A Case Study of the Security Vetting Process of Smart-home Assistant Applications

Hang Hu[2], Limin Yang[1], Shihan Lin[3], Gang Wang[1]

[1]University of Illinois at Urbana-Champaign [2]Virginia Tech [3]Fudan University

hanghu@vt.edu, liminy2@illinois.edu, shlin15@fudan.edu.cn, gangw@illinois.edu

***Abstract*—The popularity of smart-home assistant systems such as Amazon Alexa and Google Home leads to a booming third-party application market (over 70,000 applications across the two stores). While existing works have revealed security issues in these systems, it is not well understood how to help application developers to enforce security requirements. In this paper, we perform a preliminary case study to examine the security vetting mechanisms adopted by Amazon Alexa and Google Home app stores. With a focus on the authentication mechanisms between Alexa/Google cloud and third-party application servers (*i.e.* endpoints), we show the current security vetting is insufficient as developers' mistakes cannot be effectively detected and notified. A weak authentication would allow attackers to spoof the cloud to insert/retrieve data into/from the application endpoints. We validate the attack through ethical proof-of-concept experiments. To confirm vulnerable applications have indeed passed the security vetting and entered the markets, we develop a heuristic-based searching method. We find 219 real-world Alexa endpoints that carry the vulnerability, many of which are related to critical applications that control smart home devices and electronic cars. We have notified Amazon and Google about our findings and offered our suggestions to mitigate the issue.**

I. INTRODUCTION

Smart home assistant systems such as Amazon Alexa and Google Home are entering tens of millions of households today [12]. As a result, the corresponding app marketplace is also expanding quickly. Just like installing apps on smartphones, users can enable third-party applications for smart-assistant devices. These applications are called "skills" or "actions". So far there are collectively more than 70,000 skills available [2], [1], many of which are security/safety-critical. For example, there are skills that allow users to manage bank accounts, place shopping orders, and control smart-home devices through a voice interface.

Considering the sensitive nature of smart-home assistants, researchers have looked into the security aspects of these systems and their third-party applications. For example, recent studies show that it is possible to craft a voice clip with hidden commands embedded that are recognizable by the Alexa device but not by human observers [7], [19]. In addition, researchers demonstrate the feasibility of a "skill squatting" attack to invoke a malicious application whose name sounds like the legitimate one [13]. A recent survey study [4] investigated the network interfaces of many IoT devices (including smart-assistant devices) to reveal their weak encryptions and unpatched OS/software. While most existing studies focus on the system and device-level flaws, limited efforts are investigated to *vetting the security of third-party applications*, and more importantly, *helping developers to improve the security of their applications*.

In this paper, we perform a preliminary *case study* to examine the mechanisms that Amazon and Google implemented to vet the security of third-party applications for their smart home assistants. More specifically, before a third-party application (or "skill") can be published to the app stores, they must go through a series of automated tests and manual vetting. In this paper, we seek to understand (1) what aspects the security vetting process is focused on, and (2) how effective the vetting process is to help developers to improve security.

As a preliminary study, we focus on the authentication mechanism used by the third-party application's server (called "endpoint") to authenticate the cloud (namely, cloud authentication). We choose cloud authentication because cloud-endpoint interaction is a key component that makes smart-home assistant skills structurally different from the conventional smartphone apps. Smart assistant skills need to route their traffic to a central cloud to translate a voice command to an API call in order to interact with the application server.

Amazon Alexa runs both *automated vetting* and *manual vetting* before a skill can be published, while Google Home only runs *manual vetting*. Our methodology is to build our own (vulnerable) skills and walk them through the required testing to understand the vetting process. Our results show concerning issues in terms of the enforcement of cloud authentication. First, we find that the Google Home vetting process does not require the endpoints to authenticate the cloud and their queries, which leaves the endpoints vulnerable to spoofed queries. Second, Amazon Alexa requires skills to perform cloud authentication, but does a poor job enforcing it on third-party developers. Alexa performs automated vetting that is supposed to detect and inform developer mistakes in the skill implementation. However, the security tests are erroneous and have missed important checks (*e.g.*, application identifiers). As a result, a vulnerable skill, in theory, can pass the vetting to enter the app store.

To illustrate the problem, we run controlled experiments to show how an outsider can spoof the cloud to query the target endpoint. More specifically, an attacker can build its own skill application, and use this skill to obtain a valid signature from the cloud for the attack traffic. Then the attacker can replay

the signed traffic to attack the target endpoints. The attack is possible because the cloud uses the same private key to sign all the traffic for all the skills. The signature obtained by the attacker's skill works on the victim's endpoint too. We validate the feasibility of the attack and show that vulnerable skills can bypass both the automated tests and the manual vetting process to enter the app markets.

To confirm that there are indeed vulnerable skills in practice, we perform a scanning experiment. Since all Google Home skills are by default vulnerable, this experiment focused on searching for vulnerable Alexa skills. We leverage ZMap to locate live HTTPS hosts and replay a spoofed but *non-intrusive* query to see if a given HTTPS host returns a valid response. In this way, we located 219 vulnerable real-world Alexa endpoints. A closer analysis shows that some of these vulnerable endpoints are related to important skills such as those that control electric cars, smart locks, security cameras, and watering systems.

We make three main contributions:

- First, we present an empirical analysis of the security vetting process used by Amazon and Google to vet their smart-home assistant skills. We find that the current vetting process is insufficient to identify and notify developers of the authentication issues in their endpoints.
- Second, we validate the problem by running a proof-of-concept cloud spoofing attack, in an *ethical manner*.
- Third, we discover real-world applications that carry the vulnerability. We notified Amazon and Google about our findings and offered our suggestions to mitigate the issue.

II. Background & Motivation

Alexa and Google Home Skills. Both platforms support third-party applications, which are called "Skills" on Alexa and are called "Actions" on Google Home. For convenience, we refer to applications of both platforms as "skills". Figure 1 shows how a user interacts with a skill. (❶) a user talks to the edge device to issue a voice command. (❷) the edge device passes the audio to the Alexa cloud. (❸) the cloud is responsible to convert the speech to text and recognize which skill the user is trying to interact with. In addition, the cloud infers the "intent" of the command and match it with the known intents pre-defined by the skill developers. Here, *intent* is a short string to represent the functionality of the skill. After that, the cloud sends an HTTPS request to the skill's endpoint (*i.e.*, a web server). (❹) the endpoint sends the response back, and (❺) the cloud converts the text-based response to audio, and (❻) plays it at the edge-device. Note that the edge device never directly interacts with the endpoint, and every request is routed through the cloud. For certain skills, users need to explicitly "enable" them, but many other skills can be directly triggered/used by calling the skill's name.

Skill developers need to implement the endpoint to respond to user requests. For simple skills that do not require a database, both Alexa and Google provide a "serverless" option for developers to hard-code the responses in the cloud. For sophisticated skills, an endpoint is needed.

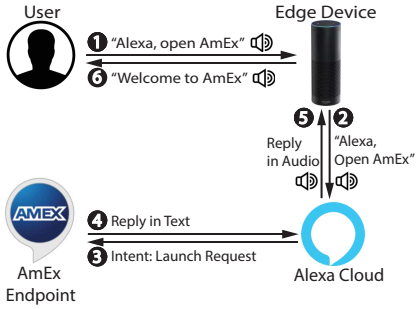

Fig. 1: The execution of a simple voice command.

Authentication between Entities. The system contains three main entities: the edge device, the cloud, and the endpoint. Both Alexa and Google require HTTPS for all communications, which also helps the clients to authenticate the servers. In order for the servers to authenticate the clients, first, in step ❷, the cloud needs to authenticate the edge device. This can be done because an "access token" has been exchanged when the user first sets up the edge device at home. Second, in step ❸, the endpoint also needs to authenticate the cloud. This step helps the endpoint to ensure that the queries are indeed coming from the Alexa/Google cloud instead of outsiders. We call it "cloud authentication", which is done in different ways for Alexa and Google. *Amazon Alexa* uses a public-key based method. The cloud signs its request payload with a private key, and the skill endpoints can verify the signature using the cloud's public key when receiving the request. The verification is *required*. *Google Home* does not require authentication between the cloud and the endpoints.

Security Vetting Process. To make sure the skill and the endpoint are implemented properly, there are two types of vetting deployed by Amazon Alexa and Google Home.

- *Automated Skill Vetting.* Alexa requires a skill to pass a series of tests before allowing the skill to enter the app store. The test is fully automated and covers both functional tests and security tests. Google Home, however, doesn't have an automated test for the skill endpoint.
- *Manual Vetting.* For both Alexa and Google, there are dedicated teams that perform manual vetting on the skill before publishing the skill.

Our Focus. Our goal is to perform a case study to learn how effective the security vetting is and how well it helps developers to develop secure skills. We primarily focus on *cloud authentication* between the cloud and the third-party endpoints because it is *entirely implemented by the skill developers*. In addition, the cloud-to-endpoint communication is also the key reason why smart-assistant skills are fundamentally different from conventional mobile apps — there is a need for the cloud in the middle to translate a voice command to an API call. Considering app developers often lack the security experience [3], [15]. Amazon and Google are in the best position to act as the gatekeeper to ensure all the developer components that interact with their infrastructure (*i.e.*, the cloud) are securely implemented.

Implementation	Certificate Options					
	Standard		Wildcard		Invalid	
	Pass?	#Req.	Pass?	#Req.	Pass?	#Req.
Valid	✓	30	✓	30	✗	23
Ignore App-ID	✓	30	✓	30	✗	23
Ignore Time	✓	30	✓	30	✗	23
Accept All	✗	30	✗	30	✗	23
Reject All	✗	35	✗	35	✗	33
Offline	✗	0	✗	0	✗	0

TABLE I: Results of Alexa automated test. *We report whether the skill passed the test, and the number of testing requests that the endpoint received.*

III. Automated Skill Vetting

We start with Alexa's automated skill vetting (since Google Home does not have an automated vetting process). Our goal is to understand what security tests are running against the skill under vetting. We build our own skills, deliberately leave mistakes, and examine if the automated tests can detect them.

A. Setting Up Vulnerable Skills

We implement an Alexa skill with 6 different versions and each version contains different security or functional errors.

Supported Intents. Every Alexa skill should support 6 default command-lines defined by Amazon Alexa, and at least 1 custom command-lines defined by the developer. The 6 default command-lines are mapped to 6 built-in *intents*. These intents include "LaunchRequest", "StopIntent", "CancelIntent", "FallbackIntent", "HelpIntent", and "NavigateHomeIntent", which are used to perform the basic controls of the skill. We implement the skill to support all 6 default intents and 1 custom intent that takes an integer parameter.

HTTPS Certificate. Both Alexa and Google require HTTPS for the endpoints. Two types of certificates are allowed including standard certificate and wildcard certificate. For our experiment, we test both types of valid certificates, and use a self-signed certificate as the baseline.

Implementing the Cloud Authentication. The cloud authentication is used for the endpoints to authenticate the incoming requests from the cloud. According to the Alexa documentation, the request from the cloud will contain the signature from the cloud. In addition, each request also contains an *application-ID* which indicates which application (skill) this request is intended for; and a *timestamp*. Below, we develop 6 different versions of the endpoints:

1) **Valid implementation**: For a given request, we validate the cloud signature, application-ID, and timestamp before sending a response.
2) **Ignoring application-ID**: Everything is implemented correctly, except that we ignore the application-ID.
3) **Ignoring timestamp**: Everything is implemented correctly, except that we ignore the timestamp.
4) **Accepting all requests**: We do not perform authentication, and always return a legitimate response.
5) **Rejecting all requests**: We drop all the requests.
6) **Offline endpoint**: The endpoint is not online.

B. Skill Testing Results

We tested our skill with 18 different settings (3 certificates $\times$ 6 endpoint implementations) in September 2019. As shown in Table I, standard certificate and the wildcard certificate return the same results. However, when using an invalid certificate (self-signed), even the correct implementation could not pass the test. The test was terminated immediately when the invalid certificate was detected. This result indicates that the automated tests have successfully identified invalid certificates.

As shown in Table I, the "*Accept All*" implementation failed to pass the test. Analyzing the server logs shows that Alexa cloud has sent a number of queries that carry empty or invalid signatures. If we accept these requests, Alexa will determine the endpoint is vulnerable and should not be published in the store.

However, we notice that the "*Ignore Application-ID*" and "*Ignore Timestamp*" implementations both passed the automated test. This means that if the endpoint validates the signature but *ignores the application-ID or the timestamp*, the skill can still proceed to be published. The result raises a major concern. Without validating the application-ID, an endpoint may accept a (malicious) request that is not intended for itself.

IV. Spoofing the Cloud

The above experiment has two takeaways. First, Alexa enforces the endpoint to validate the signature of the incoming request; This means that published skills only accept incoming requests signed by Alexa. Second, Alexa does not enforce the endpoint to validate the application-ID or the timestamp. This means it's possible a skill endpoint may accept and process outdated requests or requests that are not intended for itself. This can lead to a cloud spoofing attack.

Attacking Method. Given a target skill (the victim), the attacker's goal is to spoof the cloud to interact with the endpoint to insert or retrieve data. We use Figure 2 to describe the attack process. The idea is that the attacker builds its own skill, and use this skill to sign the malicious request that will be used for the attack. (❶) the attacker registers its own skill and the mocking intent. The mocking intent should mimic one of the victim skill's intents (so that the crafted payload is understandable by the victim endpoint). (❷) Both Alexa and Google have a text interface that allows the developers to type in their command-lines for testing purposes. Using this text interface, the attacker can trigger the cloud to send a request with malicious payload to its own endpoint. (❸) At this point, the request already carries a valid signature signed by the Alexa cloud. (❹) The attacker can record this request and *replay* it to the target endpoint. The victim endpoint will believe the request is from the Alexa cloud. Since Alexa cloud *uses the same private key* to sign all the requests for all skills, the signature signed for one skill works for another skill.

An endpoint can detect this attack if the endpoint checks the *application-ID* in the request: the *application-ID* inside of the request is still the ID of the attacker's skill. Because the application-ID is inserted by the Alexa cloud before signing the payload, the attacker cannot modify this field.

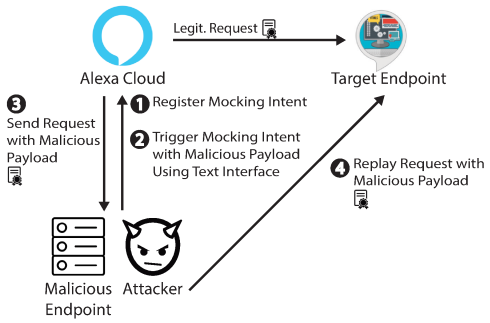

Fig. 2: The cloud spoofing attack.

Proof-of-Concept Experiment. The ability to spoof the cloud can lead to different attacks. This is different from public-facing web services since skill endpoints are not designed to be public-facing: they only expect incoming requests from the cloud. To validate the attack feasibility, we set up our own target skill `A` as the victim. The victim endpoint `A` is configured to "ignore application-ID and timestamp". Then we simulate an attacker by building another skill `B`, and use the endpoint of `B` to collect the malicious requests that will be replayed. We perform this test for both Alexa and Google Home to trigger all 6 default command lines in `A` (*e.g.*, to launch or pause the skills). All the attacks were successful.

SQL Injection Attack. Attackers may perform SQL injection attacks on top of the cloud spoofing. The idea is to design a malicious SQL statement and then get the payload signed by the cloud using her own skill. Then the attacker can replay the signed SQL statement to the victim endpoint. To validate the feasibility, we run an attack on *our own skill*. We target the skill's Custom Intent that has an integer parameter. The skill server does not have any SQL injection defense (*e.g.*, input sanitization). We run a series of SQL injection attacks (*e.g.*, inserting data, dropping a table), all of which are executed successfully. In practice, this attack might be more difficult since attackers may not have the full knowledge of the victim endpoints. The attacker needs to guess: (1) the intent name and its parameter name; (2) the name of the target table; (3) the name of the target column. For example, existing SQL injection tools such as SQLMap and web vulnerability scanners [8] would crawl the corresponding websites, find candidate URLs, and issue a large volume of testing queries. It might take hundreds or thousands of automated guesses to search for an injection opportunity (out of the scope of this paper). For Items (2)–(3), there is a way to find related metadata in many mainstream databases. For example, the MySQL database has a metadata table that contains information about table names and column names.

Ethical Considerations. The experiments above are ethical since both the attacker endpoint and the victim endpoint are developed by us. There are no other skill endpoints or users involved in the experiments. The experiment only sends a few requests to the cloud service, and then forwards the traffic to our own endpoints. Those requests won't overwhelm or damage the cloud service.

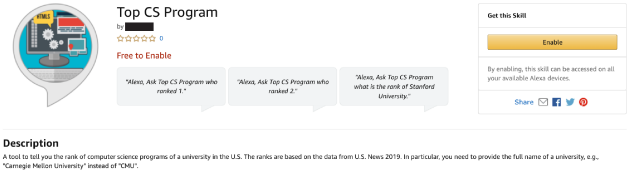

Fig. 3: Vulnerable skill in the Alexa Skill Store.

V. Manual Vetting

Before publishing, a skill needs to be vetted manually by the platform. To understand the manual vetting process, we send our vulnerable skills (vulnerable to spoofing and SQL injection) for publishing. During the submission process, we did not receive any suggestions related to security issues. Both skills received approval to be released within 1–2 weeks. Figure 3 shows the screenshot of our published skill page (the screenshot for Google is omitted for brevity). We immediately took the skill down from both stores after the experiment and informed Google and Amazon about our research. The result suggests that the current vetting process is not rigorous enough to help developers to detect vulnerabilities.

Ethical Considerations. We took active steps to ensure research ethics. At the high level, this skill is a "vulnerable" skill instead of a "malicious" skill. It is supposed to be the victim instead of the attacker in the threat model, which should not introduce any malicious impact. One concern of publishing a vulnerable skill is that the skill may be accidentally used by an innocent user. To avoid this, we have closely monitored our skill endpoint throughout the releasing process. Once the skill received the approval, we immediately performed a quick test to ensure the skill is truly available on the store, and then took it down from the store right away. During this process, we monitored our server and we did not see any incoming requests (except those from our own). This means no real users have ever used this vulnerable skill. In addition, the skill is designed to *provide* information (about Computer Science programs in the US), without collecting any user data. Even if a user accidentally used the skill, there is no actual harm.

VI. Alexa Vulnerable Endpoints

So far, we show that the security vetting process is not rigorous enough to prevent a vulnerable skill from entering the app store. Next, we focus on *Alexa skills* and examine whether there are indeed real-world vulnerable skill endpoints. Google Home skills are by default vulnerable to spoofing and thus are omitted in this measurement.

A. Methodology to Locate Skill Endpoints

For this analysis, we aim to detect endpoints vulnerable to cloud spoofing. We did not further test SQL injection considering the intrusive nature of SQL injection attacks. We face two main challenges. First, the smart assistant devices (edge device) do not directly interact with the skill endpoints. Instead, all the network traffic is first routed to the Amazon cloud. As such, it is difficult for outsiders (*i.e.*, researchers) to know the IP or domain name of the endpoint. Second, even

IPs enable Port443	Round 1			Round 2	Total
	Domain Set	Candidate Hosts	Vul. EPoints	Vul. EPoints	Vul. EPoints
48,141,053	3,196	3,346,425	122	100	219

TABLE II: Searching results of vulnerable endpoints.

if the IP is known, the skill service is not necessarily always hosted under the root path.

Method Overview. We propose a heuristic-based searching method, based on two intuitions. First, a skill endpoint is required to support HTTPS, which means the port 443 should be open. Second, an Alexa endpoint should support the default intents such as "LaunchRequest" and "StopIntent". The response for a default intent request should follow the special JSON format defined by Alexa. As such, we search for vulnerable skill endpoints by scanning the HTTPS hosts with a testing query. The query carries the spoofed "LaunchRequest" intent which is a default intent that every Alexa skill should support. We choose this intent because "LaunchRequest" won't cause any internal state change or reveal any non-public information.

Implementation. Given the large number of HTTPS hosts and the need for *guessing the path*, it is not feasible to test a large number of possible paths on all HTTPS hosts. As such, we prioritize search efficiency by sacrificing some coverage. First, we focus on a small set of HTTPS hosts and test many possible *paths*. Then we select the *most common path* to scan the rest HTTPS hosts.

For round-1, we select a small set of HTTPS hosts that are more likely to be the skill endpoints. More specifically, we crawled 32,289 Alexa skills pages from Amazon store, and extract their URLs of the privacy policies. Our hypothesis is that the skill endpoint might share the same *domain name* with the privacy policy URL. Note that some skills host their privacy policy on cloud services (*e.g.*, "`amazonaws.com`"). As such, we make a whitelist of web hosting services and only consider the hostname (instead of the domain name) in their privacy policy URLs as the candidate.

Then we test a list of possible paths. We obtain the path information by analyzing the example code on the Developer Forum of Alexa and related question threads in StackOverflow. For each host, we test the root path "/", and other possible paths including "/alexa", "/echo", "/api", "/endpoint", "/skill", "/iot", "/voice", "/assistant", and "/amazon".

After round-1, we expect to find some real-world skill endpoints. Then, we select the most common non-root *path* name. We use this pathname and the root path to test all the HTTPS hosts that have not been tested in round-1.

Ethical Considerations. We have taken active steps to ensure research ethics. First, for each host, we only send a handful of queries that have minimal impact on the target host. Second, as detailed below, we re-use the ZMap scanning results [9] instead of performing our own network-wise scanning to identify HTTPS hosts. The scope is aligned with ZMap port 443 scanning which excludes a list of hosts that don't want to be scanned. We respect Internet hosts that don't want to be scanned by ZMap and did not test these hosts. Third, we only test a non-intrusive Intent that does not cause any internal state change of the skill service or reveal any non-public information.

B. Detecting Vulnerable Skill Endpoints

We start with a list of 48,141,053 IPv4 addresses with an open 443 port from ZMap's scanning result archive [9].

Round-1 Search. As shown in Table II, we obtained the Privacy policy URLs from all the 32,289 skills available in the Alexa U.S. skill store. We extracted 3,196 unique domain names. By matching these domain names with those of the 48 million HTTPS hosts, we got 3,346,425 candidate hosts.

By testing the spoofed intent (and candidate paths), we found 122 Alexa skill endpoints that provided a valid response. Here we use an IP address to uniquely represent an endpoint server. In fact, we have identified 174 URLs that have returned a valid response. Some of the URLs are actually mapped to the same IP address.

Round-2 Search. Based on the round-1 result, we find that "`/alexa`" is the most common path (88 out of 174), followed by the root path (30 out of 174). Next, we use these two paths to perform the round-2 searching. As shown in Table II, we discovered 100 *additional* vulnerable endpoints.

Vulnerable Skill Endpoints. From the two rounds of searching, we detected in total 219 vulnerable Alexa endpoints. It should be noticed that the searching result is only a lower-bound considering the incomplete guessing of pathnames. There could be even more vulnerable skill endpoints.

We then examine the geolocation distribution of these vulnerable endpoints based on their countries. We observe that more than half of vulnerable endpoints (115, 52.5%) are located in the United States, followed by Germany (35, 16.0%) and Ireland (16, 7.3%). The top 3 countries cover 75.8% of all vulnerable endpoints.

Case Studies. We send another spoofed "HelpIntent" request to each endpoint, and the returned information helps to identify the actual skills. Some vulnerable skills are less "safety-critical" which are related to games, sports, and news. However, there are indeed skills that are providing critical services. For example, one vulnerable skill on Alexa is used for controlling electric cars. At least three vulnerable skills are from online banking services. A number of vulnerable skills are used to control other smart-home or IoT devices to adjust the air purifier and thermostats, set an alarm for the home security system, and keep track of water and electricity usage. We give a few specific examples below.

"*Brunt*" is an automated home furnishing accessory company, and its products include smart plugs, wireless chargers, air purifiers, blind controllers, and power sockets. The vulnerable skill "Brunt" supports turning on and off Brunt devices and changing their configurations. "*My Valet*" is one of the most popular skills that can control Tesla cars. The skill can be used remotely to lock and unlock the car, obtain information of the car's current location, and open the roof and the trunk. Note that the skill is not officially developed by Tesla, and the skill's

endpoint is vulnerable to cloud spoofing. "*Newton Mail*" is an email application, supporting reading recent emails and other common operations such as deleting an email.

VII. Related work

IoT Security & Privacy. With the wide adoption of IoT devices, security and privacy have become a pressing issue [5]. A related research direction looks into the *user authentication* schemes of IoT devices [18], malicious command-lines injection [6] and controlling the device through inaudible voice [19] due to a lack of authentication. A recent work that incorrect endpoint side checks can lead to severe attacks including password brute-forcing, leaked password probing, and security access token hijacking [20]. Our work is different since we look into smart-home assistant systems and examine the interaction between the cloud and third-party endpoints (instead of user-end authentication).

App Developers and Security Coding. A related body of work focuses on understanding the mistakes made by app developers. For example, researchers show that many poorly implemented security mechanisms in mobile apps are due to developers who are inexperienced, distracted or overwhelmed [3], copying and pasting code from online forums [16], asking more permission than they need [17], [11] or failing to use cryptographic API correctly [10]. Even with tools to help the developers in the Android development environment [14], it is often not enough to prevent insecure apps [17]. While most existing works are focused on smartphone apps, we for the first time investigate this issue in smart assistant systems. Our result shows that more work needs to be done to help the developers.

VIII. Discussion & Conclusion

The problem described in the paper comes from the insufficient security vetting process before releasing the applications into the market. There is a confusion between cloud and application-level authentication in Alexa's automated testing. Alexa enforced an endpoint to verify the cloud identity but did not enforce the verification of the application identity. This makes endpoints vulnerable to replayed requests that were intended for other applications (*e.g.*, the attacker's skill). We have reported our findings to the Alexa and Google Home team and informed them about our experiments. The countermeasure is to implement dedicated skill tests and enforce developers to check the application-ID and the timestamp.

Our work has a few limitations. First, our search only covers a limited number of "paths". The number of vulnerable endpoints can only be interpreted as a lower bound. Second, we only confirmed that the endpoints were vulnerable to cloud spoofing attacks. We did not further test SQL injection attacks for ethical considerations. An open question is how to design the security vetting process to effectively help developers. First, we need to improve the coverage of the automated tests to perform more security checks. Second, we need to provide informative and actionable feedback to developers. Such a mechanism can be integrated into the software development kit (SDK) for developers.

Acknowledgment

This project was supported in part by NSF grants CNS-1750101 and CNS-1717028, and Google Research.

References

[1] Google assistant actions up 2.5x in 2018 to reach 4,253 in the us. https://techcrunch.com/2019/02/18/google-assistant-actions-up-2-5x-in-2018-to-reach-4253-in-the-u-s/, 2019.

[2] The number of alexa skills in the u.s. more than doubled in 2018. https://techcrunch.com/2019/01/02/the-number-of-alexa-skills-in-the-u-s-more-than-doubled-in-2018/, 2019.

[3] Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M. L., and Stransky, C. You get where you're looking for: The impact of information sources on code security. In *Proc. of IEEE S&P'16* (2016).

[4] Alrawi, O., Lever, C., Antonakakis, M., and Monrose, F. Sok: Security evaluation of home-based iot deployments. In *Proc. of IEEE S&P'19* (2019).

[5] Apthorpe, N., Varghese, S., and Feamster, N. Evaluating the contextual integrity of privacy regulation: Parents' iot toy privacy norms versus coppa. In *Proc. of USENIX Security'19* (2019).

[6] Bispham, M. K., Agrafiotis, I., and Goldsmith, M. Nonsense attacks on google assistant. *arXiv preprint arXiv:1808.01947* (2018).

[7] Carlini, N., Mishra, P., Vaidya, T., Zhang, Y., Sherr, M., Shields, C., Wagner, D., and Zhou, W. Hidden voice commands. In *Proc. of USENIX Security'16* (2016).

[8] Doupé, A., Cavedon, L., Kruegel, C., and Vigna, G. Enemy of the state: A state-aware black-box web vulnerability scanner. In *Proc. of USENIX Security'12* (2012).

[9] Durumeric, Z., Wustrow, E., and Halderman, J. A. Zmap: Fast internet-wide scanning and its security applications. In *Proc. of USENIX Security'13* (2013).

[10] Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., and Smith, M. Why eve and mallory love android: An analysis of android ssl (in) security. In *Proc. of CCS'12* (2012).

[11] Felt, A. P., Wang, H. J., Moshchuk, A., Hanna, S., and Chin, E. Permission re-delegation: Attacks and defenses. In *Proc. of USENIX Security'11* (2011).

[12] Koetsier, J. Amazon echo, google home installed base hits 50 million; apple has 6% market share, report says. https://www.forbes.com/sites/johnkoetsier/2018/08/02/amazon-echo-google-home-installed-base-hits-50-million-apple-has-6-market-share-report-says, 2018.

[13] Kumar, D., Paccagnella, R., Murley, P., Hennenfent, E., Mason, J., Bates, A., and Bailey, M. Skill squatting attacks on amazon alexa. In *Proc. of USENIX Security'18* (2018).

[14] Li, T., Agarwal, Y., and Hong, J. I. Coconut: An ide plugin for developing privacy-friendly apps. In *Proc. of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2018).

[15] Liu, F., Wang, C., Pico, A., Yao, D., and Wang, G. Measuring the insecurity of mobile deep links of android. In *Proc. of USENIX Security'17* (2017).

[16] Miryung Kim, Bergman, L., Lau, T., and Notkin, D. An ethnographic study of copy and paste programming practices in oopl. In *Proc. of ISESE'04* (2004).

[17] Nguyen, D. C., Wermke, D., Acar, Y., Backes, M., Weir, C., and Fahl, S. A stitch in time: Supporting android developers in writing secure code. In *Proc. of CCS'17* (2017).

[18] Tian, Y., Zhang, N., Lin, Y.-H., Wang, X., Ur, B., Guo, X., and Tague, P. Smartauth: User-centered authorization for the internet of things. In *Proc. of USENIX Security'17* (2017).

[19] Zhang, G., Yan, C., Ji, X., Zhang, T., Zhang, T., and Xu, W. Dolphinattack: Inaudible voice commands. In *Proc. of CCS'17* (2017).

[20] Zuo, C., Wang, W., Lin, Z., and Wang, R. Automatic forgery of cryptographically consistent messages to identify security vulnerabilities in mobile services. In *Proc. of NDSS'16* (2016).