Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

Mapeamento de vulnerabilidades no Amazon Echo através do uso de Alexa Skills

Renato Henrique Alpes Sampaio

Trabalho de Graduação

Recife
25 de setembro de 2024

Universidade Federal de Pernambuco
Centro de Informática

Renato Henrique Alpes Sampaio

Mapeamento de vulnerabilidades no Amazon Echo através do uso de Alexa Skills

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Adriano Augusto de Moraes Sarmento*

Recife
25 de setembro de 2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Sampaio, Renato Henrique Alpes.
Mapeamento de vulnerabilidades no Amazon Echo através do uso de Alexa Skills / Renato Henrique Alpes Sampaio. - Recife, 2024.
44 : il., tab.

Orientador(a): Adriano Augusto de Moraes Sarmento
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2024.
Inclui referências, apêndices.

1. Alexa. 2. Skills. 3. Amazon. 4. Echo. 5. Vulnerabilidades. I. Sarmento, Adriano Augusto de Moraes. (Orientação). II. Título.

000 CDD (22.ed.)

RENATO HENRIQUE ALPES SAMPAIO

MAPEAMENTO DE VULNERABILIDADES NO AMAZON ECHO ATRAVÉS DO USO DE ALEXA SKILLS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Aprovado em: 26/09/2024

BANCA EXAMINADORA

_____

Prof. Dr. Adriano Augusto de Moraes Sarmento (Orientador)

Universidade Federal de Pernambuco

_____

Profa. Dra. Carla Taciana Lima Lourenço Silva Schuenemann (Examinadora Interna)

Universidade Federal de Pernambuco

Agradecimentos

Este trabalho não poderia ter sido realizado sem o apoio da minha mãe e do meu pai, que ouviram com muita paciência todas as minhas explicações mirabolantes sobre o funcionamento deste projeto. A ideia original e o direcionamento da pesquisa vieram do meu *research advisor* (e cineasta!) Jodson Leandro, que também compartilhou muito de seu conhecimento durante meu estágio na Tempest. Este documento também não existiria neste formato sem meu orientador Adriano Sarmento, que também teve muita paciência com o tempo que foi necessário para a escrita e deu o *feedback* necessário para que esta monografia fosse a melhor possível. Por fim, agradeço também a oportunidade de pesquisa oferecida pelo estágio da Tempest, sem a qual nada disso teria sido possível.

I might be technically correct here but it doesn't matter for practical purposes. But I like to work at the intersection of theory and impractice. And so by doing a lot of work we can make it matter. And then I'll be even more right, both theoretically right, and it will only matter for most practical purposes.

—THOMAS MURPHY VII (GradIEEEnt half decent: The hidden power of imprecise lines)

Resumo

Contextualização: A Amazon Alexa é uma das assistentes virtuais mais utilizadas no mundo e o Amazon Echo é um dos dispositivos IoT que mais está presente em casas. Considerando que o ecossistema Alexa Skills permite que desenvolvedores não-afiliados a Amazon publiquem Skills que podem ser utilizadas por usuários finais, é interessante verificar se essas funcionalidades podem ser abusadas por estes. Objetivos: O objetivo deste trabalho é mapear vulnerabilidades conhecidas pela literatura no ecossistema Alexa Skills e reproduzir seus resultados de forma a verificar se ainda podem ser exploradas. Métodos: O mapeamento é realizado através da leitura da literatura existente e da documentação disponibilizada pela Amazon sobre o funcionamento de seus sistemas. A reprodução de resultados é realizada através do uso do console de desenvolvedor Alexa para a criação de Skills de teste que fazem uso das técnicas estabelecidas previamente. Resultados: No capítulo 4 foram listadas dez fraquezas que podem ser exploradas como vulnerabilidades para uso em ataques, bem como uma vulnerabilidade específica utilizada na literatura. Nos experimentos realizados foi possível reproduzir os ataques de Skill Squatting, Alexa vs Alexa, Mask Attack e Bypass de API de informações sensíveis, incluindo o uso de formas de contornar barreiras implementadas pela Amazon desde a publicação dos artigos originais. Conclusões: Conclui-se que os ataques listados ainda são viáveis, com pequenas modificações e que, portanto, é possível utilizar um dispositivo Amazon Echo como vetor de ataque contra usuários.

Palavras-chave: Alexa, Skills, Echo, Amazon, Vulnerabilidades

Abstract

Introduction: Amazon Alexa is one of the most common virtual assistants throughout the world, and Amazon Echo is one of the IoT devices that are most present within homes. When taking into account that the Alexa Skills ecosystem allows third-party developers to publish Skills that can be used by end-users, questions are raised regarding how these features may be used by malicious actors. Objectives: The main goal of this work is to map which vulnerabilities are known to the literature regarding the Alexa Skills ecosystem, and also to verify which results can still be reproduced today. Methods: This mapping shall be carried out through the examination of the available literature, and of the documentation provided by Amazon regarding the inner workings of their systems. The reproduction of results will be executed through the usage of the Alexa developer console for the development of test Skills that make use of the previously established techniques. Results: Chapter 4 lists ten weaknesses that can be exploited as vulnerabilities to be used as part of an attack, as well as a specific vulnerability that was used within the literature. During the course of the experimentation, it was possible to reproduce the Skill Squatting, Alexa vs Alexa, Mask Attack, and Sensitive information API bypass attacks, including the usage of ways to bypass barriers set up by Amazon since the original papers were published. Conclusion: This work concludes that the listed attacks are still viable, with small tweaks, and that it is therefore possible to use an Amazon Echo device as an attack vector against end-users.

Keywords: Alexa, Skills, Echo, Amazon, Vulnerabilities

Sumário

Lista de Figuras

CAPÍTULO 1

Introdução

1.1 Motivação

A assistente pessoal Alexa, desenvolvida pela Amazon, está presente em mais de 100 milhões de dispositivos[3]. Destes, os dispositivos da linha Echo são os mais comuns para uso pessoal. Uma das features oferecidas pela Alexa, é uma espécie de loja de aplicativos chamada "Alexa Skills", que permite que desenvolvedores *third-party* desenvolvam funcionalidades e as ofereçam para usuários finais da Alexa. Desta forma, é interessante verificar que partes do fluxo de funcionamento de uma interação com uma Alexa Skill podem ser exploradas por um desenvolvedor malicioso para atacar usuários.

A literatura relacionada já contém extensa discussão relacionada a possíveis ataques e vulnerabilidades encontradas no passado. Este trabalho tem a intenção de agregar os resultados mais relevantes e de verificar se ainda se aplicam atualmente. Estes resultados serão reproduzidos quando possível e adaptados para que voltem a funcionar quando for constatado que foram parcialmente corrigidos.

1.2 Objetivos

- Entender se é possível atacar um usuário do Amazon Echo através de uma Alexa Skill.
- Mapear vulnerabilidades presentes nas diversas etapas de uma interação com uma Alexa Skill.
- Mapear comportamentos indesejáveis apresentados pelo Echo Dot 3 que podem ser explorados por um atacante.
- Identificar ataques conhecidos existentes na literatura.
- Elaborar demonstrações *proof-of-concept* que utilizem as informações obtidas.

1.3 Metodologia

A execução do projeto foi dividida em três etapas:

1. Leitura da documentação e de artigos relevantes
2. Mapeamento de fraquezas e vulnerabilidades

3. Elaboração de ataques *proof-of-concept*

1.3.1 Leitura

Foi realizada a leitura de toda a documentação disponível em [2], com foco em encontrar comportamentos inseguros por design. Em seguida, foram utilizadas as palavras-chave "alexa", "skills", "echo", "amazon", "vulnerability", "security"e "attack"em buscadores como o Google Scholar para encontrar artigos relevantes.

1.3.2 Mapeamento

Nesta etapa foi estabelecida uma superfície de ataque baseada no modelo de ameaça utilizado e na documentação encontrada. Cada ponto de interação da superfície teve fraquezas identificadas e classificadas através da metodologia *Root Cause Mapping* com uso da *Research View* disponível em [6].

1.3.3 Elaboração

Por fim, os ataques considerados mais plausíveis encontrados nos trabalhos relacionados foram reproduzidos e tiveram seus usos classificados de acordo com a matriz MITRE ATT&CK.

1.4 Contribuições

Neste projeto foi estabelecida uma superfície de ataque de Alexa Skills que não existia na literatura, bem como um mapeamento de fraquezas para a taxonomia CWE correspondente. Também foram realizados ajustes no fluxo de execução de ataques conhecidos que foram corrigidos pela Amazon, de forma a permitir que continuem causando os efeitos desejados. O código desenvolvido nos experimentos também foi disponibilizado nos apêndices para referência futura.

1.5 Organização do Trabalho

Na seção 2 serão apresentados os conceitos básicos necessários para compreender o resto do trabalho. Na seção 3 serão discutidos os principais trabalhos relacionados e seus resultados. Na seção 4 é realizado um reconhecimento de quais partes do processo de uso de uma skill por um usuário são de interesse para a pesquisa. Na seção 5 serão relatados os experimentos realizados, bem como sua metodologia, ressalvas, limitações e contraste com os resultados relatados na literatura. Por fim, na seção 6 serão resumidos e discutidos os resultados obtidos. Também serão listados possíveis trabalhos futuros relacionados a esta pesquisa.

CAPÍTULO 2

Conceitos Básicos

Neste capítulo serão definidos os conceitos necessários para o entendimento dos capítulos que se seguem. Na primeira seção serão discutidos conceitos relacionados à assistente e aos dispositivos utilizados. A segunda seção define conceitos utilizados em Segurança de Sistemas da Informação.

2.1 Alexa

Assistente de voz desenvolvida pela Amazon. Sua interface com o usuário é normalmente constituída por uma conversa onde o usuário faz uma pergunta ou enuncia um comando e recebe uma resposta em voz alta. O fluxo de comunicação com a Alexa é mediado inteiramente pelo Alexa Service, serviço da *cloud* da Amazon responsável por processar todos os pedidos e encaminhar para os *endpoints* apropriados quando necessário. Uma visão de alto nível deste processo pode ser observada na figura 2.1. O Alexa Service fica responsável por verificar qual usuário fez o pedido e qual dispositivo deste usuário está sendo utilizado, bem como identificar qual serviço[1] está sendo requisitado através do uso de *Natural Language Processing*, criar uma conexão com o *backend* do serviço escolhido e verificar que este *backend* é quem diz ser. Toda essa comunicação é realizada através do envio de *payloads* JSON pelo protocolo HTTPS.

2.1.1 Alexa Skills

A *feature* motivadora deste projeto, funcionando como uma espécie de aplicativo para uso com a Alexa. Permite que terceiros criem funcionalidades para a Alexa, podendo ser utilizada para a integração de um serviço com a assistente. É importante destacar que diferente de um

[1] Normalmente uma Skill

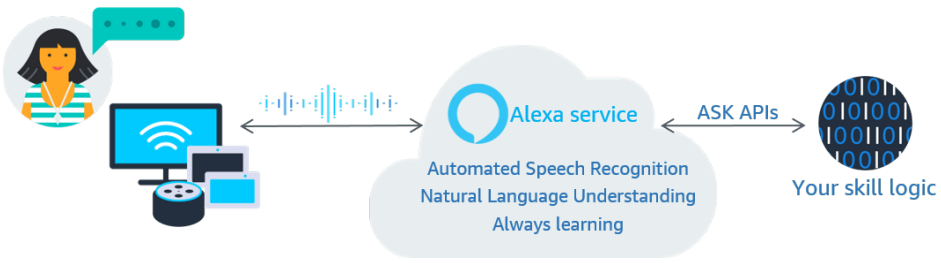

Figura 2.1 *Overview* de uma interação com a Alexa[9]

aplicativo, nada é instalado do lado do cliente, ou seja, o funcionamento de uma interação com uma Skill acontece inteiramente do lado do servidor.

2.1.2 Amazon Echo Dot 3

O terceiro modelo do Amazon Echo Dot, um *speaker bluetooth* desenvolvido especificamente para uso com a Alexa pelos usuários finais. Existem modelos mais recentes, que podem ter comportamentos diferentes. Desta forma, destaca-se que os resultados obtidos valem para o modelo específico utilizado nos testes.

2.1.3 Configuração do ambiente

A configuração do aparelho Echo Dot é feita através do aplicativo Amazon Alexa, disponibilizado para Android. Inicialmente é realizado o login com a conta de testes[2], que também é a conta que será utilizada para o desenvolvimento das *proof-of-concepts*. Em seguida, o aparelho é ligado na tomada e a função *add device* do aplicativo é selecionada, informando-se o tipo do aparelho que se deseja adicionar. Após essa etapa, é necessário utilizar o aplicativo para fornecer credenciais da rede Wi-Fi que será utilizada pelo Echo Dot. A partir deste ponto, o setup foi concluído e é possível pular as perguntas que se seguem.

2.1.4 Ativação de Skills

Antes que um usuário possa utilizar uma Alexa Skill, é necessário que esta seja ativada. Isto pode ser feito manualmente através do *companion app* que controla o Echo, ou através de um comando de voz. A ativação por comando de voz pode acontecer de duas formas: intencionalmente, isto é, o usuário explicitamente fala em voz alta a frase de ativação especificada pela skill; ou de forma inferida, isto é, o usuário realiza um pedido que a Alexa não consegue atender com suas funcionalidades padrão e a Alexa ativa uma skill que ela considera capaz de completar o pedido.

2.1.5 Voice Model

Após a ativação da Skill, um usuário pode interagir com ela através do Voice Model definido pelo desenvolvedor. Este modelo consiste em pelo menos três partes: um Invocation Name, que é o nome utilizado pelo usuário para iniciar a skill; uma ou mais Utterances, que são frases definidas pelo desenvolvedor que serão mapeadas para um Intent; e um ou mais Intents, que são objetos que definem uma intenção do usuário. Certos Intents também podem conter um ou mais Slots, que são informações extras (como um número de telefone ou cor) que serão utilizados no processamento deste Intent. A figura 2.2 demonstra um modelo possível.

Nota-se que embora existam Slots padrão (cor, número, etc) os desenvolvedores também pode criar Slots customizados. Estes Slots customizados recebem uma lista de palavras que podem ser utilizadas por um usuário neste Slot e a utilizam para treinar um modelo de reconhecimento de voz específico para ele.

[2]Criada através do Amazon Alexa Developer Console

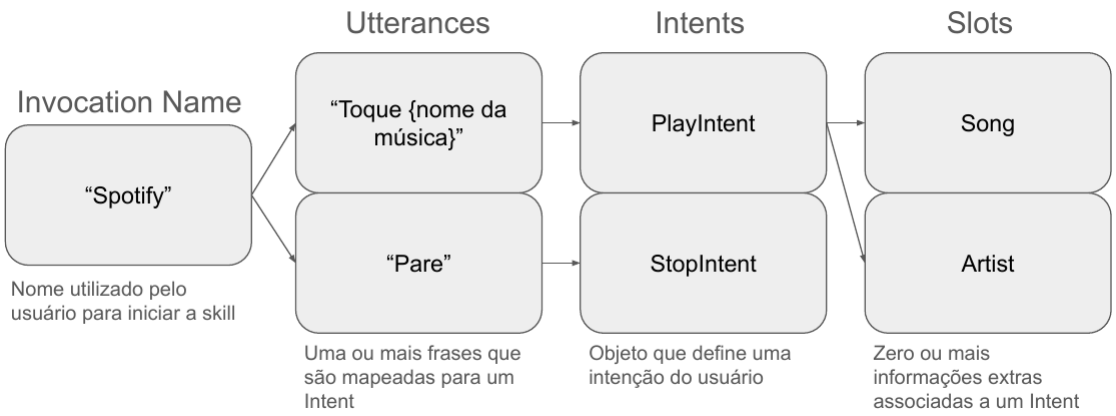

Figura 2.2 Um exemplo de Voice Model simplificado que poderia ser implementado para uma Skill de reprodução de áudio

2.1.6 Interação com uma Skill

Com essas definições, é possível estabelecer o passo a passo de uma interação com uma skill:

1. o usuário faz um pedido em voz alta utilizando a *wake-up word* ("Alexa", por padrão). Exemplo: "Alexa, abra o Spotify e toque Idol por Yoasobi";

2. o Echo Dot reconhece a *wake-up word*, grava até que o usuário pare de falar e envia o resultado para o Alexa Service, nos servidores da Amazon. O arquivo de áudio é então processado por um algoritmo de Automated Speech Recognition, retornando uma transcrição que é enviada para um algoritmo de Natural Language Understanding;

3. o NLU reconhece a skill solicitada através do Invocation Name utilizado (neste caso, “Spotify”) e utiliza o voice model previamente definido para ligar a Utterance “toque Idol por Yoasobi” ao Intent “PlaySong”. A definição deste Intent inclui os Slots “Song” e “Artist”, que são preenchidos com “Idol” e “Yoasobi” respectivamente;

4. o objeto Intent gerado pelo passo anterior é encapsulado em um JSON que é enviado em uma requisição POST HTTP para o backend especificado pelo desenvolvedor (normalmente uma função lambda da AWS);

5. o backend responde com um JSON que contém o objeto “response”, que é composto de diversos parâmetros que indicam o que o Echo deve responder ao usuário. Neste exemplo, é utilizada a funcionalidade “outputSpeech”, que vai ler em voz alta um texto do tipo “plainText”.

6. o Alexa Service envia a resposta que vai ser enunciada ao usuário pelo Echo e o arquivo de áudio que deve ser reproduzido.

7. o Echo Dot fala em voz alta “Tocando Idol por Yoasobi” e dá início a reprodução da música.

Um diagrama desta interação pode ser observado na figura 2.3

2.2 Segurança

A segurança de informação pode ser discutida sob diversas óticas. Neste trabalho são de particular interesse os chamados pilares da segurança da informação: confidencialidade, integridade e disponibilidade. Também é importante definir conceitos relacionados a taxonomia, como fraqueza, vulnerabilidade e ataque. Ainda neste contexto, também é necessário tomar conhecimento das ferramentas de taxonomia que existem na indústria atualmente.

2.2.1 Confidencialidade

Característica de segurança que garante que apenas quem é autorizado tem acesso a uma determinada informação. Normalmente protegida através do uso de encriptação e controle de acesso.

2.2.2 Integridade

Característica de segurança que garante que uma determinada informação é confiável e não foi modificada por terceiros. Normalmente protegida através do uso de assinaturas ou certificados digitais.

2.2.3 Disponibilidade

Característica de segurança que garante que um serviço ou informação pode ser acessado quando necessário. Normalmente protegida através de redundâncias.

2.2.4 Fraqueza

"Uma condição em um *software*, *firmware*, *hardware*, ou componente de serviço que, dentro de certas circumstâncias, pode contribuir para a introdução de vulnerabilidades."[5]. A fundação OWASP disponibiliza uma lista das dez fraquezas consideradas mais perigosas pela organização em 2021[8]:

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components

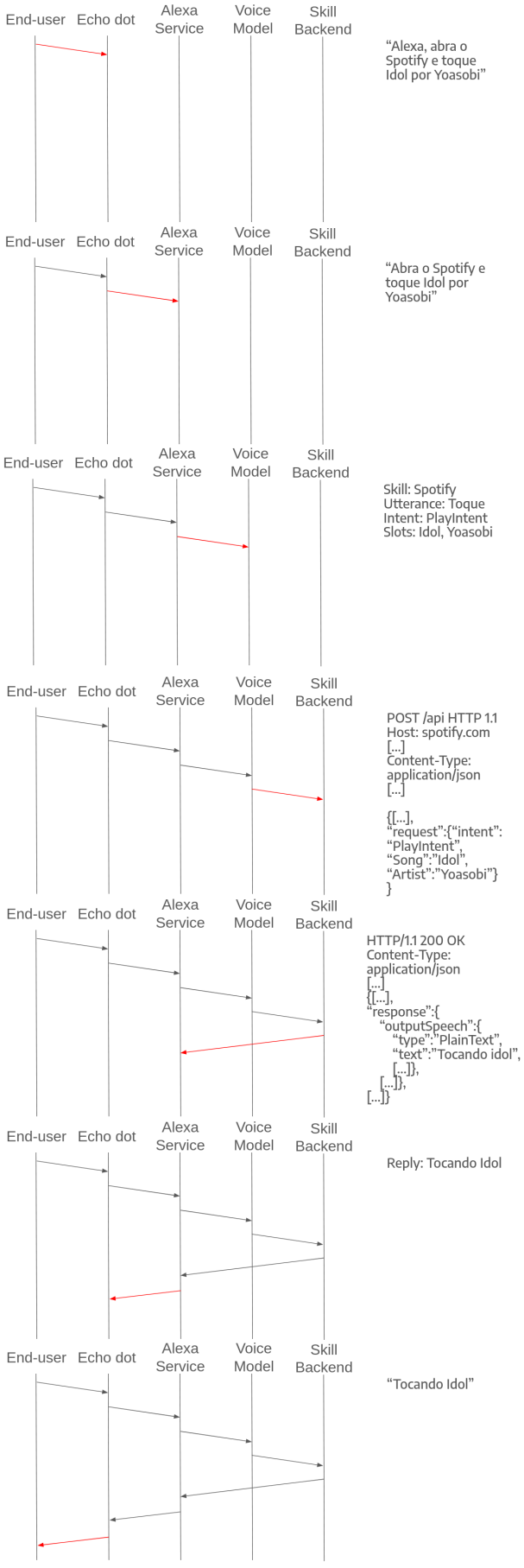

Figura 2.3 Diagrama da interação com uma skill

7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

As fraquezas exploradas neste trabalho se encaixam nas categorias Insecure Design e Injection.

2.2.5 Vulnerabilidade

"Uma falha em um *software*, *firmware*, *hardware*, ou componente de serviço resultando de uma ou mais fraquezas que podem ser exploradas, causando um impacto negativo a confidencialidade, integridade ou disponibilidade de um ou mais componentes impactados."[5] Vulnerabilidades concretas estão vinculadas a um *software*, *firmware*, *hardware*, ou componente de serviço em particular e são registradas como CVEs[3] por entidades conhecidas como CVE Numbering Authorities. Um exemplo de alto impacto é o CVE-2014-0160 Falha na implementação TLS do OpenSSL AKA Heartbleed.

2.2.6 Common Weakness Enumeration

Taxonomia criada pela ONG estadunidense MITRE com o objetivo de enumerar e classificar fraquezas de sistemas da informação[1]. As fraquezas listadas são classificadas a partir de sua forma mais geral, passando por níveis de especialização, até chegar a uma forma final específica (Por exemplo, Improper Access Control - (284) > Improper Authentication - (287) > Weak Authentication - (1390) > Use of Weak Credentials - (1391) > Use of Default Credentials - (1392) > Use of Default Password - (1393)). A figura 2.4 ilustra essa classificação.

2.2.7 Ataque

Uma exploração encadeada de uma ou mais vulnerabilidades, que permite que um atacante cause um determinado efeito negativo sobre a confidencialidade, integridade e/ou disponibilidade de um serviço ou informação.

2.2.8 MITRE ATTI&CK

Matriz de técnicas e passos utilizados na execução de ataques como definidos pela ONG MITRE[7]. Nesta metodologia, um ataque em específico é constituído por um subconjunto dos passos listados, não necessariamente na ordem listada. Os passos definidos pela metodologia são:

1. Reconhecimento: Processo de coleta de informações relevantes sobre a vítima, como *hosts*, redes e identidades.

[3]Common Vulnerabilities and Exposures

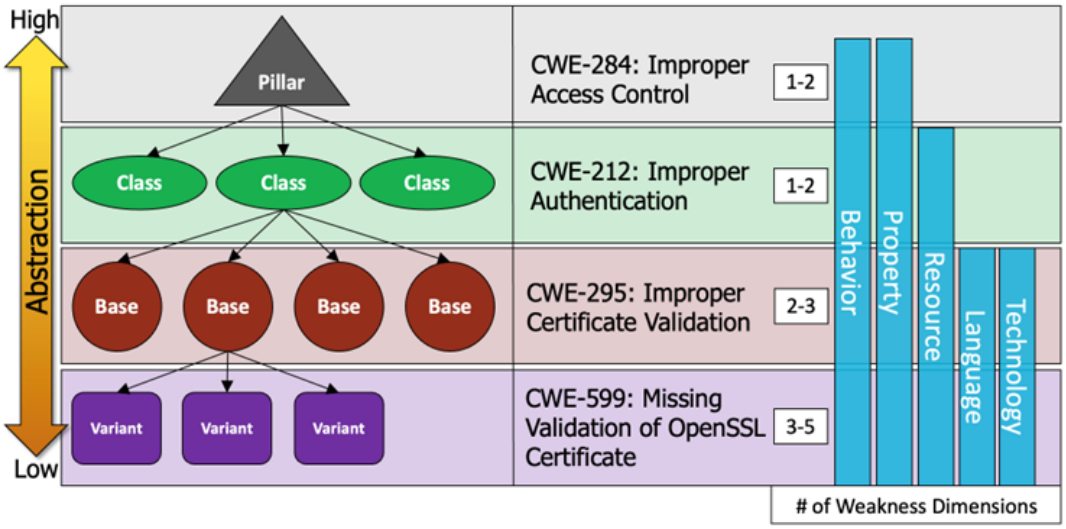

Figura 2.4 Hierarquia de classificação CWE[6]

2. Desenvolvimento de recursos: Etapa de preparação das capacidades do atacante, como o comprometimento de contas e desenvolvimento de ferramentas.

3. Acesso inicial: Momento de contato inicial com a vítima, normalmente entregando algum tipo de *payload*, como injeção de conteúdo, *phishing* e comprometimento de *supply chain*.

4. Execução: Processo de execução do *payload* escolhido, como execução através de usuário e uso de interpretadores de comandos e *scripts*.

5. Persistência: Etapa onde o atacante busca garantir que seu acesso é capaz de persistir na infraestrutura da vítima, resistindo a *reboots* e tentativas de remoção, como a modificação de processos do sistema, o comprometimento de binários e o uso de *scripts* de *boot*.

6. Escalação de privilégios: Processo de aumento das capacidades que o atacante possui dentro da infraestrutura, como a manipulação de *tokens* de acesso, injeção de processos e o sequestro de fluxos de execução.

7. Evasão de defesas: Mecanismos utilizados durante o ataque para evitar detecção e adiar remoção, como a ofuscação de arquivos e a modificação da identificação de processos.

8. Acesso a credenciais: Processo de obtenção de credenciais que permitam maior acesso aos ativos da vítima, como o uso de força bruta, *adversary-in-the-middle* e a captura de *inputs*.

9. Descoberta: Processo de mapeamento do acesso obtido por etapas anteriores do ataque, como a descoberta de contas, processos, usuários e arquivos.

10. Movimento lateral: Processo de comprometimento de ativos que não necessariamente tem mais acesso, porém dificultam a remoção do acesso que o atacante tem, como o *spearphishing* interno e modificação de arquivos internos

11. Coleta: Etapa em que o atacante obtém os dados que são de seu interesse, como o uso da captura de *inputs*, coleta de dados locais e coleta de e-mails.

12. Comando e controle: Técnicas utilizadas para exercitar os privilégios conferidos aos ativos sob controle do atacante, como o uso de protocolos de tunelamento, programas de acesso remoto e injeção de conteúdo.

13. Exfiltração: Momento em que o atacante extrai as informações coletadas para uma rede sob seu controle, como o uso de serviços *web*, o uso de mídia física e o uso de canais de comando e controle.

14. Impacto: Processo que o atacante utiliza para gerar um efeito negativo desejado, como a encriptação de dados, a negação de serviço e a manipulação de dados.

CAPÍTULO 3

Trabalhos Relacionados

Neste capítulo serão explorados os trabalhos que influenciaram a pesquisa e os experimentos realizados.

3.1 Funcionamento do ecossistema Alexa

Em Su et al [15] os autores descrevem o funcionamento do ecossistema, modelando os componentes que o constituem: usuário, dispositivo, Alexa Voice Service e Skils. Os autores também descrevem o processo de desenvolvimento de Skills; o processo de certificação destas; realizam uma comparação entre Alexa Skills e aplicativos Android; e estabelecem o modelo adversarial que é utilizado neste trabalho: Os atacantes não tem acesso a *cloud* da Alexa ou aos dispositivos Alexa dos usuários e as vítimas são os usuários de Skills Alexa. Neste modelo, os autores também modelam como desenvolvedores de skills exploram três categorias de vulnerabilidades que afetam a privacidade dos usuários: *over-privileged resource access* (Skills que requerem mais permissões do que o necessário), *hidden code-manipulation* (Modificação do código *backend* de uma Skill após a publicação) e *hidden content-manipulation* (Modificação do conteúdo entregue pela Skill após a publicação). Através destas, é possível evadir o processo de certificação da Amazon e obter informações sensíveis de usuários.

Em Kim et al [11] os autores descrevem o funcionamento do algoritmo utilizado para *Natural Language Understanding* pelo Alexa Service para entender Utterances e realizar Name-Free Invocation.

3.2 Ataques e vulnerabilidades

Em Lentzsch et al [12], os autores realizam uma avaliação de mais de noventa mil skills, revelando algumas das técnicas utilizadas neste trabalho, como o bypass da certificação da Amazon. Este artigo também realiza uma avaliação da efetividade de algums dos ataques reproduzidos nos experimentos deste trabalho, incluindo o Skill Squatting e o bypass da API de dados sensíveis.

Em Esposito et al [10], os autores estabelecem o funcionamento de dois dos ataques reproduzidos neste trabalho: Alexa vs Alexa e Mask Attack, bem como duas vulnerabilidades utilizadas na implementação destes: Full-Volume Vulnerability e Break Tag Chain. Os autores também avaliam a efetividade destes ataques através de testes contra usuários reais. As vulnerabilidades encontradas foram reportadas a Amazon, que as classificou como de severidade média.

Em Sabir et al [14], os autores avaliam o grau de confiança que usuários tem na Alexa e quão bem eles sabem quem desenvolveu a skill que estão usando e se sequer sabem qual skill estão usando. Seus resultados indicam que a maioria dos usuários falham em ambas as categorias.

3.3 Seleção de pontos de interesse para os experimentos

Neste trabalho serão exploradas as cadeias de ataque que fizerem uso de vulnerabilidades baseadas em fraquezas inerentes as decisões de *design* tomadas pela arquitetura de uso de Skills no ambiente Alexa. Estas cadeias de ataque incluem Skill Squatting[12], Alexa vs Alexa[10], Mask Attack[10] e o Bypass da API de informações sensíveis[12]. Considerou-se que, por serem inerentes a arquitetura, a chance de que essas fraquezas não tenham sido corrigidas desde a publicação dos artigos relevantes é alta. Estas fraquezas serão detalhadas no capítulo 4.

CAPÍTULO 4

Superfície de Ataque

Neste capítulo pretende-se elaborar uma lista de pontos que podem ser controlados por um atacante e de comportamentos pontencialmente perigosos existentes nestes pontos[1]. Desta forma é possível criar vulnerabilidades e entender como e onde tentar realizar ataques. Assim, esta seção contém os pontos de interesse encontrados ao examinar a documentação existente [2] e o Echo Dot utilizado nos testes, separados por momentos em que um atacante é capaz de interagir com o fluxo de processamento, ordenados de acordo com as etapas da interação do usuário como visualizada na figura 2.3.

4.1 Conexão Bluetooth

Num momento pré-interação com o dispositivo, podem ser encontradas fraquezas dentro da classificação CWE-306 *Missing Authentication for Critical Function*. O modo de pareamento do dispositivo pode ser ativado por voz ("Alexa, bluetooth" já é suficiente) e não realiza identificação antes de parear, permitindo que um atacante com acesso físico a algum ambiente próximo ao dispositivo se conecte e assuma o controle do speaker. Além disso, se o bluetooth já tiver sido ativado previamente e não foi desativado, é possível conectar sem pedir nenhuma permissão. Em uma exploração utilizando esta fraqueza, um atacante poderia, por exemplo, gritar "Alexa, bluetooth" através de uma janela aberta e obter controle remoto do speaker, criando uma vulnerabilidade para utilizar como parte de uma cadeia de ataque.

4.2 Reconhecimento de comandos de voz

Nas etapas de interação do usuário com o Echo Dot e do Echo Dot com o Alexa Service, podem ser encontradas fraquezas com as classificações CWE-862 *Missing Authorization* e CWE-116 *Improper Encoding or Escaping of Output*. Embora o dispositivo seja capaz de reconhecer se a voz de quem está falando pertence a algum usuário previamente cadastrado, o comportamento padrão permite que qualquer fala registrada pelo dispositivo seja entendida como um comando, incluindo terceiros que nunca foram encontrados pelo dispositivo antes[2]. Além disso, o dispositivo reconhece comandos enviados através de seu próprio speaker como comandos legítimos [10]. Isso permite que um atacante que obteve controle do speaker através da exploração de outra fraqueza envie comandos de voz que serão executados pelo dispositivo como parte de uma cadeia de ataque.

[1] fraquezas de *Injection* e *Insecure Design*

4.3 Ativação da Skill

O passo de ativação da skill ocorre durante a comunicação do Alexa Service com o Voice Model da Skill e é crucial para qualquer ataque capaz de fazer mais que apenas controlar o speaker. Um atacante interessado em fazer com que um usuário alvo ative uma skill em particular pode usar uma combinação de alguns dos seguintes comportamentos:

1. o dispositivo realiza a ativação de qualquer skill cuja frase de ativação foi enunciada, sem confirmar com o usuário que essa realmente era sua intenção. Desta forma, é possível criar uma skill com uma frase de ativação comumente utilizada no cotidiano (Por exemplo: "Bom dia"); (CWE-115 *Misinterpretation of Input*)

2. o Alexa Service realiza a ativação automática de skills que retornem uma resposta com um objeto CanFulfillIntent positivo quando um usuário faz um pedido sem utilizar uma frase de ativação que especifique uma skill em particular. Esse comportamento permite que um atacante crie uma skill que sempre retorna CanFullfillIntent = true, aumentando as chances de que sua skill seja executada automaticamente (Ainda é necessário competir com outras skills legítimas que também retornarem true); (CWE-1038 *Insecure Automated Optimizations*)

3. o ecossistema de desenvolvimento de skills permite que sejam criadas skills com frases de ativação com pronúncia idêntica a de skills já existentes. Além disso, o algoritmo não prioriza skills mais antigas ou que já foram ativadas previamente no momento em que decide qual skill o usuário gostaria de ativar; (CWE-1039 *Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations*)

4. falhas na API da Amazon já permitiram que um atacante ativasse skills arbitrárias, bastando que um usuário clicasse em um link malicioso[4]. (CWE-79 *Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')*)

4.4 Execução da Skill

A execução de uma skill ocorre durante a etapa de comunicação do Voice Model com o *backend* da Skill e pode ser realizada em um servidor sob completo controle de seus desenvolvedores, de maneira que é impossível garantir que o comportamento da skill não foi modificado depois do processo de publicação e certificação[15] (CWE-358 *Improperly Implemented Security Check for Standard*). Além disso, mesmo quando utilizados os serviços de host da Amazon, desenvolvedores podem facilmente contornar o uso das APIs que a Amazon disponibiliza para coleta de informações sensíveis dos usuários (como endereço e telefone), obtendo a informação de forma direta[12] (CWE-424 *Improper Protection of Alternate Path)*. Por fim, durante os experimentos realizados, o dispositivo não alertou o usuário de nenhuma forma, mesmo quando a resposta da Skill continha *tags* <*break/*> que totalizavam mais de nove minutos de silêncio. Desta forma, uma Skill pode permanecer em execução sem que o usuário esteja ciente deste fato (CWE-150 *Improper Neutralization of Escape, Meta, or Control Sequences*).

4.5 Resposta ao Usuário

O sistema de envio de respostas ao usuário utilizado na comunicação do *backend* da Skill com o usuário final permite o envio de respostas arbitrárias, e mais especificamente, permite o envio de respostas vazias (caracteres que geram um longo período de silêncio, como a diretiva <break/>) ou até mesmo de arquivos de áudio arbitrários que serão tocados pelo speaker[10], criando a fraqueza discutida na seção anterior.

Por fim, [10] menciona o uso de uma vulnerabilidade nomeada "Full-volume vulnerability", ou FVV para garantir que um áudio continue tocando em volume máximo mesmo após a *wake-up word* ser identificada (normalmente o volume fica praticamente inaudível). A vulnerabilidade consiste em enviar um comando "Alexa, turn off"no início do áudio, fazendo com que o dispositivo acorde, diminua o volume (como era esperado) processe o comando e volte a dormir imediatamente. Após esse passo, o dispositivo aparenta esquecer que há um arquivo de áudio sendo reproduzido e o volume retorna ao nível anterior.

Um atacante pode utilizar estes comportamentos para fazer com que usuários acreditem que a interação com a skill acabou, quando na verdade a escuta continua ativa, ou para fazer com que usuários acreditem que estão falando com a Alexa, quando na verdade estão interagindo com uma skill.

4.6 Exploração de fraquezas nos experimentos

No capítulo 5, serão exploradas as fraquezas de ativação de Skill no ataque Skill Squatting (especificamente os CWEs 115 e 1039), as fraquezas da conexão bluetooth e do reconhecimento de voz (CWEs 306, 862, 116 e vulnerabilidade FVV) no ataque Alexa vs Alexa, as fraquezas de execução da Skill e resposta ao usuário (CWE-150 e vulnerabilidade FVV) no ataque Mask Attack e a fraqueza de execução da Skill (CWE-424) no bypass da API de informações sensíveis. Além disso, todos os ataques dependem da fraqueza de execução da skill identificada como CWE-358 para serem executados.

CAPÍTULO 5

Experimentos e Análise

Neste capítulo serão executados experimentos projetados para reproduzir os ataques que foram considerados como tendo alta chance de sucesso. Cada um destes ataques executa papéis diferentes na metodologia estabelecida no MITRE ATT&CK e podem ser usados de forma independente ou em cadeia dependendo dos objetivos do atacante.

5.1 Ambiente de testes

Os experimentos a seguir foram realizados em um Echo Dot 3, com a versão de software mais recente disponível em Outubro de 2023. Para a realização dos experimentos, foi necessária a criação de uma conta de desenvolvedor Alexa no Developer Console da Amazon. Todas as skills maliciosas desenvolvidas foram utilizadas em modo de teste, de forma que apenas a conta de desenvolvedor associada pode ativá-las e invocá-las. Foi utilizado o aplicativo para Android Amazon Alexa, disponível na Play Store, para configurar e controlar o Echo Dot quando necessário. O código de backend criado para os experimentos está disponível no Apêndice A.

5.2 Skill Squatting

Neste experimento, foi utilizado o ataque elaborado por [12]. No ataque, o atacante utiliza a ativação automática de skills e a possibilidade de utilizar frases de ativação com colisões fonéticas para executar uma skill com funcionalidade arbitrária. Este ataque se enquadra como uma técnica de Acesso Inicial e Execução na metodologia MITRE ATT&CK. O procedimento experimental foi o seguinte:

1. o dispositivo de testes ativa a skill de notícias G1 na loja;
2. na conta de desenvolvimento utilizada pelo atacante, é criada uma skill com funcionalidade arbitrária. Neste caso, a skill informa não ser a skill solicitada, ao ser executada;
3. o Invocation Name da skill criada é definido com "g. um"(a pontuação indica que "g"deve ser pronunciada como a letra);
4. após confirmar que a skill G1 funciona da forma esperada no Echo Dot, a skill maliciosa é publicada. Esta confirmação foi realizada através da frase "Alexa, abra o G1";
5. o usuário de testes enuncia "Alexa, abra o G1";

6. a skill maliciosa é ativada e executada, informando em voz alta não ser a skill G1;

O algoritmo utilizado para decidir qual skill deve ser ativada quando há colisão de Invocation Name não é disponibilizado pela Amazon. No entanto, [12] nota que, de acordo com os experimentos realizados, o algoritmo tende a priorizar skills cadastradas mais recentemente, aumentando as chances de um ataque ser bem-sucedido. Desta forma, o comportamento observado neste experimento condiz com o que era esperado pela literatura.

Uma gravação da realização deste experimento está disponível em `https://drive.google.com/file/d/1vXYkPVqzMtKan-_Z1zwCEHHHhuKFXqF6/view?usp=drive_link` (Comportamento normal da skill alvo) e `https://drive.google.com/file/d/1hnrqalhdeDoRWFciCM84b8ODnVkHJJzu/view?usp=drive_link` (Realização do ataque).

5.3 Alexa vs Alexa

Neste experimento, foi utilizado o ataque elaborado por [10]. Este ataque utiliza a fraqueza que permite que o Echo interprete comandos enviados através do próprio speaker combinada com a *full-volume vulnerability* para controlar o dispositivo remotamente. Este ataque se enquadra como uma técnica de Acesso Inicial e Execução na metodologia MITRE ATT&CK. O procedimento experimental foi o seguinte:

1. foi preparado um arquivo .mp3 utilizando um serviço de *text-to-speech* para a seguinte frase: "Alexa, desligue. Espaçamento. Alexa, que horas são?";
2. em seguida, o atacante se conecta no Echo através do bluetooth;
3. utilizando a conexão bluetooth, o arquivo de áudio preparado previamente é reproduzido pelo Echo;
4. o dispositivo executa o comando arbitrário, retornando o horário local.

Nota-se que o ataque original discutido em [10] foi realizado em inglês. No artigo, o autor nota que foi possível realizar o ataque mesmo sem fazer uso da *full-volume vulnerability* e que esta apenas aumenta a chance de sucesso do ataque.

O resultado experimental obtido para o ataque em português indica que não é possível realizar o ataque sem o uso da FVV, uma vez que a redução de volume após a *wake-up word* ser reconhecida impede que o comando seja processado. Nota-se também que mesmo com o uso da FVV, foram necessárias diversas tentativas de reprodução do áudio até que o comando fosse reconhecido. Desta forma, o ataque é muito menos confiável neste idioma.

O ataque original também menciona que é possível utilizar uma Skill de rádio maliciosa para realizar a reprodução do áudio, porém não é possível utilizar FVV nesta modalidade do ataque (junto com outros *tradeoffs*). Desta forma, reproduzir esta modalidade do ataque ficou fora do escopo deste projeto.

Uma gravação da realização deste experimento está disponível em `https://drive.google.com/file/d/1j9NGc1WVt_bZVkhG5-Nk1wIS9vc8qKno/view?usp=drive_link`

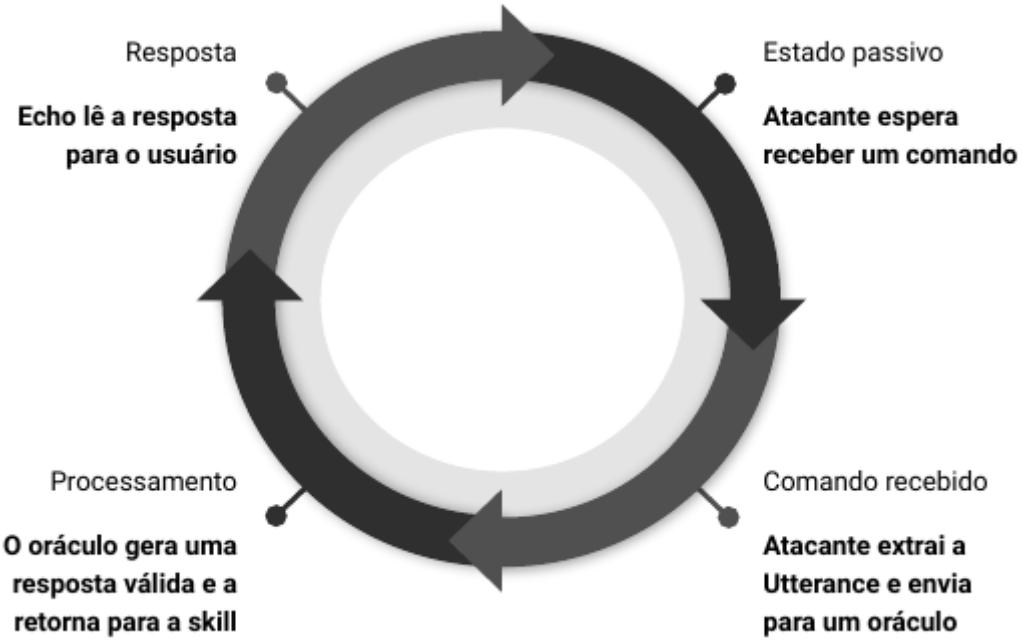

Figura 5.1 Ciclo de execução de um Mask Attack

5.4 Mask Attack

Neste experimento, foi utilizado o ataque elaborado por [10]. Este ataque utiliza o controle que o atacante tem sobre a resposta enviada ao usuário para criar a ilusão de que o usuário alvo está se comunicando com a Alexa, quando na verdade está interagindo com uma skill maliciosa. Um resumo geral da execução do ataque pode ser observado na Figura 5.1. Este ataque se enquadra como uma técnica de Evasão de Defesa, Acesso a Credenciais, Coleta, Comando e Controle e Exfiltração na metodologia MITRE ATT&CK. O procedimento experimental foi o seguinte:

1. foi criada uma skill na conta de desenvolvimento com um Invocation Name arbitrário (neste caso, "cubo mágico").

2. é criado um Intent chamado "InterceptIntent"com apenas uma Utterance. Esta Utterance é preenchida exclusivamente com Slots customizados nomeados "CatchAll". Cada um destes Slots é treinado com uma string alfanumérica aleatória;

3. a skill criada é executada;

4. o *backend* recebe o LaunchIntent e responde com a quantidade máxima da tag SSML <break/> permitida pela API da Alexa;

5. a reprodução da resposta recebida reproduz as tags <break/> como silêncio, de forma a passar a impressão de que nenhuma skill está sendo executada;

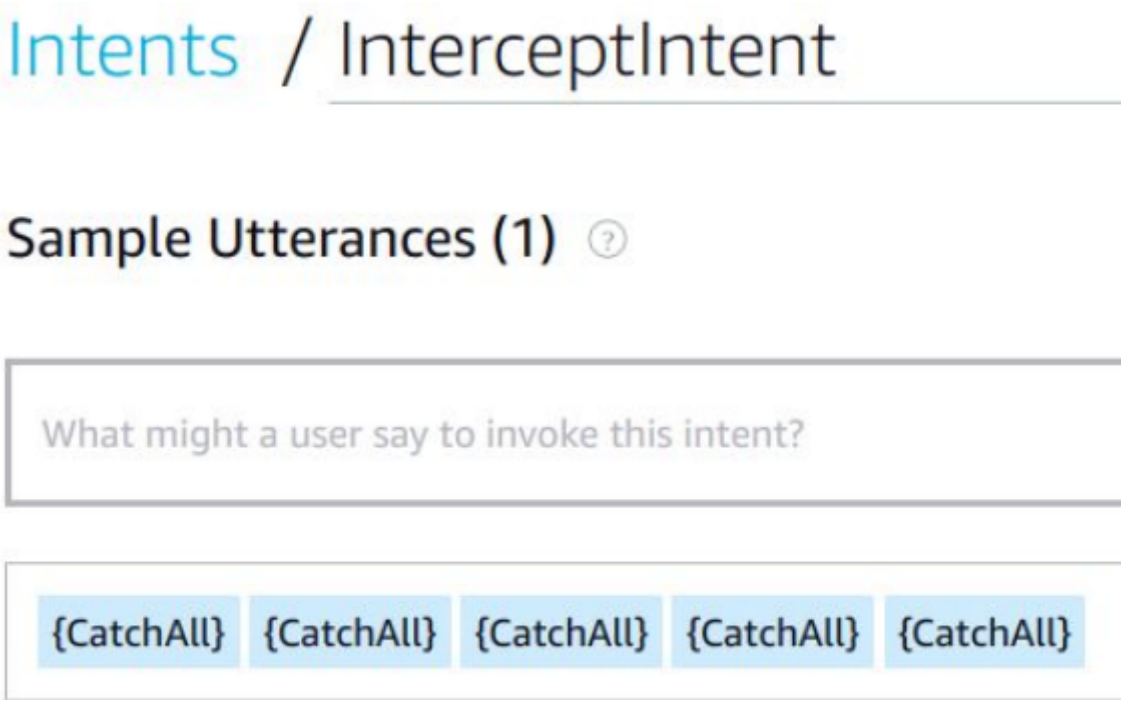

Figura 5.2 Configuração do Intent utilizado no experimento.

6. o usuário de testes realiza uma interação com o Echo: "Alexa, como está o clima em Recife?";
7. a skill recebe cada palavra como um Slot CatchAll preenchido em InterceptIntent;
8. o *backend* da skill passa a interação do usuário para um oráculo e retorna a resposta recebida;
9. após o fim da resposta, a skill reproduz uma nova resposta preenchida com a quantidade máxima permitida de tags <break/>.

Nota-se que [10] menciona o uso de valores alfanuméricos para treinamento dos Slots. Neste experimento, foi testado também o uso de palavras aleatórias, strings vazias e sequências aleatórias de caracteres, obtendo o melhor resultado com strings alfanuméricas. A quantidade de Slots utilizada neste experimento foi maior que a utilizada originalmente, pois o uso de um único Slot sugerido pelos autores não permitia obter todas as palavras enunciadas pelo usuário de testes.

No ataque original, são utilizadas 400 tags <break time="10s"/>, totalizando mais de uma hora de silêncio e sendo limitadas apenas pelo número máximo de caracteres permitidos em uma resposta (8000). No entanto, nos experimentos realizados, verificou-se que o uso de mais de uma tag <break time="10s"/> fazia a resposta ser rejeitada pela API da Amazon. Desta forma, foram utilizadas apenas uma tag <break time="10s"/> seguida de repetidas tags <break/>. Verificou-se também que a resposta era rejeitada antes de atingir o limite de 8000 caracteres. A quantidade máxima de tags encontrada no final dos experimentos pode ser verificada no

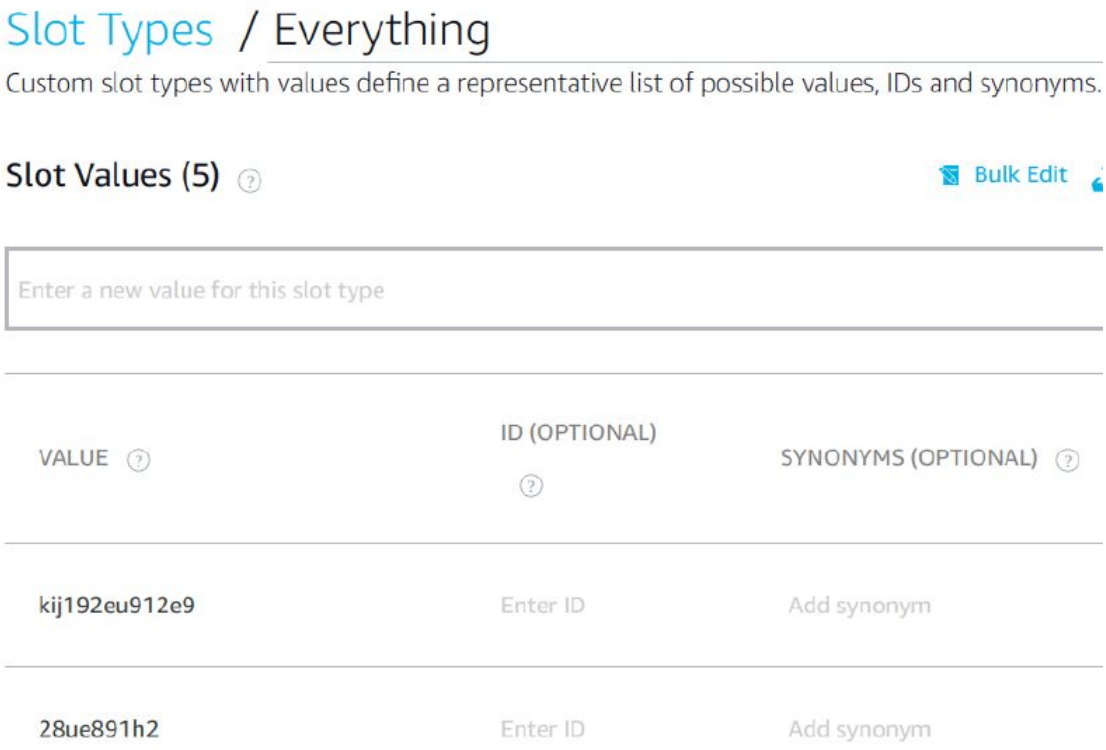

Figura 5.3 Configuração utilizada para treinamento do modelo de reconhecimento de voz do Slot CatchAll

Apêndice A, totalizando pouco menos de 10 minutos de silêncio. Ainda assim, os resultados obtidos violam a documentação da Amazon[2], que indica que uma resposta com mais de 10 segundos de silêncio é rejeitada.

Detalha-se ainda que a implementação do oráculo utilizada no experimento funcionava como um *mock* da API da Alexa, retornando apenas uma resposta *hard-coded* da pergunta de teste. Uma implementação mais completa enviaria a Utterance obtida do usuário para a API real da Alexa e receberia uma resposta válida, retornando esta para o usuário. Além disso, um atacante real poderia implementar um oráculo que recebe a Utterance e retorna uma resposta escolhida pelo atacante, incluindo respostas que induziriam o usuário a divulgar informações confidenciais como senhas e PINs.

Uma gravação da realização deste experimento está disponível em `https://drive.google.com/file/d/12rRUYQtAfvoIqEWA7gcLISi0YLjQJpma/view?usp=drive_link`

5.5 Bypass da API de informações sensíveis

Um dos problemas mencionados em [12] como sendo explorado por desenvolvedores de skills reais é a possibilidade de obter informações pessoais de usuários sem o uso da API de informações sensíveis apropriada. Normalmente, é necessário fazer uma chamada a esta API para obter consentimento do usuário (através de uma confirmação no aplicativo) quando é necessário obter informações consideradas sensíveis como Nome, E-mail, Endereço, Telefone, etc. Para confirmar essa possibilidade, foi criada uma skill simples de *phishing*, que pergunta a informação diretamente ao usuário. Este ataque se enquadra como uma técnica de Reconhecimento, Coleta e Exfiltração na metodologia MITRE ATT&CK. O procedimento experimental foi o seguinte:

1. foi criada uma skill maliciosa, com Invocation Name "Pescaria teste", que solicita o número de telefone do usuário para continuar seu uso;
2. em seguida, o usuário de testes invoca a Skill e escuta o *prompt* do *phishing*;
3. por fim, o usuário de testes entrega a informação solicitada, que é armazenada no *backend*.

Nota-se que inúmeras outras formas de obter informações sensíveis são listadas em [12], como solicitar que o usuário adicione seu endereço a uma lista Alexa criada pela Skill (Skills só tem acesso as listas criadas por elas mesmas) ou redirecionar o usuário através do aplicativo para uma página web que pede seu nome para realizar um cadastro e seu e-mail para enviar um código.

Uma gravação da realização deste experimento está disponível em `https://drive.google.com/file/d/1Zjnp75D9JrNh-_0vpjLLUI6U4fxiTfF7/view?usp=drive_link`

	Fraquezas utilizadas	Alteração realizada	MITRE ATT&CK
Skill Squatting	CWE-115, CWE-1039, CWE-358	N/A	Acesso Inicial e Execução
Alexa vs Alexa	CWE-116, CWE-306, CWE-862, FVV, CWE-358	Uso do português como o idioma do ataque.	Acesso Inicial e Execução
Mask Attack	CWE-150, FVV, CWE-358	Mais Slots. Modificação na forma como o silêncio é gerado na resposta.	Evasão de Defesa, Acesso a Credenciais, Coleta, Comando e Controle e Exfiltração
Bypass da API	CWE-424, CWE-358	N/A	Coleta e Exfiltração

Tabela 5.1 Resultados obtidos para cada ataque

5.6 Resultados

Um breve resumo dos resultados obtidos nos experimentos pode ser observado na tabela 5.1

5.7 Limitações

Todos os experimentos foram realizados em um ambiente de testes e portanto as skills criadas não foram submetidas ao processo de verificação de código realizado durante o processo de publicação. No entanto, o impacto desta verificação é limitado, tendo em vista que skills publicadas podem escolher rodar em um *backend* privado, que pode ser modificado após a publicação para alterar o comportamento da skill da forma que for necessária para permitir o funcionamento desejado. Lentzsch et al [12] elabora um diagrama contendo um passo a passo para evitar a detecção durante este processo.

CAPÍTULO 6

Conclusão e Trabalhos Futuros

Neste trabalho demonstrou-se que é possível utilizar uma Alexa Skill para atacar um usuário. Foram encontrados diversos ataques ainda funcionais e que usam técnicas que estão em constante evolução. Além disso estes ataques são facilitados pelo grau de confiança implícita que os usuários tem na Alexa[14]. Não foram encontrados ataques capazes de tomar controle do hardware do Echo Dot 3, apenas do seu fluxo de execução de Skills. Também foi elaborada uma lista de fraquezas e vulnerabilidades na interação com Alexa Skills e comportamentos indesejados do Echo Dot que permitem que atacantes ataquem usuários

Foram identificados ataques existentes na literatura e foram elaboradas demonstrações funcionais destes com análises comentadas. O ataque Mask Attack foi atualizado, para que voltasse a funcionar de acordo com as limitações existentes atualmente na API da Alexa. Nos experimentos realizados foram obtidos os seguintes resultados:

1. Skill Squatting: oferece uma porta de entrada para quem um atacante inicie sua cadeia de ataque através da execução de uma skill controlada por ele.

2. Alexa vs Alexa: permite que um atacante execute qualquer comando que usuário poderia realizar, incluindo invocar outras skills, controlar aplicações de *smart home* (abrir portas, apagar luzes, desligar sensores e alarmes), fazer chamadas, comprar na Amazon, etc.

3. Mask Attack: permite capturar comandos de voz do usuário, obter senhas e PINs, capturar dados pessoais, modificar respostas e inferir hábitos do usuário (exemplo: horários que está fora de casa).

4. Bypass da API de informações sensíveis: permite violar políticas da Amazon que lidam com informações pessoais dos usuários.

Pesquisas com potencial de aprofundamento nesta área incluem explorar ataques que façam uso da funcionalidade CanFulfillIntent para se aproveitar da Name-free Interaction. Também é possível explorar ataques capazes de aumentar a visibilidade de determinado produto em algoritmos de ranking ou explorar ataques que utilizem cards Alexa para modelos Echo com tela. Por fim, uma ideia mais complexa seria explorar uma cadeia de ataque completa. Começando por realizar a publicação na loja, encadeando múltiplos ataques em sequência e implementando a funcionalidade completa das APIs da Amazon nos oráculos. Desta forma, seria criada uma demonstração partindo de zero acesso até controle total do Echo.

APÊNDICE A

Código utilizado nos experimentos realizados

A.1 Skill Squatting

O código utilizado no *backend* é o arquivo lambda_function.py padrão da skill de exemplo, com exceção da classe LaunchRequestHandler, que foi utilizada como se segue:

```
class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool

        return ask_utils.is_request_type("LaunchRequest")(
            handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Ataque realizado com sucesso, esta não
            é a skill g1"

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
                .response
        )
```

A.2 Mask Attack

O código utilizado no *backend* é o arquivo lambda_function.py padrão da skill de exemplo, com exceção da classe LaunchRequestHandler, que foi modificada, e da classe InterceptIntentHandler, que foi adicionada.

```
    class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
```

```
        return ask_utils.is_request_type("LaunchRequest")(
            handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = '.<break time="10s"/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
            />.<break/>.<break/>.<break/>.<break/>.<break/>.<
            break/>.<break/>.<break/>.<break/>.<break/>.<break
```

/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<break/>.<

break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break
/>.<break/>.<break/>.<break/>.<break/>.<break/>.<
break/>.<break/>.<break/>.<break/>.<break/>.<break

```
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>.<break/>.<break/>.<break/>.<break/>.<
        break/>.<break/>.<break/>.<break/>.<break/>.<break
        />.<break/>. continue'

    return (
        handler_input.response_builder
            .speak(speak_output)
```

```
            .ask(speak_output)
            .response
    )

class InterceptIntentHandler(AbstractRequestHandler):
"""evil handler"""
def can_handle(self, handler_input):
    # type: (HandlerInput) -> bool
    return ask_utils.is_intent_name("InterceptIntent")(
        handler_input)

def handle(self, handler_input):
    #type: (HandlerInput) -> Response
    speak_output = "Interceptor ativado. No momento em
        Recife, Brasil, a temperatura é de vinte e nove
        graus celsius com céu parcialmente nublado. A previs
        ão para hoje é de chuvas rápidas com máxima de
        trinta e um graus e mínima de vinte e quatro graus.
        Interceptor continua ativo<break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
        break/><break/><break/><break/><break/><break/><
```

```
            break/><break/><break/><break/><break/><break/><
            break/><break/><break/><break/><break/><break/><
            break/><break/><break/><break/><break/><break/><
            break/><break/><break/><break/><break/><break/><
            break/><break/><break/><break/><break/><break/><
            break/><break/><break/><break/><break/><break/><
            break/><break/><break/><break/><break/><break/><
            break/><break/><break/> continue"

        return (
            handler_input.response_builder
            .speak(speak_output)
            .response
        )
```

O skill builder no fim do arquivo também inclui uma linha "sb.add_request_handler(InterceptIntentHandler())"para adicionar o novo intent

A.3 Bypass da API

O código utilizado no *backend* é o arquivo lambda_function.py padrão da skill de exemplo, com exceção da classe LaunchRequestHandler, que foi modificada, e da classe InfoIntentHandler, que foi adicionada.

```
class LaunchRequestHandler(AbstractRequestHandler):
"""Handler for Skill Launch."""
def can_handle(self, handler_input):
    # type: (HandlerInput) -> bool

    return ask_utils.is_request_type("LaunchRequest")(
       handler_input)

def handle(self, handler_input):
    # type: (HandlerInput) -> Response
    speak_output = "Bem-vindo, diga seu número de telefone
        para continuar"

    return (
        handler_input.response_builder
            .speak(speak_output)
            .ask(speak_output)
            .response
    )
```

```
class InfoIntentHandler(AbstractRequestHandler):
    """Handler for fish"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("InfoIntent")(
            handler_input)

    def handle(self, handler_input):
         # type: (HandlerInput) -> Response
        slot = ask_utils.request_util.get_slot(handler_input,
            "phone")
        slot = str(slot.value)
        phone = ""
        for i in slot:
            if i == "9":
                phone += "nove "
            elif i == "8":
                phone += "oito "
            elif i == "7":
                phone += "sete "
            elif i == "6":
                phone += "seis "
            elif i == "5":
                phone += "cinco "
            elif i == "4":
                phone += "quatro "
            elif i == "3":
                phone += "três "
            elif i == "2":
                phone += "dois "
            elif i == "1":
                phone += "um "
            elif i == "0":
                phone += "zero "
        speak_output = f"O número {phone} foi obtido sem pedir
            permissões ou utilizar a a. p. i. adequada."

        return (
            handler_input.response_builder
                .speak(speak_output)
                # .ask("add a reprompt if you want to keep the
                    session open for the user to respond")
                .response
```

)

O skill builder no fim do arquivo também inclui uma linha "sb.add_request_handler(InfoIntentHandler())"para adicionar o novo intent

Referências Bibliográficas

[1] About cwe. https://cwe.mitre.org/about/index.html.

[2] Alexa developer documentation. https://developer.amazon.com/en-US/docs/alexa/documentation-home.html.

[3] Amazon announces third quarter results. https://ir.aboutamazon.com/news-release/news-release-details/2020/Amazon.com-Announces-Third-Quarter-Results/.

[4] Amazon fixes alexa glitch that could have divulged personal data. https://threatpost.com/amazon-alexa-one-click-attack-can-divulge-personal-data/158297/.

[5] Cwe glossary. https://cwe.mitre.org/documents/glossary/index.html.

[6] Cwe "root cause mapping"guidance. https://cwe.mitre.org/documents/cwe_usage/guidance.html.

[7] Mitre att&ck. https://attack.mitre.org/.

[8] Owasp top 10 - 2021. https://owasp.org/Top10/.

[9] What is the alexa skills kit? https://developer.amazon.com/en-US/docs/alexa/ask-overviews/what-is-the-alexa-skills-kit.html.

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

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

[12] Christopher Lentzsch, Sheel 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. 02 2021.

[13] Khalid Mahmood Malik, Hafiz Malik, and Roland Baumann. Towards vulnerability analysis of voice-driven interfaces and countermeasures for replay attacks. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 523–528, 2019.

[14] Aafaq Sabir, Evan Lafontaine, and Anupam Das. Hey alexa, who am i talking to?: Analyzing users' perception and awareness regarding third-party alexa skills. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[15] Dan Su, Jiqiang Liu, Sencun Zhu, Xiaoyang Wang, and Wei Wang. "are you home alone? yes"disclosing security and privacy vulnerabilities in alexa skills, 2020.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).