Jack Temko
SMCAA

Security Flows

Our design philosophy was based around using hashes and shared secrets to verify the authenticity of both the component and AP. In order to do this we use Challenge–response authentication. The idea is that the server sends out a nonce (random bytes) and the client hashes the nonce with its secret, then sends back the hash. Then the server compares the received hash with its own calculated hash to ensure that the client shares the same secret.

We also use this hashing method in order to implement *secure_send* and *secure_recieve*. Before the message is sent out, the message to be sent is hashed with the secret, and sent along with the message. On the receiving side, the device hashes the message with its secret and compares that with the received hash. If anything in the message or secret is different, the hash check fails meaning that the message has lost integrity and/or authenticity.

Attestation

1. AP checks to ensure the correct pin is entered using *memcmp*.
2. AP calls *send_validate* to validate component's authenticity
3. Component calls *receive_validate* to respond to AP's validation request
4. AP sends attestation command to component
5. Component calls *send_validate* to validate AP's authenticity
6. AP calls *receive_validate* to respond to component's validation request
7. Component responds with attestation data
8. AP receives attestation data and prints it

Booting

1. AP calls *validate_components* ensure authenticity of all components
2. AP calls *boot_components* to boot all components
3. AP prints boot message
4. AP boots

Helper Functions

validate-components

For each component:

1. AP call *send_validate* to validate components's authenticity
2. Component calls *receive_validate* to respond to AP's validation request

boot-components

For each component:

1. AP sends boot command to component
2. Component calls *send_validate* to validate AP's authenticity
3. AP calls *receive_validate* to respond to component's validation request
4. Component sends AP it's boot message
5. AP receives component's boot message and prints it

send-validate

1. Server creates a random 16 byte nonce using chips onboard TRNG chip
2. Server sends the nonce to the client
3. Client calls *receive_validate* to respond to server
4. Server calls *verify_signature* with the received signature

receive-validate

1. Client receives nonce from server
2. Client calls *create_signature* with the received nonce
3. Client sends signature to server

Cryptography

create_signature

int create_signature(uint8_t data, size_t size, uint8_t* secret, uint8_t* dest)*

1. Initializes an MD5 hash

2. Updates the hash with `data`
3. Updates the hash with the 16 byte `secret`
4. Writes MD5 hash to `dest`

verify_signature

int verify_signature(uint8_t data, size_t size, uint8_t* secret, uint8_t* signature)*

1. Calls *create_signature* with *data* and *secret*
2. Compares that result with *signature* using *memcmp*