An Analysis of Amazon Echo's Network Behavior

Jan Janak*, Teresa Tseng†, Aliza Isaacs†, Henning Schulzrinne*
*Department of Computer Science, Columbia University, USA
†Barnard College, USA
Email: janakj@cs.columbia.edu, {ttt2148,ai2376}@barnard.edu, hgs@cs.columbia.edu

***Abstract*—With over 20 million units sold since 2015, Amazon Echo, the Alexa-enabled smart speaker developed by Amazon, is probably one of the most widely deployed Internet of Things consumer devices. Despite the very large installed base, surprisingly little is known about the device's network behavior. We modify a first generation Echo device, decrypt its communication with Amazon cloud, and analyze the device pairing, Alexa Voice Service, and drop-in calling protocols. We also describe our methodology and the experimental setup. We find a minor shortcoming in the device pairing protocol and learn that drop-in calls are end-to-end encrypted and based on modern open standards. Overall, we find the Echo to be a well-designed device from the network communication perspective.**

I. Introduction

With over 20 million units sold since 2015 [1], Amazon Echo, the Alexa-enabled smart speaker developed and sold by Amazon, is probably one of the most widely deployed Internet of Things (IoT) consumer devices. The Echo found its way to many homes [2], high school classrooms [3], and some hotels [4]. Despite the very large installed base, surprisingly little is known about the device's network behavior. How secure is the Wi-Fi connection process? How secure is the connection to Amazon cloud? Are the calls made from the Echo encrypted? In this paper, we aim to shed some light on the device's encrypted network communication to answer the questions.

We obtained a first generation Amazon Echo and modified its firmware to make it vulnerable to man-in-the-middle (MITM) attacks. We then connected the device to a MITM-capable testbed where we could record, decrypt, and analyze the network traffic between the device, the companion smartphone application, and Amazon cloud.

We describe our hardware and firmware modifications, the methodology, and the design of the testbed. We then record and analyze the device paring protocol used between the Echo, the companion smartphone application, and Amazon cloud. Next, we decrypt and analyze the Alexa Voice Service (AVS) protocol, focusing on the differences from the publicly documented AVS API available to third-party developers of Alexa-enabled products.

We also record, decrypt, and analyze the protocols used by the Echo's real-time drop-in communication feature (device calling and intercom). We find that this feature is based on modern standard protocols with custom authentication and authorization. We also find that media streams are end-to-end encrypted and that the system is designed to keep the stream within the local (home) network where possible.

The primary contribution of this paper is an analysis and documentation of encrypted network communication of Amazon Echo. Specifically, we analyze: 1) the device pairing protocol (OOB), the AVS protocol, and 3) the Alexa drop-in calling signaling and media protocols. We also discuss some of the design tradeoffs and discovered limitations. Given the large installed base and potentially privacy-invasive nature of the device, we believe more information about the device's network behavior is needed.

The rest of the paper is organized as follows. In Section II we review literature and work related to Amazon Echo. We describe the device modifications, methodology, and the design of our experimental setup in Section III. Section IV analyses in detail the selected network protocols used by first generation Amazon Echo. In Section V we discuss some of the limitations and design tradeoffs discovered in the device pairing and drop-in calling protocols. We conclude and discuss limitations and potential future work in Section VI.

II. Related Work

Clinton et al. [5] performed a hardware analysis of first generation Amazon Echo. Firmware extraction from newer Echo models is discussed in [6]. iFixit published a detailed teardown guide [7] for the device. We used some of these findings in our work: 1) the description of debugging pads on base of the devices, and 2) the ability of the Echo to boot from an external Secure Digital (SD) card. We used a firmware image obtained from [8] and followed the steps outlined in [9] to gain access to the Echo's embedded MultiMediaCard (eMMC) filesystem.

A security analysis of the Echo is presented in [10]. The authors performed a variety of attacks including a sound-based attack, personal identification number (PIN) brute-forcing attack, replay attack on network traffic, and an attempt to exploit Amazon Alexa's application programming interface (API). While no major vulnerabilities were found, the authors discuss potential weaknesses in the 4-digit PIN authentication method and in the wake-word detection algorithm. If successfully exploited, it may be possible to activate the Echo using a highly distorted sound sample which will not be recognized as a wake word by the user, but will be recognized as such by the Echo. This kind of attack could lead to privacy-invasive exploitation of the device.

The authors in [11] analyzed the network behavior of two Echo Dot devices over a 21-day period and created a network signature of the device from the captured traffic. The robustness

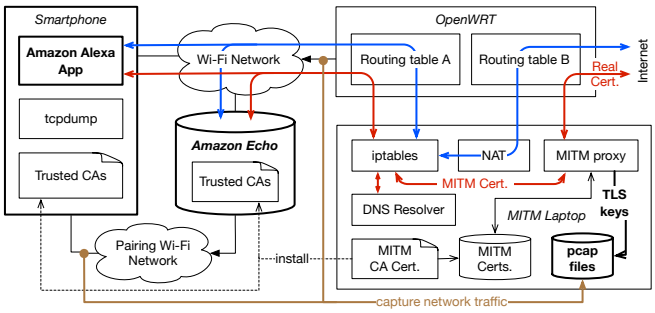

Fig. 1. Experimental setup with Amazon Echo, Android smartphone with Alexa app, and a laptop with Mitmproxy. The Android device and the Echo have been modified to accept Mitmproxy's certificates.

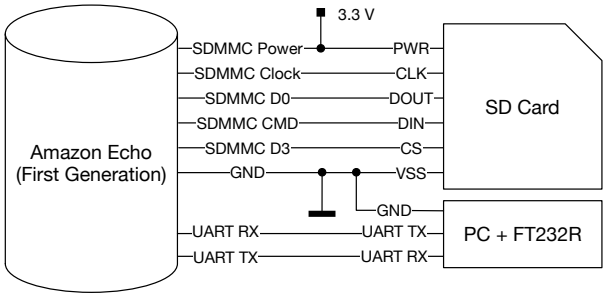

Fig. 2. A modified Amazon Echo with an external SD card and a UART-USB interface attached to the device's debugging pads.

of the signature will likely be limited due to a small number of devices and limited datasets.

The paper [12] analyzes the client and cloud components of the Amazon Alexa ecosystem from a digital forensics perspective. The authors analyzed available artifacts and developed a proof-of-concept digital forensic tool with the ability to collect and analyze the artifacts. The tool uses official and unofficial APIs and collects artifacts from the cloud, Amazon Echo, and smartphone devices.

While there have been several studies [13], [14] focusing on the Echo's hardware security and network behavior, little is known about the device's encrypted traffic. To the best of our knowledge, our study is the first attempt to document and analyze Echo's encrypted network communication.

III. Methodology and Experimental Setup

Fig. 1 shows the architecture of our setup to capture and decrypt network traffic between Amazon cloud, Echo device, and Alexa Android application. We used a first generation Amazon Echo with software version 618622220. The Alexa application was running on a rooted Huawei Honor 6X phone with Android 7.0. Our Wi-Fi access point (AP) was a Linksys WRT1900AC with OpenWRT 17.01.4. Mitmproxy 4.0 (https://mitmproxy.org) was running on a Lenovo X60 laptop with Ubuntu Linux 18.04. We used tcpdump on the rooted smartphone to capture pairing traffic between the Echo and the Alexa app. We also set Mitmproxy's certificate authority (CA) as trusted on the Androi device.

To access encrypted network traffic between the Echo, Alexa app and Amazon cloud, we diverted all outgoing network traffic from the two devices to the laptop with iptables on the OpenWRT router. The laptop also served as a transparent network address translator (NAT) and router for all the network traffic we did not want to decrypt. To decrypt Transport Layer Security (TLS) traffic, we configured iptables on the laptop to forward such traffic to Mitmproxy. This was only applied to TLS connections between the Echo or Alexa app and Amazon cloud. Mitmproxy performed a MITM attack on the connections, replacing Amazon server certificates with locally generated ones. We used Wireshark to decrypt and analyze all captured network traffic.

We modified the Echo device to make it accept Mitmproxy server certificates. All our hardware modifications are based on the work done by the authors in [8], [9]. We exposed the debugging pads at the bottom of the device and connected a UART-USB converter to gain access to the serial console used by the bootloader. We then connected an external SD card to the remaining pads. The entire hardware modification is shown in Fig. 2.

We then loaded the firmware image from [8] on the external SD card and rebooted the device. The custom firmware had a command line interface enabled, which allowed us to change the boot procedure. We followed the steps from [9] to obtain a root shell. In the root shell, we appended the Mitmproxy CA certificate to file /etc/ssl/certs/ca-certificates.crt. After another reboot, the Echo would treat TLS connections modified by Mitmproxy as legitimate, allowing us to decrypt the communication.

IV. Network Behavior

In this section, we document and analyze three network protocols used by Amazon Echo: the device pairing protocol (OOBE), the AVS protocol, and the drop-in calling protocol. All three protocols are partially or fully encrypted with TLS on the network.

A. Device Pairing Protocol (OOBE)

The out-of-box experience (OOBE) is a pairing protocol used to: a) provision a network credential (Wi-Fi network name and password) into the Echo device, and b) to associate the device with an Amazon account. This protocol is executed between the Echo, a pairing client in the form of Amazon Alexa smartphone app or a web application, and backend services in Amazon cloud. Pairing must be performed after the device has been reset to factory defaults, or when the Wi-Fi network becomes unusable, e.g., due to a Wi-Fi password change. Pairing can also be manually activated by pressing and holding down the "dot" button on the Echo.

The pairing exchange takes place over an open temporary Wi-Fi network created by the Echo. The device creates a Wi-Fi peer-to-peer (P2P) group and configures itself as the group owner (GO). The service set identifier (SSID) of the temporary network is "Amazon-XYZ" where XYZ represents three digits from the device's serial number. The Echo can

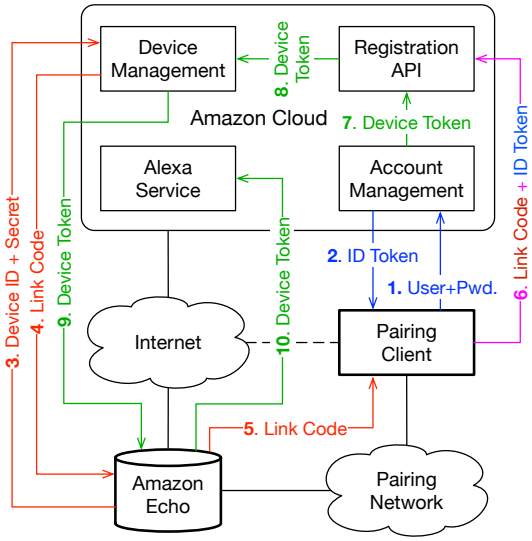

Fig. 3. Data flow diagram of the device pairing protocol (OOBE).

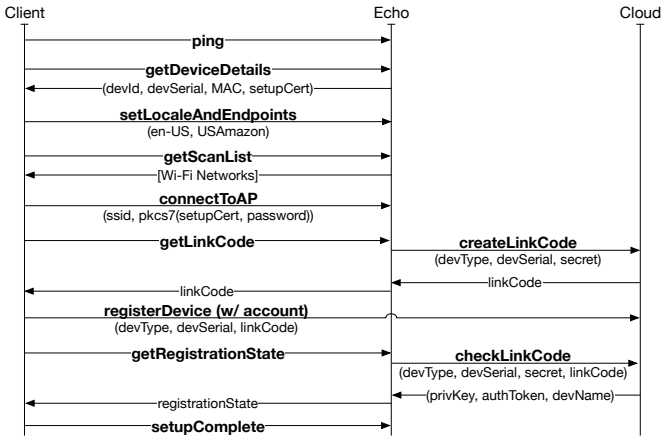

Fig. 4. Call flow diagram of the device pairing protocol (OOBE).

create a temporary Wi-Fi network regardless of the state of its main Wi-Fi interface, i.e., even when connected as client to another network on a different channel.

On the temporary network, the Echo configures itself with a fixed IP address, starts DHCP and DNS servers, and configures iptables to redirect all DNS requests from connected clients to its internal DNS server. The internal DNS server has a few well-known Amazon hostnames and IP addresses hard-wired, presumably to be able to resolve those even when the device itself is not connected to the internet. If connected to the internet during pairing, the Echo forwards (NAT-ed) traffic from the pairing client to the internet via its main Wi-Fi interface. This is used by the pairing client to associate the device with the user's Amazon account. The Echo audibly announces only the first connected pairing client with no identification. Multiple pairing clients can be connected simultaneously.

Fig. 3 shows the data exchanged between the entities involved in the pairing process. Fig. 4 shows the interaction between the pairing client, Echo, and Amazon cloud. The Echo provides a pairing API on TCP ports 8080 and 443. A HTTP server on port 8080 accepts JSON-serialized Thrift [15] messages for the path `/OOBE`. Port 443 is a built-in TCP proxy that forwards HTTPS connections from the pairing client to Amazon cloud.

The pairing client periodically invokes the `ping` API method to determine if it is connected to the correct network on an Echo in pairing mode. Since different Echo models may use different static IPs, the client sends the request to several pre-configured IPs simultaneously. Pairing is initiated with the first device that correctly acknowledges the ping request. The ping method serves as a rudimentary service discovery mechanism. Unlike, e.g., ZeroConf, the ping method can be used by browser applications such as the pairing application available at https://alexa.amazon.com.

Upon discovering an Echo in pairing mode (indicated by spinning orange light), the client invokes the `getDeviceDetails` method. The method returns basic device information: device model, serial number, Wi-Fi hardware address, supported languages, and a pairing X.509 public key certificate (used later). The client then selects the language and configures Amazon cloud endpoints corresponding to the device's geographic location.

Next, the client determines whether the Echo has any Wi-Fi credentials configured already and obtains the list of scanned Wi-Fi networks. In Fig. 4, these two steps are represented by the `getScanList` request. The client prompts the user to select a Wi-Fi network and invokes `connectToAP` to connect the Echo to the selected network.

The pairing client encrypts the Wi-Fi credential with AES-256 in CBC mode using a random secret key. The key is then encrypted with the public key from the X.509 certificate returned by `getDeviceDetails`. The encrypted key and Wi-Fi credential are then transmitted in the Cryptographic Message Syntax (CMS) format [16]. The X.509 certificate provided by the Echo is self-signed and this method is thus vulnerable to active MITM attacks. The encrypted message can be decrypted with the openssl command:

```
openssl smime -decrypt -in <data> -inform \
  pem -inkey /var/local/oobe-web-setup.cert
```

File `/var/local/oobe-web-setup.cert` can be obtained from Echo's firmware image.

The pairing client then obtains a *link code* from Amazon cloud via the Echo. The link code consists of five alphanumeric characters and represents the device during registration with an Amazon user account. The format suggests that the code may have been designed for scenarios where the user is expected to transmit the string manually from the device to Amazon via the web interface. The Echo submits its device type, serial number, and a secret string to the `createLinkCode` API in Amazon cloud. That API appears to be part of a device rendezvous service, most likely connected to an inventory database that keeps track of the serial numbers and secrets for all manufactured devices. The secret string appears to be set on each Echo during the manufacturing process.

Having received the link code, the pairing client registers

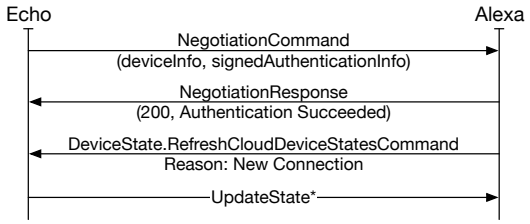

Fig. 5. The setup phase of the persistent connection between the Echo and Alexa Voice Service (AVS).

the device with the user's Amazon account. Since the client is connected to the Echo's pairing Wi-Fi network, it uses the device as a proxy to reach Amazon cloud in order to register the device. The client's request is end-to-end encrypted with TLS and authenticated with a HTTP cookie that authenticates the user. The client submits the device type, serial number, and the link code identifying the device to be registered with the authenticated user's account. Amazon cloud verifies the link code and associates the device with the user's account. The last step is entirely implemented by Amazon cloud and we were not able to obtain more information about it.

The client invokes `getRegistrationState` on the Echo to determine registration status. Using its secret, the Echo queries the rendezvous service for the state of the link code by posting to the `checkLinkCode` API. Once registered, this API returns a private key, an authentication token, and the human-friendly name assigned to the device by the user. The private key and the authentication token are used by the Echo to access Amazon cloud on behalf of the user. The authentication token is included in every HTTP request to Amazon services and appears to include data encrypted with a symmetric key wrapped with a public-private key known only to Amazon and an initialization vector.

The client terminates the pairing process by invoking the `setupComplete` method on the Echo, instructing the device to shut down the temporary Wi-Fi network.

B. Alexa Voice Service (AVS)

The Alexa Voice Service (AVS) is an Amazon cloud service that provides speech recognition, natural language understanding, and text to speech capabilities to Amazon Echo devices. In this section, web briefly describe the proprietary AVS protocol used by our first generation Amazon Echo. AVS is also available to third-party developers of Alexa-enabled products in the form of a public API. The public API is extensively documented online [17], is based on HTTP/2, and uses Amazon's Login With Amazon (LWA) authorization service [18].

Each Echo maintains a persistent long-term SPDY [19] connection to an Echo-specific AVS endpoint. Fig. 5 shows the connection setup phase. The first command sent by the Echo is `NegotiationCommand` which authenticates the device. A portion of the command's JSON payload is cryptographically signed with the private key obtained by the device during pairing. The signed portion contains device type and serial number, an authentication token (also obtained during pairing), and a timestamp. The server then immediately notifies the Echo to refresh the state of various subsystems.

Further communication over the connection is similar to the public AVS protocol described in [17], with the exception that the Echo provides interfaces not accessible via the public API such as the SipClient interface described in Section IV-C. We omit the rest of the protocol due to space constraints and refer interested readers to the public AVS documentation.

C. Alexa Drop-in Calling

Alexa drop-in calling can be used to place calls to other Alexa-enabled Echo devices, to the Alexa smartphone application, to selected U.S. phone numbers, or to Skype. In this section, we take a closer look at the protocols used by this feature. Uttering a phrase such as "Alexa drop in on . . . ", "Alexa call . . . ", or "Alexa answer" causes the device to establish a two-way call with another Alexa-enabled device, a phone number, or a Skype account. The system supports two call types: regular call and intercom.

Regular calls use the caller's address book, provided to the system by the Alexa smartphone app, to look up the callee. If the callee has Alexa-enabled devices, the call is routed to those devices. All devices indicate the incoming call simultaneously, but only one can answer the call. If the callee does not have any Alexa-enabled devices, the call is routed to a public switched telephone network (PSTN) or Skype gateway.

In intercom mode, the call is established between Alexa-enabled devices without the need for the callee to answer the call. The called device answers automatically, provided that the callee has granted a "drop-in" permission to the caller. The caller must specify a particular device to call, not a user or phone number. Often, this would be another Echo paired with the same Amazon account, e.g., an Echo in another room.

The drop-in calling feature is entirely based on open standards. Compatible devices run a Session Initiation Protocol (SIP) [20] user agent (UA) based on PJSIP v2.7.1. Signaling is based on SIP with Outbound [21], Path [22], and Globally Routable UA URI (GRUU) [23] extensions. A media path for each call is negotiated with Session Traversal Utilities for NAT (STUN) [24], Interactive Connectivity Establishment (ICE) [25], and Traversal Using Relays Around NAT (TURN) [26] protocols. Audio is encoded with the Opus codec [27] and the stream is end-to-end encrypted with AES-256 using Secure Real-Time Transport Protocol (sRTP) [28] and Session Description Protocol Security Descriptions (SDES) [29] protocols.

The UA is fully managed by the cloud-based Alexa service over its persistent (SPDY) control connection. Shortly after device start, the UA sends a `ConfigureCommsRequest` provisioning request to Alexa which provides the UA with SIP registration configuration. The configuration includes the SIP username, registrar domain, and a registration authorization credential. The UA establishes and maintains a persistent TLS connection to Amazon's SIP service on non-standard port 443 (HTTPS). We assume the port has been chosen to facilitate firewall traversal. The UA registers with the SIP service using

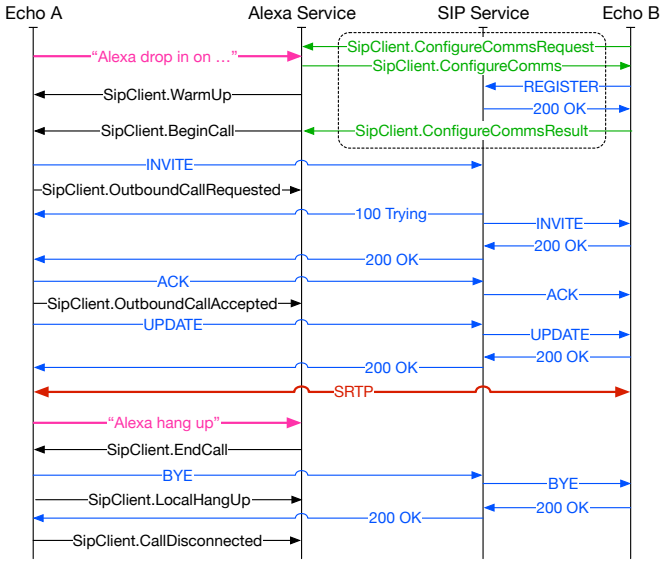

Fig. 6. Intercom call flow diagram (utterances in purple, SIP in blue, Alexa protocol in black). Dashed rectangle represents startup registration signaling.

the configuration provided by Alexa. All devices associated with the same Amazon user account register the same SIP URI. Each UA is also assigned a device-specific SIP URI. The account-specific SIP URI is used to simultaneously reach all devices during regular calls. The device-specific SIP URI is used to reach a particular device during intercom calls.

Having detected an utterance to establish a call, the Alexa service issues a `SipClient.WarmUp` command followed by a `SipClient.BeginCall` command to the UA. The latter command includes JSON payload with detailed call-related configuration: the caller and callee SIP URI, a call authorization token, available TURN and STUN endpoints, and various Alexa-specific attributes. Fig. 6 shows an intercom call flow diagram. The calling UA sends a SIP INVITE to the callee and notifies Alexa that the call is being established with a `SipClient.OutboundCallRequested` message. Once the call has been accepted, the UA notifies Alexa again with a `SipClient.OutboundCallAccepted` message. Both notifications carry payload describing the state of the UA.

Both media streams between two Echo devices are end-to-end encrypted. As is common in SIP-based systems, the encryption key is derived from a master key exchanged in SIP signaling. Since all SIP signaling takes place over authenticated TLS connections, the key is secure against eavesdropping attacks. However, Amazon's SIP service has access to the master key (by design) and can thus decrypt the media streams.

Whether media passes through Amazon cloud depends on network configuration. Fig. 7 shows an architecture diagram with three possible cases. A call established between two devices in the same LAN, as is common for intercom calls between two devices in the same home, remains in the LAN. A call placed to a device behind another home router will likely have media streams forced through a TURN relay in Amazon cloud. This is common when calling another Amazon user.

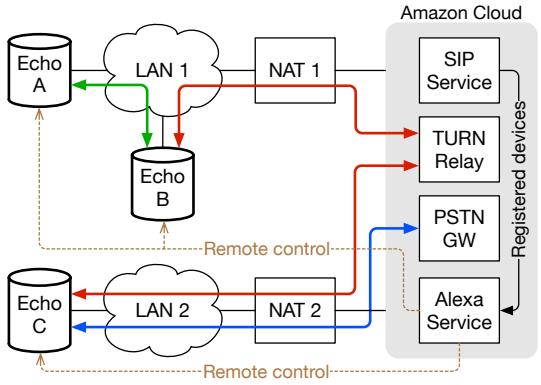

Fig. 7. Architecture of the calling subsystem. SIP, TURN, and PSTN services are provided by Amazon cloud. Calls between devices in the same LAN stay local (green). Others are relayed (red) or terminated in the cloud (blue).

Calls to PSTN or Skype are always terminated at a cloud-based gateway.

Uttering "Alexa hang up" terminates the call. Alexa sends the request `SipClient.EndCall` to the UA which sends a SIP BYE. After the call ended, the UA notifies Alexa with a `SipClient.CallDisconnected` message.

The drop-in calling feature uses a custom SIP authentication method tied to the Alexa service. Each SIP request carries a proprietary `X-authtoken` header with a short-lived authorization token generated by Alexa. A portion of the token is signed by a private key tied to the Amazon user account. Only the Alexa service has access to the key. Since the authorization token is cryptographically bound to both calling and called SIP URIs, the UA cannot initiate a call unless authorized by Alexa. Each call needs a new authorization token.

V. Discussion

The modification shown in Fig. 2 is only effective with the first generation Echo hardware. Consequently, the MITM attack described in Section III is harder to perform on newer Echo models where the firmware applies additional security measures. Thus, installing a custom CA certificate on the device is harder. Howeveer, the network protocols analysed in this paper are compatible with newer Echo models and other parts of the ecosystem.

Neither the temporary Wi-Fi network nor the pairing protocol are encrypted and are vulnerable to eavesdropping. An attacker who could observe the pairing exchange could obtain the link code and might be able to associate a previously de-registered device with different Amazon account. A newly purchased Echo comes pre-registered to the Amazon user account used to purchase the device. This mechanism effectively prevents hijacking an Echo using the pairing protocol, as long as the device is already registered in Amazon cloud.

One of the supported pairing clients is a web application implemented in JavaScript. In order for the client to be able to communicate with the Echo device via unencrypted HTTP, the client itself must be served via unencrypted HTTP. This is necessary to work around the security limitations imposed

by modern web browsers. The web application is first loaded via HTTPS and after the user has logged in, the browser is redirected to a HTTP URL which downloads the JavaScript pairing client. This design leaves the JavaScript pairing client vulnerable to code injection by remote attackers.

The custom authentication mechanism used in drop-in call signaling could use more scrutiny, e.g., to see whether it might be vulnerable to well-known attacks on SIP-based systems such as SIP header substitution, downgrade, or media encryption downgrade. Since the intercom feature in Amazon Echo answers incoming calls automatically, potential vulnerabilities in the design of Echo's SIP infrastructure could turn an Echo device into a remotely controllable microphone.

VI. Conclusion and Future Work

Despite the very large installed base, not much is known about Amazon Echo's network behavior. In this paper, we analyzed and documented the device pairing protocol (OOBE), the Amazon Voice Service (AVS) protocol, and the Alexa drop-in calling protocol used by a first generation Amazon Echo. We modified the firmware to make the device vulnerable to MITM attacks. We then mounted a MITM attack against the device and decrypted the communication between the device, pairing smartphone application, and Amazon cloud. We also described the approach and the experimental setup in detail.

We found limitations in the device pairing protocol (OOBE) which under certain circumstances could be used to associate a de-registered Echo with a different Amazon account. This vulnerability is effectively mitigated by the pre-existing association of a new device to the purchasing Amazon account. Intercom calls are authorized with one-time authorization tokens issued by the Alexa service. Both signaling and media are end-to-end encrypted and use modern industry standard protocols. Overall, we find the first generation Amazon Echo to be a well-designed device from the network communication perspective.

Our experiments were limited to the first generation Amazon Echo. Later models make the MITM approach considerably more difficult. This limitation does not change the analysis of network communication which is the primary contribution of this paper. The analyzed protocols are compatible and interoperable with more recent Amazon Echo models.

Analysing the network communication employed by more recent Echo models, other smart speaker brands, or analysing the communication with other IoT devices on the same LAN is left for future work.

References

[1] Amazon. Amazon Quarterly Results. [Online]. Available: https://ir.aboutamazon.com/quarterly-results

[2] NPR and Edison Research. The Smart Audio Report. [Online]. Available: https://www.nationalpublicmedia.com/smart-audio-report/latest-report

[3] Amazon. Alexa in Higher Education. [Online]. Available: https://developer.amazon.com/alexa/education

[4] ——. Alexa for Hospitality. [Online]. Available: https://www.amazon.com/alexahospitality

[5] I. Clinton, L. Cook, and S. Banik. (2016) A Survey of Various Methods for Analyzing the Amazon Echo. [Online]. Available: https://vanderpot.com/Clinton_Cook_Paper.pdf

[6] S. Vasile, D. Oswald, and T. Chothia, "Breaking all the things—a systematic survey of firmware extraction techniques for iot devices," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2018, pp. 171–185.

[7] iFixit. Amazon Echo Teardown. [Online]. Available: https://www.ifixit.com/Teardown/Amazon+Echo+Teardown/33953

[8] A. Vanderpot. Echohacking Wiki. [Online]. Available: https://github.com/echohacking/wiki

[9] M. Barnes. (2017) Alexa, are you listening? [Online]. Available: https://labs.mwrinfosecurity.com/blog/alexa-are-you-listening

[10] W. Haack, M. Severance, M. Wallace, and J. Wohlwend. (2017) Security Analysis of the Amazon Echo. [Online]. Available: https://courses.csail.mit.edu/6.857/2017/project/8.pdf

[11] M. Ford and W. Palmer, "Alexa, are you listening to me? An analysis of Alexa voice service network traffic," *Personal and Ubiquitous Computing*, pp. 1–13, 2018.

[12] H. Chung, J. Park, and S. Lee, "Digital Forensic Approaches for Amazon Alexa Ecosystem," *Digital Investigation*, vol. 22, pp. S15–S25, 2017.

[13] D. Weinstein. Amazon Echo Dot Explorations (Part 1). [Online]. Available: https://medium.com/@dweinstein/amazon-echo-dot-explorations-part-1-5f5ae38ffeab

[14] N. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic," *arXiv preprint 1705.06805*, 2017. [Online]. Available: https://arxiv.org/abs/1705.06805

[15] Apache Software Foundation. Apache Thrift. [Online]. Available: https://thrift.apache.org/

[16] R. Housley, "Cryptographic Message Syntax (CMS)," RFC 5652, Internet Engineering Task Force, Sep. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5652.txt

[17] Amazon. Amazon Alexa for Developers. [Online]. Available: https://developer.amazon.com/alexa

[18] ——. Login With Amazon Service. [Online]. Available: https://developer.amazon.com/docs/login-with-amazon/conceptual-overview.html

[19] The Chromium Project. SPDY: An experimental protocol for a faster web. [Online]. Available: https://dev.chromium.org/spdy/spdy-whitepaper

[20] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, Internet Engineering Task Force, Jun. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[21] C. Jennings, R. Mahy, and F. Audet, "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)," RFC 5626, Internet Engineering Task Force, Oct. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5626.txt

[22] D. Willis and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts," RFC 3327, Internet Engineering Task Force, Dec. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3327.txt

[23] J. Rosenberg, "Obtaining and Using Globally Routable User Agent URIs (GRUUs) in the Session Initiation Protocol (SIP)," RFC 5627, Internet Engineering Task Force, Oct. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5627.txt

[24] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389, Internet Engineering Task Force, Oct. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5389.txt

[25] A. Keranen, C. Holmberg, and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal," RFC 8445, Internet Engineering Task Force, Jul. 2018. [Online]. Available: http://www.ietf.org/rfc/rfc8445.txt

[26] R. Mahy, P. Matthews, and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)," RFC 5766, Internet Engineering Task Force, Apr. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5766.txt

[27] J. Valin, K. Vos, and T. Terriberry, "Definition of the Opus Audio Codec," RFC 6716, Internet Engineering Task Force, Sep. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6716.txt

[28] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)," RFC 3711, Internet Engineering Task Force, Mar. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3711.txt

[29] F. Andreasen, M. Baugher, and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams," RFC 4568, Internet Engineering Task Force, Jul. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4568.txt