<table>
<thead>
<tr>
<th>Role</th>
<th>Name</th>
<th>Role</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>Beau Schwab</td>
<td>Comms. Director</td>
<td>Samuel Goodman</td>
</tr>
<tr>
<td>Developer</td>
<td>David Nunley</td>
<td>Lead Developer</td>
<td>Andrew Langan</td>
</tr>
<tr>
<td>Developer</td>
<td>Cameron Crossley</td>
<td>Developer</td>
<td>Ethan Martindale</td>
</tr>
<tr>
<td>Developer</td>
<td>Tyler McColeman</td>
<td>Video Editor</td>
<td>Seth Tydings</td>
</tr>
<tr>
<td>Developer</td>
<td>Ezequiel Flores</td>
<td>Developer</td>
<td>Henry Reid</td>
</tr>
<tr>
<td>Developer</td>
<td>Grayson Seger</td>
<td>Developer</td>
<td>Developer</td>
</tr>
<tr>
<td>Team Advisor</td>
<td>Eli Cochran</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DELAWARE AREA CAREER CENTER**
CRYPTOGRAPHIC ALGORITHMS

- AES-128-CTR
- HMAC-SHA256
- SECP256r1 ECC
- SHA256
SECURE SECRET GENERATION

Compile-time Python scripts

- deployment/make_secrets.py
  - Generates shared public/private ECC keys
  - Generates unwrapped attest AES key
  - Generates shared HMAC key
- application_processor/make_secret.py
  - Hashes PIN n times for comparison
  - Hashes PIN n-1 times as wrapper for global attest key
  - Hashes Replacement Token
- component/make_secret.py
  - Encrypts attestation data with unwrapped key
NEW I²C PROTOCOL

- No more registers
- Common packet format
- Packet checksums
- Callbacks instead of polling
- 1:1 Request to Response
- Length encoded along with data

```c
struct header_t {
    packet_magic_t magic;
    uint32_t checksum;
};

template<packet_type_t T> struct __packed payload_t;

// Encrypted packet payload
template<> struct __packed payload_t<packet_type_t::SECURE> {
    uint8_t magic;
    uint8_t len;
    uint32_t nonce;
    uint8_t data[64];
    uint8_t __padding[10]; // Pad to multiple of AES block size
    uint8_t hmac[32];
};

template<packet_type_t T> struct packet_t {
    header_t header;
    payload_t<T> payload;
};
```
ATTEST

**Strengths**
- Encryption of data
- Hashed PIN as wrapper
- Signature validation

**Weaknesses**
- No delays
- Small key space
- Signature reuse

```cpp
mitre::application_processor.cpp
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
1 tc_sha256_init(&sha256_ctx);
2 for (uint32_t i = 0; i < ITERATIONS; ++i) {
3   tc_sha256_update(&sha256_ctx, pin, 6);
4 }
5 tc_sha256_final(hash, &sha256_ctx);
6 7 if (memcmp(hash, ATTEST_HASH, 32) != 0) {
8   print_error("Error :\n");
9   return;
10 }
11 tc_sha256_init(&sha256_ctx);
12 for (uint32_t i = 0; i < ITERATIONS - 1; ++i) {
13   tc_sha256_update(&sha256_ctx, pin, 6);
14 }
15 tc_sha256_final(hash, &sha256_ctx);
16
17 unwrap_aes_key(unwrapped_key, ATTEST_KEY_WRAPED, hash, ATTEST.WRAPPER.NONCE);
```
**Strengths**
- Hashed token

**Weaknesses**
- Could not get validation to function

```c
mitre - application_processor.cpp
1  tc_sha256_init(&sha256_ctx);
2  tc_sha256_update(&sha256_ctx, buf, 16);
3  tc_sha256_final(hash, &sha256_ctx);
4
5  if (memcmp(hash, REPLACEMENT_HASH, 32) == 0) {
6      return error_t::SUCCESS;
7  } else {
8      return error_t::ERROR;
9  }
```
**Strengths**

- Secure key exchange protocol for secure send
- TRNG engine
- Mutual signature validation

**Weaknesses**

- Only 1 key pair
- Fault injection during RNG
SECURE SEND & RECEIVE

Strengths
- Ephemeral keys prevent replay across boots (Authenticity)
- Nonces prevent replay during sessions (Authenticity)
- HMAC to prevent modification (Integrity)
- AES encryption to prevent reading (Confidentiality)

Weaknesses
- No MITM protection during KEX
- Leaked HMAC key + MITM = Full compromise
NOT SO RANDOM RAND()

Problem
- \texttt{g\_nKey} is used to encrypt all communications
- \texttt{time(NULL)} returns -1 without implementation
- \texttt{srand(time(NULL))} has a deterministic output
- \texttt{rand()} is not a CSPRNG

Solution
- Use onboard TRNG hardware
- Remove all \texttt{rand()} based code

```
1  srand(time(NULL));
2  bzero(g\_nKey, BLOCK\_SIZE);
3  for(int i = 0; i < KEY\_SIZE;i++)
4      g\_nKey[i] = rand() \% 256;
```
BINARY LEAK

- Exploit binary leaked in public channel “uccon_supply_dump.img”

- Written in Rust

- Exploits Mitre-provided I²C Peripheral library

- Unsuccessful RE due to lack of time
FINAL COMMENTS

- Compile-time secrets
- Memory corruption
- Embedded hardware
- Read the documentation
QUESTIONS?

HTTPS://GITHUB.COM/0XDACC/2024-MITRE-ECTF-DACC.GIT

HTTPS://WWW.LINKEDIN.COM/IN/ANDREWLANGAN/

HTTPS://WWW.LINKEDIN.COM/IN/SAM-GOODMAN-CYB/

HTTPS://WWW.LINKEDIN.COM/IN/BEAUSCHWAB26/