of 190 Alexa built-in commands reported in [24]. Our result shows that MegaMind could successfully redact all the private information in the utterances. However, in a few cases, MegaMind falsely redacts standard Alexa commands. These false alarms mostly happen in commands related to playing music, in which the redactor redacts the name of the artist. This problem only occurs when the redactor aims to redact all the matching first and last names. However, in real cases, a redactor can be configured only to redact the name of people using the device.

Effectiveness of purchase prevention. Out of 190 built-in Alexa commands reported in [24], 15 commands are listed as purchase-related commands. We measure how accurately MegaMind parental control extension can block these purchase-related commands. MegaMind parental control extension could find 13 of purchase-related commands using MegaMind NLP helper functions. However, two commands related to getting a taxi from ride-share skills were missed by MegaMind because there were no words associated with purchasing a product in these utterances. However, the parental control extension of MegaMind can easily block these utterances by disabling ride share skills.

Speech-to-text conversion accuracy. The word error rate for DeepSpeech speech-to-text engine is reported to be 7.5% [15]. However, this word error rate is for generic conversations. Voice commands may contain some words and phrases that were not present in DeepSpeech's training data-set. As a result, we use a database of Alexa built-in commands [24] consisting of 190 commands for Alexa to measure DeepSpeech's accuracy. We convert these commands to Speech using a human-like neural network-based cloud text-to-speech converter. We then convert back the spoken commands to text using DeepSpeech and measure the accuracy. Our experiments shows that DeepSpeech word error rate for this data set is 12.28%.

One other important aspect of speech-to-text conversion accuracy is in finding skill names. We measure the accuracy of MegaMind using DeepSpeech in accurately detecting the skill names for 100 commands aiming to open 100 randomly selected skills from the top skills of Alexa skill market [19]. MegaMind could find the Skill names correctly in 82% of cases.

Speech-to-text conversion is a hot research topic and it is expected that the accuracy of local speech-to-text engines improves in the future. Our prototype uses a pre-trained English model for DeepSpeech, which has been trained with generic English speeches. However, we envision that in the near future, it will be possible to train a voice assistant-specific language model for DeepSpeech using voice assistant commands and skill names in order to further improve the accuracy. In addition, the DeepSpeech pre-trained models are only trained with noise-free pre-recorded standard and formal English speeches and do not support different accents and ambient noise. Training a robust language model requires a huge amount of labeled audio recording from users. Previously, only big companies had access to this kind of database. This task is getting feasible given the recent efforts from the open source community to build large transcribed databases of users' speeches by asking people to donate their voice to the database, and donate their time to validate the transcriptions [33]. For instance, Mozilla Common Voice project, at the time of writing this paper has reached 12000 hours of audio recording in 40 different languages, which 9500 hours of that is validated [33].

10 RELATED WORK

Our work on MegaMind is inspired by systems that provide security extensions for mobile operating systems, such as ASM [58] and ASF [47].

Almond [48] is an open source virtual assistant system. It uses a natural language interface and protects user's privacy by keeping their data locally. PrIA [59] is an intelligent assistant that provides personalized services, such as a news recommendation service, for the user without providing user's private information to cloud services. MegaMind shares Almond's and PrIA's visions of enhancing user's privacy when using assistants. Yet, we have designed MegaMind as a security and privacy extension to existing voice assistant systems, in contrast to these systems, which provide a new ecosystem or new services.

Use of assistants in a smart home setup creates security and privacy challenges when used by multiple people [73]. These challenges are different from those addressed by MegaMind, which focuses on security and privacy of using cloud services via voice assistants.

LipFuzzer [76] uses a linguistic-model-guided fuzzer to find semantic inconsistencies between the user and the voice assistant, resulting in the user talking to an unintended skill. While MegaMind's goal is orthogonal to LipFuzzer's, its extensions can alleviate some of these unintended results.

There is a line of work on enhancing the privacy of voice-based systems by eliminating personal features from audio recordings via local preprocessing [44, 45]. MegaMind's local speech-to-text conversion also eliminates all voice-based features. Although speech-to-text conversion requires more processing power, having the transcribed commands enables more sophisticated language processing.

Other previous research has also identified the need for voice assistants to provide strong security defenses including authorization, access control, and privilege separation [52, 55, 56, 67, 68, 72]. Besides, there are previous empirical studies that highlight the importance of security and privacy of smart speaker applications [51, 61, 66].

There is a large body of work showing different classes of security attacks on voice assistants. *Inaudible voice attacks (IVA)* and *concealed voice attacks (CVA)* stealthily deliver voice commands to a voice assistant without the user knowing. BackDoor [64], Dolphin [74] and LipRead [65] use inaudible sounds transmitted over ultrasound frequencies to issue inaudible voice commands to virtual assistants. Similarly, research projects on concealed voice commands showed that devices continue to respond to wake words and utterances even when "mangled" to such a degree that they are unintelligible to users [49, 69]. Recently, CommanderSong [71], Lyexa [62], SurfingAttack [70], MetaMorph [50], and Abdullah et al. [43] proposed more elaborate and practical inaudible/concealed voice attacks.

Another important attack is the *voice squatting attack (VSA)*, where a malicious skill developer creates an invocation phrase similar to a legitimate skill, in the hope that sometimes the wrong skill may be invoked and data may leak [60, 75].

Another important attack is the *fake skill termination attack (FSTA)*, [75], where a malicious skill developer creates a long silent audio response in order to trick the user into thinking that no skill is running anymore, at which point the user may say something private. Table 3 shows how MegaMind extensions help in protecting against attacks.

Attack category	Examples	How?
Phishing	VSA	Skill deny-listing
Eavsdropping	FSTA, CVA	Privacy protction
Unautorized cmd	IVA, CVA	Action limiting

Table 3: *MegaMind protection for attacks.*

11 CONCLUSIONS

We presented MegaMind, an extensibility system for enhancing the security and privacy of voice assistants. MegaMind enables development of powerful extensions with ease. We demonstrated several such extensions, including one for secure communication with a third-party skill and one for issuing queries anonymously, both of which bring a level of unprecedented security for voice assistants. We presented a prototype that works with the existing Alexa Voice Service ecosystem and showed that it achieves a low conversation latency even on inexpensive hardware, such as a Raspberry Pi 4 board. We also showed that MegaMind is effective in achieving various security and privacy goals.

ACKNOWLEDGMENTS

The work was supported in part by NSF Award #1846230. We thank the anonymous reviewers and the shepherd for their insightful comments. We also thank Hamed Haddadi for evaluating MegaMind's artifact.

REFERENCES

[1] 2016. Toddler asks Amazon's Alexa to play song but gets porn instead. https://nypost.com/2016/12/30/toddler-asks-amazons-alexa-to-play-song-but-gets-porn-instead/.
[2] 2016. US First Names Database . https://data.world/len/us-first-names-database.
[3] 2017. Cortana in the car: Microsoft launches new automotive tech platform, strikes Renault-Nissan partnership. https://www.geekwire.com/2017/cortana-car-microsoft-launches-new-automotive-tech-platform-strikes-renault-nissan-partnership/.
[4] 2017. Google Home can now power on your Vizio TV. https://www.cnet.com/news/google-home-can-now-power-on-your-vizio-tv/.
[5] 2018. Amazon Alexa-Powered Device Recorded and Shared User's Conversation Without Permission. https://www.wsj.com/articles/amazon-alexa-powered-device-recorded-and-shared-users-conversation-without-permission-1527203250.
[6] 2018. Google Assistant – LIFX.com. https://www.lifx.com/products/google-assistant.
[7] 2018. Hey, I didn't order this dollhouse! 6 hilarious Alexa mishaps. https://www.digitaltrends.com/home/funny-accidental-amazon-alexa-ordering-stories//.
[8] 2018. Is Alexa Listening? Amazon Echo Sent Out Recording of Couple's Conversation. https://www.nytimes.com/2018/05/25/business/amazon-alexa-conversation-shared-echo.html/.
[9] 2018. Meet STEMosaur! https://cognitoys.com/.
[10] 2018. SmartThinQ with VoiceAssistants. https://www.lg.com/us/support/works-with-google-alexa-voice-assistant.
[11] 2018. Tesla's Siri integration now works with Model 3. https://electrek.co/2018/03/20/teslas-siri-integration-now-works-with-model-3/.
[12] 2018. Volkswagen now lets Apple users unlock their cars with Siri. https://www.theverge.com/2018/11/12/18087416/volkswagen-vw-car-net-app-siri-shortcuts.
[13] 2018. Was the Alexa Butt Dial a Big Deal? Steps You Can Take if You Are Concerned. https://voicebot.ai/2018/05/28/was-the-alexa-butt-dial-a-big-deal-steps-you-can-take-if-you-are-concerned/.
[14] 2019. Amazon's Alexa Allegedly Calls 911 During Domestic Violence Incident. https://wflanews.iheart.com/featured/pm-tampa-bay-with-ryan-gorman/content/2019-07-17-amazons-alexa-allegedly-calls-911-during-domestic-violence-incident/.
[15] 2019. DeepSpeech 0.6: Mozilla's Speech-to-Text Engine Gets Fast, Lean, and Ubiquitous . https://hacks.mozilla.org/2019/12/deepspeech-0-6-mozillas-speech-to-text-engine/.

[16] 2020. DeepSpeech Slow Inference Speed Raspberry Pi 3B . https://github.com/mozilla/DeepSpeech/issues/3000/.
[17] 2020. Raspberry Pi 4 vs Raspberry Pi 3B+. https://magpi.raspberrypi.org/articles/raspberry-pi-4-vs-raspberry-pi-3b-plus//.
[18] 2021. Alexa for PC . https://developer.amazon.com/en-US/alexa/devices/pcs.
[19] 2021. Alexa skill market . https://www.amazon.com/alexa-skills/b?ie=UTF8&node=13727921011.
[20] 2021. Amazon Alexa. https://en.wikipedia.org/wiki/Amazon_Alexa.
[21] 2021. Amazon Echo. https://en.wikipedia.org/wiki/Amazon_Echo.
[22] 2021. AVS Device SDK. https://developer.amazon.com/alexa-voice-service/sdk.
[23] 2021. Cortana Devices SDK. https://developer.microsoft.com/en-us/cortana/devices.
[24] 2021. Every Alexa command you can give your Amazon Echo smart speaker. https://www.cnet.com/how-to/every-alexa-command-you-can-give-your-amazon-echo-smart-speaker/.
[25] 2021. Firejail Security Sandbox. https://firejail.wordpress.com/.
[26] 2021. Google Assistant. https://assistant.google.com.
[27] 2021. Google Assistant SDK for devices. https://developers.google.com/assistant/sdk/.
[28] 2021. Google Home. https://en.wikipedia.org/wiki/Google_Home.
[29] 2021. HomePod. https://en.wikipedia.org/wiki/HomePod.
[30] 2021. Invoke. https://en.wikipedia.org/wiki/Invoke_(smart_speaker).
[31] 2021. Laugh factory joke bank. http://www.laughfactory.com/jokes/.
[32] 2021. Mi AI Speaker. https://www.mi.com/aispeaker/.
[33] 2021. Mozzila Common Voice. https://voice.mozilla.org/en//.
[34] 2021. Offensive/Profane Word List . http://www.cs.cmu.edu/~biglou/resources/bad-words.txt.
[35] 2021. Personal Digital Assistant – Cortana Home Assistant. https://www.microsoft.com/en-us/cortana.
[36] 2021. Pico Text-to-Speech. https://www.openhab.org/addons/voice/picotts/.
[37] 2021. Siri – Apple. https://www.apple.com/siri/.
[38] 2021. Slot Type Reference. https://developer.amazon.com/en-US/docs/alexa/custom-skills/slot-type-reference.html/.
[39] 2021. Total number of Amazon Alexa skills from January 2016 to September 2019 . https://www.statista.com/statistics/912856/amazon-alexa-skills-growth/.
[40] 2021. Understand How Users Invoke Custom Skills. https://developer.amazon.com/en-US/docs/alexa/custom-skills/understanding-how-users-invoke-custom-skills.html/.
[41] 2021. Voice Assistant on Fitbit Smartwatches . https://www.fitbit.com/global/us/technology/voice.
[42] 2021. What is WordNet? https://wordnet.princeton.edu/.
[43] H. Abdullah, W. Garcia, C. Peeters, P. Traynor, K. R. B. Butler, and J. Wilson. 2019. Practical Hidden Voice Attacks Against Speech and Speaker Recognition Systems. (2019).
[44] R. Aloufi, H. Haddadi, and D. Boyle. 2019. Privacy Preserving Speech Analysis Using Emotion Filtering at the Edge: Poster Abstract. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SenSys '19)*.
[45] R. Aloufi, H. Haddadi, and D. Boyle. 2020. Privacy-preserving Voice Analysis via Disentangled Representations. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*. 1–14.
[46] Amazon Alexa. 2018. Create the Interaction Model for Your Skill. https://developer.amazon.com/docs/custom-skills/create-the-interaction-model-for-your-skill.html.
[47] M. Backes, S. Bugiel, S. Gerling, and P. von Styp-Rekowsky. 2014. Android Security Framework: Enabling Generic and Extensible Access Control on Android. *arXiv preprint arXiv:1404.1395* (2014).
[48] G. Campagna, R. Ramesh, S. Xu, M. Fischer, and M. S. Lam. 2017. Almond: The Architecture of an Open, Crowdsourced, Privacy-Preserving, Programmable Virtual Assistant. In *Proc. ACM WWW*.
[49] N. Carlini, P. Mishra, Vaidya T., Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou. 2016. Hidden Voice Commands. In *Proc. of 25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX).
[50] T. Chen, L. Shangguan, Z. Li, and K. Jamieson. 2020. Metamorph: Injecting Inaudible Commands into Over-the-air Voice Controlled Systems. (2020).
[51] L. Cheng, Ch. Wilson, S. Liao, J. Young, D. Dong, and H. Hu. 2020. Dangerous Skills Got Certified: Measuring the Trustworthiness of Skill Certification in Voice Personal Assistant Platforms. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*.
[52] T. Denning, T Kohno, and H Levy. 2013. Computer Security and the Modern Home. *Commun. ACM* 56 (2013). Issue 1.
[53] W. Diao, X. Liu, Z. Zhou, and K. Zhang. 2014. Your Voice Assistant is Mine: How to Abuse Speakers to Steal Information and Control Your Phone. In *Proc. of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices (SPSM)* (Scottsdale, AZ).
[54] D. Dubois, R. Kolcun, A. Mandalari, M. Paracha, D. Choffnes, and H. Haddadi. 2020. When Speakers Are All Ears: Characterizing Misactivations of IoT Smart Speakers. *Proceedings on Privacy Enhancing Technologies* (2020).
[55] E. Fernandes, .J Jung, and A. Prakash. 2016. Security Analysis of Emerging Smart Home Applications. In *Proc. of 37th IEEE Symposium on Security and Privacy* (San Jose, CA).
[56] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash. 2016. FlowFence: Practice Data Protection for Emerging IoT Application Frameworks. In *Proc. of 25th USENIX Security Symposium* (Austin, TX).
[57] Alex Hern. 2019. Apple contractors 'regularly hear confidential details' on Siri recordings. https://www.theguardian.com/technology/2019/jul/26/apple-contractors-regularly-hear-confidential-details-on-siri-recordings.
[58] S. Heuser, A. Nadkarni, W. Enck, and A. Sadeghi. 2014. ASM: A Programmable Interface for Extending Android Security. In *Proc. USENIX Security Symposium*. 1005–1019.
[59] S. Jain, V. Tiwari, A. Balasubramanian, N. Balasubramanian, and S. Chakraborty. 2017. PrIA: A Private Intelligent Assistant. In *Proc. ACM Workshop on Mobile Computing Systems & Applications (HotMobile)*.
[60] D. Kumar, R. Paccagnella, P. Murley, E. Hennenfent, J. Mason, A. Bates, and M. Bailey. 2018. Skill Squatting Attacks on Amazon Alexa. In *Proc. of 27th USENIX Security Symposium* (Baltimore, MD).
[61] S. Liao, Ch. Wilson, L. Cheng, H. Hu, and H. Deng. 2020. Measuring the Effectiveness of Privacy Policies for Voice Assistant Applications. In *Annual Computer Security Applications Conference (ACSAC '20)*.
[62] R. Mitev, M. Miettinen, and A. Sadeghi. 2019. Alexa Lied to Me: Skill-based Man-in-the-Middle Attacks on Virtual Assistants. In *Proc. ACM Asia Conference on Computer and Communications Security (AsiaCCS)*.
[63] D. Mukhopadhyay, M. Shirvanian, and N. Saxena. 2015. All your voices are belong to us: Stealing voices to fool humans and machines. In *Proc. of European Symposium on Research in Computer Security (ESORICS)* (Vienna, Austria).
[64] N. Roy, H. Hassanieh, and R. Choudhury. 2017. BackDoor: Making Microphones Hear Inaudible Sounds. In *Proc. of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)* (Niagara Falls, NY).
[65] N. Roy, S Shen, H Hassanieh, and R Choudhury. 2018. Inaudible Voice Commands: The Long-Range Attack and Defense. In *Proc. of 15th Symposium on Networked Systems Design and Impalementation (NSDI)* (Renton, WA).
[66] F. Shezan, H. Hu, J. Wang, G. Wang, and Y. Tian. 2020. Read Between the Lines: An Empirical Measurement of Sensitive Applications of Voice Personal Assistant Systems. In *Proceedings of The Web Conference 2020 (WWW '20)*. Association for Computing Machinery.
[67] A.K. Simpson, F. Roesner, and T. Kohno. 2017. Securing Vulnerable Home IoT Devices with an In-Hub Security Manager. In *Proc. of 1st International Workshop on Pervasive Smart Living Spaces (PerLS)* (Malmó, Sweden).
[68] B. Ur, J. Jung, and S Schechter. 2013. The Current State of Access Control for Smart Devices in Homes. In *Proc. of Workshop on Home Usable Privacy and Security (HUPS)* (Newcastle, UK).
[69] T. Vaidya, Y. Zhang, M. Sherr, and C. Shields. 2015. Cocaine Noodles: Exploiting the Gap Between Human and Machine Speech Recognition. In *In Proc. of the 9th USENIX Conference on Offensive Technologies (WOOT)* (Washington, D.C.).
[70] Q. Yan, K. Liu, Q. Zhou, H. Guo, and N. Zhang. 2020. SurfingAttack: Interactive Hidden Attack on Voice Assistants Using Ultrasonic Guided Waves. (2020).
[71] X. Yuan, Y. Chen, Y. Zhao, Y. Long, X Liu, K Chen, S Zhang, H Huang, X Wang, and C Gunter. 2018. CommanderSong: A Systematic Approach for Practical Adversarial Voice Recognition. In *Proc. of 27th USENIX Security Symposium* (Baltimore, MD).
[72] E. Zeng, Sh. Mare, and F. Roesner. 2017. End User Security & Privacy Concerns with Smart Homes. In *Proc. of the 13th USENIX Conference on Usable Privacy and Security (SOUPS)* (Santa Clara, CA).
[73] E. Zeng and F. Roesner. 2019. Understanding and Improving Security and Privacy in Multi-User Smart Homes: A Design Exploration and In-Home User Study. In *Proc. USENIX Security Symposium*.
[74] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu. 2017. DolphinAttack: Inaudible Voice Commands. In *CCS 2017* (Dallas, TX).
[75] N. Zhang, X. Mi, X. Feng, X. Wang, Y. Tian, and F. Qian. 2019. Dangerous Skills: Understanding and Mitigating Security Risks of Voice-Controlled Third-Party Functions on Virtual Personal Assistant Systems. In *Proc. of the IEEE Symposium on Security and Privacy* (San Francisco, CA).
[76] Y. Zhang, L. Xu, A. Mendoza, G. Yang, P. Chinprutthiwong, and G. Gu. 2019. Life after Speech Recognition: Fuzzing Semantic Misinterpretation for Voice Assistant Applications. In *Proc. Internet Society NDSS*.