C445H – Part 1

1.

a.

b. Skill squatting attack using longest match/similar name which means your request will be sent to the wrong application.

(Alt: gaining unauthorized access to a system and performing a password dump. If they’ve been hashed with something weak like MD5, these can easily be broken with a dictionary attack)

c. Man-in-middle attack. Intercepting packets from Alexa device to intended server and changing the packet contents. (For what purpose?)

d. General availability attack: flood a service with requests so that it can’t service legitimate users.

e. Prevents malicious systems from being able to invoke arbitrary skills. If they can’t pass the authentication locally, no request is ever made to alexa servers

f. Local DNS cache poisoning so that the Alexa device connects to a malicious service. Then all requests will be routed to a service under the attacker’s control

g. 1. Prevent other people from accessing potentially sensitive information

2. To access the user’s own data. The results are specific to the user

3.

h. If the user has a smart home, then a replay attack can open doors, turn on/off lights, etc. A malicious party could use a replay attack to access the user’s home.

Some sort of denial of service against Alexa by continuously replaying commands?

I. Build a model of the user’s voice from previous commands that they have said. A replayed voice command has different properties because it is played by a speaker which introduces distortions. Anything out of the ordinary could be rejected.

j. 2. Create sleeper worm and embed into all future Alexa devices. 3. Leave a backdoor. 4. Wait a few years. 5. Execute order 66

🤔

2.

1. 3rd party JS includes are needed for external functionality such as ads, analytics, social media buttons, etc. Without them you’d directly need the source of this functionality
2. Over HTTP, the JS include can be modified by any intermediate device to include some malicious code. Not possible over HTTPS because of end-to-end encryption
3. Replace the local a.js file with a malicious one also with a f function? The f function takes in b which is a breach of confidentiality. It can do whatever, which is an integrity attack and can just loop infinitely which is an availability attack.
4. Older browsers that don’t have support for CSP which could issues for the end user
5. All the JS files have to be downloaded from the Google CDN. The requests give up information about your browser which Google can use to build a profile (fingerprinting). The cdn is likely used by other sites so whenever you visit any of them, Google can match the fingerprint and infer it’s you. TOR leaks as little information as possible and tries to make every TOR user as identical as possible so your fingerprint isn’t unique enough to track you.
6. Create as many accounts as you want by changing the email address sent to the service as you get the money just for signing up.
7. The IMEI is unique to a device and fixed. It cannot be changed without replacing a physical component. So the service can track which users have already signed up by keeping track of the IMEIs.
8. Encrypt the history and send through multiple onion routers which can route the message to the next node but can’t decrypt anything else. Each node only has the immediately preceding node which is likely not the original sender
9. Just use phone number to sign up which then requires you to input a code that they send you on sign up. Can spoof what the phone number shows up as but very very difficult to convince the operator to send messages sent to another number to you. Would require physical sim swapping to change the phone number.

3.

1. Click injection could go undetected the longest as the malicious app could provide legitimate functionality and never does anything that the user can detect.
2. Doesn’t fully address Device ID reset fraud. Could have many people just passing the FaceId/TouchId while installing apps.

Downside is more interruptions for the user as now they also have to authenticate when accidentally clicking an ad. Might prevent people from clicking on ads so less installs. Can’t pay per view as you can’t force the user to go through this process just to display an ad.

1. Monitor the number of clicks to installs. If there’s a massive number of clicks for each install, it’s likely that click flooding is happening.
2. Device ID reset fraud – Doesn't affect this because they’re just using a large number of real devices. The blockchain can record these clicks but can’t prevent the attack.

Click Flooding – Can view the frequency of clicks from the timestamps and detect click flooding attacks.

Click Injection – Can detect and reject fake clicks as each ad click has app-specific information which the fake clicks wouldn’t be able to spoof?

SDK spoofing – Generates completely legitimate looking clicks so probably wouldn’t affect this attack.

What kind of new attacks could it allow?

C447H – Part2

1

a.

I) B

Ii) A

Iii) D?

b.

I) A malicious app can monitor the virtual filesystem on Android, specifically the data usage count of a specific app such as Twitter. When the user performs some action such as sending a tweet, you can log the timestamp of each tweet/action. In the context of Twitter, you can look through the public history of tweets made available by twitter and match timestamps. After just a few tweets, it is possible to narrow down the user to a single account.

Ii) (Is it meant to be something original? Or can we just use one of the ones he already mentioned)

A malicious app could potentially use the rotation and movement of the phone to infer PIN numbers for example. The information from the gyroscope can be accessed without any special permissions. The app can know when the phone is off and when the screen turns on, the malicious app can infer what positions on the screen the user is clicking by the rotation and movement of the phone. The positions correspond to either numbers on a keypad or characters on a keyboard and either can potentially be inferred. You could also perform this attack when a banking application is opened for example. Some banking apps (e.g. Santander) require the user to input a 5 digit code every time which could be inferred by the malicious app.

c.

I)

Preprocessor – Takes in the raw app description and processes the data to make it more ready for the other stages. Handles full stops, ends of sentences, etc

NLP parser – Takes in the pre-processed data and uses NLP to identify main components such as named entities

Intermediate representation generator – Takes in the annotated document and creates a First Order Logic based Tree structure from it

Semantic engine – Takes in the structure from the IR generator as well as a semantic graph associated with the resource of a permission and annotates the sentence based off this graph. So it tells us if the resource/permissions is being described by the sentence

Ii) Limitations:

* Requires good descriptions of the app’s full functionality. Won’t work with apps that have very simplistic descriptions
* Heavily reliant on NLP and the WordNet dictionary for synonyms which can cause a large number of false negatives

Can augment the semantic graphs manually to allow for more cases. Can also add context specific dictionaries which we can choose based off the category that the app is in

2.

a.

I) B?

Ii) C

Iii) B

b. Same as 19-20 paper

c.

I) Decompile the benign app, modify the smali with your malicious code, re-sign, and repackage. You can distribute it on any 3rd party app store and to a user it will look the same as the original app.

Ii)

3.

a.

I) B

Ii) A

Iii) A

b.

I)

User makes request

Audio stream is sent to Alexa

In the cloud, the audio goes through TTS. Skill identification and intent recognition

Intent is sent to relevant skill

Skill responds with either text, audio, visual response

Response is sent back to alexa and then to the user

Ii)

Speech-recognition systems are not perfect and can make mistakes.

By taking a black-box approach and creating a skill you’re in control of, you can see what Alexa recognizes when you feed in some voice input. When alexa makes predictable mistakes, an attacker can take advantage of this and create a new skill that corresponds to the mistake made. Usually these mistakes are only a single phoneme off so “cat” and “cut” might be confused by alexa, so an attacker could create a skill using the mistaken word. Can also take advantage of homophones which are words that sound the same but mean different things, e.g. facts and fax.

Once you’ve registered the skill, alexa may use the malicious skill instead of the intended one, e.g. “cat fax” instead of “cat facts”.

c.

I) HanGaurd uses a controller which is installed on the router and monitors which are apps installed on the devices. The monitor collects information about running apps and makes the decisions which are then sent to the router through the control channel to be enforced. Only apps that are in the policy of the monitor will be allowed to reach the IoT devices. The policies can be managed by the user through a mobile app. The iOS monitor makes use of a virtual tunnel to proxy app network activity of all the authorised apps only. This strikes a balance between efficiency and IoT management. On Android, we can directly monitor procfs, a virtual file system, to see all the TCP/UDP activity. This is very efficient and so we can monitor the source/ip address of each packet and send the appropriate control messages based off which app is generating traffic.

Ii) So with apps and devices you have a mapping of authorized app for a specific IoT device. Need something similar where the user can have some authenticated voices which are then mapped to devices. If there’s no match, then the signal is sent by the monitor to the router to reject the command. If there is a match, then the monitor needs to use TTS and send that to the router? Not sure how you would be able to control an arbitrary device like this.