TigerBytes

Rochester Institute of Technology

*Brandon Adler**
*Thomas Cenova**
*Eric Scheler**
Stuart Nevans Locke
Jason Blocklove
Max Proskauer
Prateek Talukdar
Alden Davidson

Advised by: Dr. Ziming Zhao and Dr. Marcin Łukowiak

Outline

- Secure Design
  - Our System
  - Hardware and Software v2.0
- Attack Phase
  - Attack highlights
- General Comments
  - Lessons learned

Our Secure Design

SW

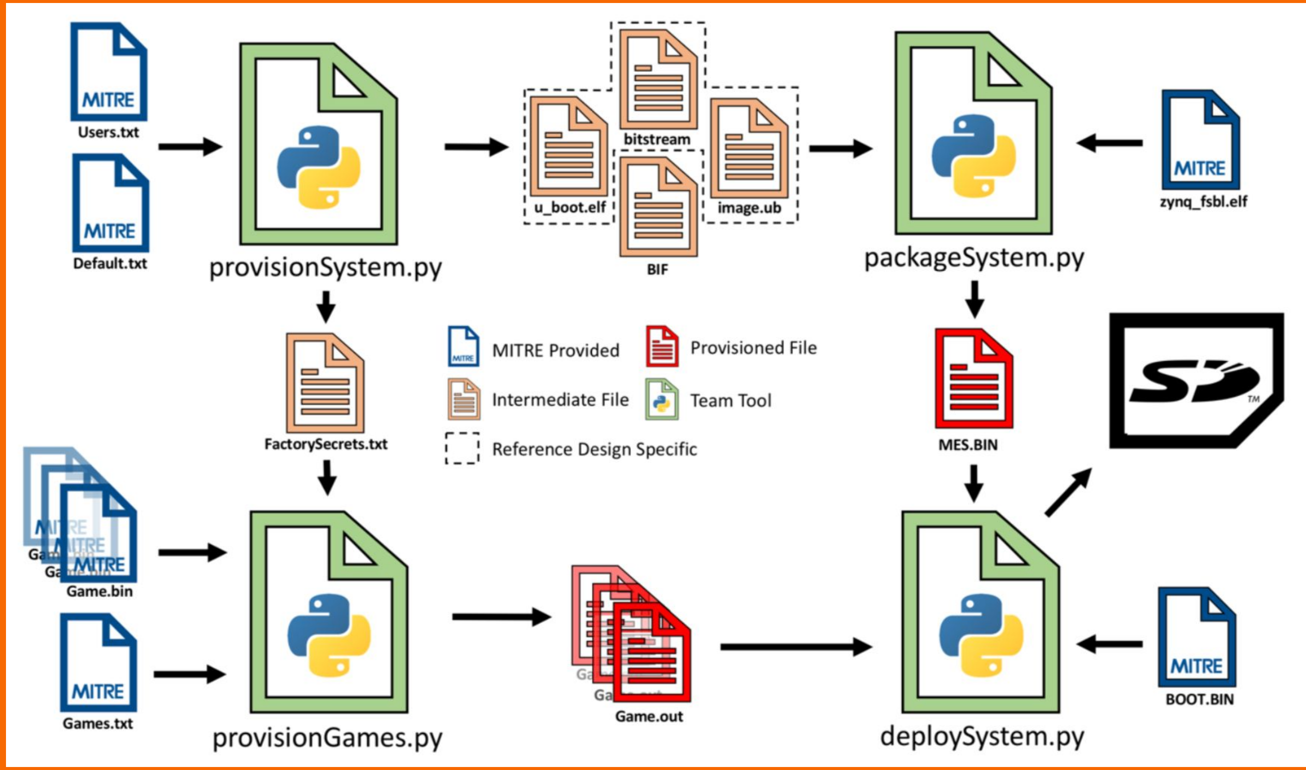
- MESH user's passwords are salted and hashed
- Each entry of the game table is authenticated to prevent tampering
- Game binaries are encrypted and authenticated
  - Game's HMAC is checked before installation and play
- PetaLinux networking and root account is disabled

HW

- Disabled all unused peripherals (Ethernet, SPI, I2C)
  - Reduces attack entry points
- Removed unused hw components (HDMI IN, GPIO…)

Our Secure Design Continued

- ProvisionSystem.py - Generates HMAC and encryption keys and generates Header file used in MESH
- ProvisionGames.py - Encrypts games with key from FactorySecrets.txt, adds metadata, and generates HMAC of the entire file



Our Secure Design v2.0 - SW

What things did you want to do, but didn’t have time for?

- Utilize SCrypt or Argon2 for password hashing

What things did you think of after-the-fact?

- Keys were plaintext in DDR so our design was very vulnerable to memory dumps

What mistakes did you make that you realized during the attack phase?

- Should have used RSA for signatures
- We weren’t randomizing memory addresses within U-Boot or PetaLinux
- Games should have remained encrypted while loading into PetaLinux

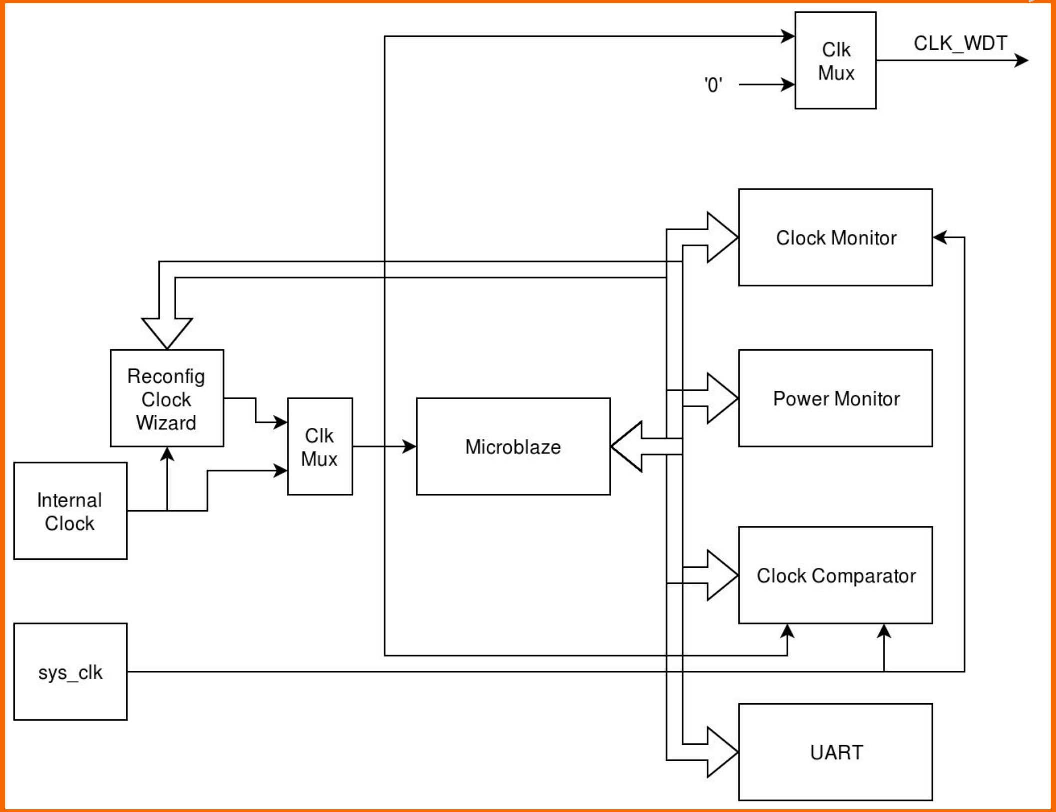
Our Secure Design v2.0 - HW

Planned Implementation

- Implement a hardware monitoring system to monitor the clock and power on the SoC.

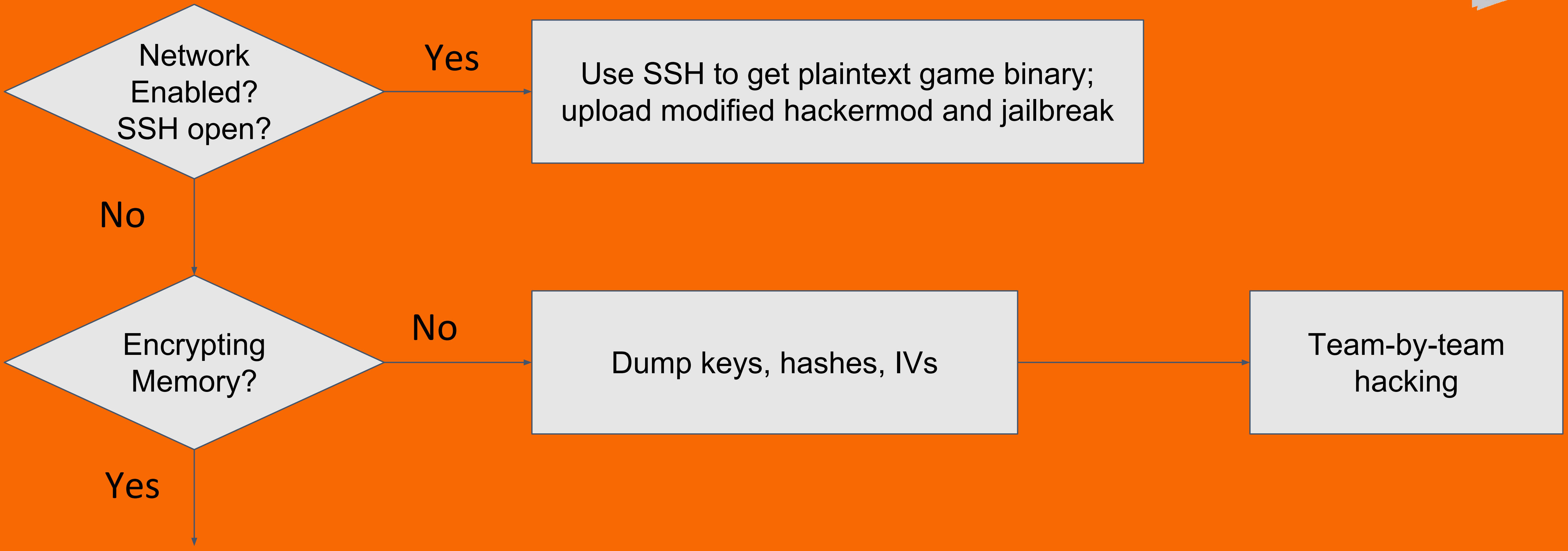
Post attack revised implementation

- Place all sensitive information into BRAM to prevent DDR dumping (BRAM clears on resets)
- Encrypt main memory by routing it through encryption/decryption block in PL for data protection



HW Monitoring System Block Diagram

Attack Approaches We used



Attack Highlight #1: Design Flaw in Encryption

Team ***A*** used *bcrypt* to protect user PINs. If we brute-force a PIN using *hashcat* on GPUs, it may take up to 60 hours...

But, we found a design flaw in their game encryption, which enabled us to brute-force the PIN in minutes.

It turns out, they calculate a unique key for each user for game encryption

```
user_key = master_key XOR (user_name and user_pin)
```

master_key can be dumped from memory; *user_name* is known

Attack Highlight #1: Design Flaw in Encryption

Therefore, to brute-force *pinbypass*'s PIN, we do not have to crack *bcrypt*.

```
def bf_pinbypass(username, pin):
    sys_game_key = #key dumped from memory

    ret = msd_xor(sys_game_key, bytes("".join(username + pin), "utf 8"))
    return ret

def decryptPINBYPASS(pin, game_bin):
    cipher_game_key = game_bin[104:136]

    aes = AES.new(bf_pinbypass("pinbypass", pin), AES.MODE_GCM, nonce=game_bin[72:88])
    plain_game_key = aes.decrypt(cipher_game_key)

    game_bin_for_decrypt = game_bin[200:216]

    aes2 = AES.new(plain_game_key, AES.MODE_GCM, nonce=game_bin[136:152])
    plain_game_bin = aes2.decrypt(game_bin_for_decrypt)

    if plain_game_bin[:4] == b'\x7f\x45\x4c\x46':
        print("decryption matches")
        print(pin)
        exit(0)
```

Attack Impacts and Countermeasures

- What is the impact of this attack?
  - We can break a system very quickly
- Suggested Fix:
  - Unique key for each user is a good idea. But, those keys should be independent.

Attack Highlight #2: Brute-force Another PIN

Team ***B*** used *public-key crypto* to protect user PINs. It takes 1 second to try a PIN on a core.

We dumped the ciphertext of PIN from memory, then brute-forced the PIN on a AWS instance with 96 cores.

A design flaw in their system and luck helped us brute-force the PIN in 10 hrs.

The first digit doesn't matter; The second digit for us was 0. We cracked the PIN by just trying 5% of the PIN space!!

```c
// pinbypass pin in memory
char* pinMem = "2583481e128bcb2692c1a2fbe16192f0a06e5b9aee62502f5fd629cf302d8886af6fa9

int main(int argc, char** argv)
{
    setbuf(stdout, NULL);
    int pin = atoi(argv[1]);

    for (; pin < 99999999; pin++)
      {
            char pin_string[9] = {0};
            sprintf(pin_string, "%08d", pin);
            //printf("%s\n", pin_string);

            if (pin % 1000 == 0)
                printf("Trying %d: \n", pin);

            if (verifyPinECDH(publickey, pin_string, salt, pinMem))
            {
                fprintf(stdout, "Here you go: %d\n", pin);
                exit(0);
            }
      }
}
```

Other Attacks

- Quad SPI hot swap
- FTP (anonymous root access)

What We Learned

- Dumping DDR is a very effective attack vector
    - Wasn't cleared after soft reboot
    - ASLR would have made this much more difficult
- Protecting secrets and IP is essential
    - Only plaintext in memory when in-use
    - Or use specialized memory when available

Questions?

Backup