TrustedAP

Using the Ethereum Blockchain to Mitigate the Evil Twin Attack

Phil Fox, pcfox@buffalo.edu
Department of Computer Science and Engineering, SUNY University at Buffalo

Introduction

Evil Twin Attacks (ETAs) are only partially mitigated to-date despite an extensive, established attack history. TrustedAP leverages the smart contract and trust layers of the Ethereum blockchain for mitigating residual ETA vulnerabilities, aiming for complete practical mitigation[1].

Background

The ETA is a low-to-medium-skill wireless network Computer Network Attack (CNA) wherein a threat agent (attacker) uses hardware masquerading as (i.e., spoofing) an *intended* Access Point (AP) for unsuspecting end-users' client devices. ETA success enables attackers' straightforward network traffic eavesdropping, exposure to malicious captive portals, and unprotected attacker network pivoting onto client devices. ETAs are partially mitigated by ensuring *all* client web-traffic is made over VPN or TLS/SSL connections, but the above vulnerabilities persist in even a temporary lapse of strong web or network security.

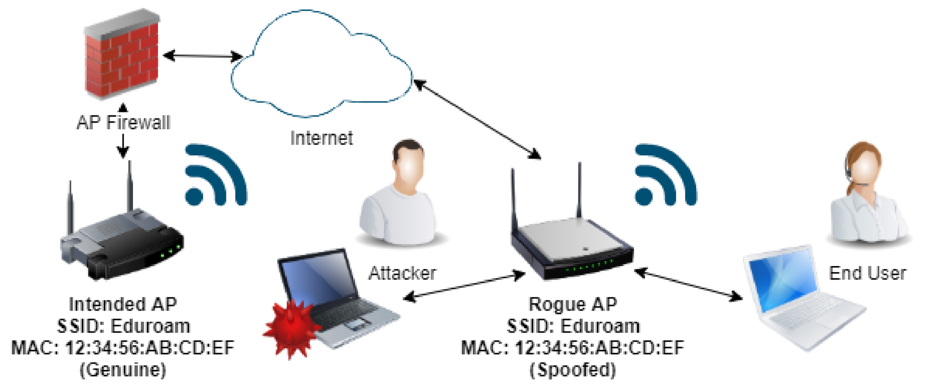

Figure 1: ETA diagram

Proposals for mitigating ETAs include establishing *off-network*, physical trust from client to AP (e.g., NFC connections between each)[2]. Alternative proposals include establishing on-network by (e.g., facilitating advanced connection protocols via a centralized certificate authority)[3]. TrustedAP aims to establish sufficient trust while offering novel affordability and ease-of-use for all parties.

Design

Trusted AP uses on and off-chain connections between end-users' client devices and wireless APs enforcing integrity and establishing trust. Fig. 2 illustrates the interactions between on and off-blockchain applications therein.

ESTABLISH AN OWNERSHIP CHAIN

A. Contract deployers, device managers, and APs are registered on-chain with an immutable custody record.

PASS, VERIFY, AND VALIDATE MESSAGES

B. Client devices post and retrieve challenge **hashes** on the blockchain, and APs post hashes of their responses for **verification**.
C. Client devices **encrypt** and send **challenges** over network connections and APs **respond** in-kind.
D. Client devices and APs **verify** messages received against logged blockchain hashes and **validate** decrypted message contents against network data.
E. The client considers the body of interactions and **decides** whether to remain connected or disconnect.

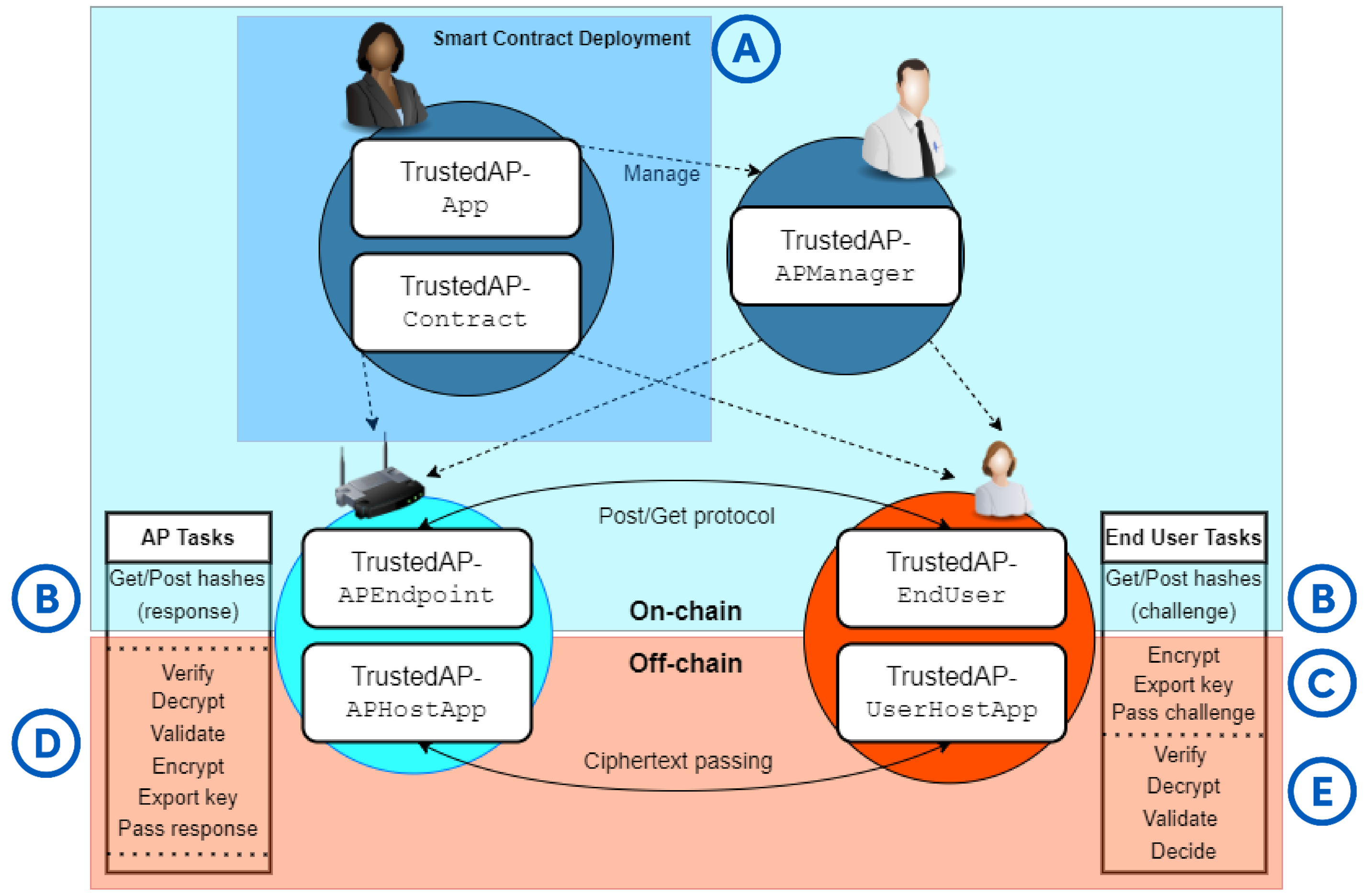

Figure 2: TrustedAP Application and Interaction Diagram

ENCRYPTION PRIMITIVES FOR OFF-CHAIN OPERATIONS

Messages passed between clients and APs are hashed on-chain using the **SHA3 algorithm** and passed off-chain using **RSA encryption** for integrity preservation. The Fig. 3 pseudo code explains relevant exchange mechanisms:

- **Client to AP exchange (Challenge/Resolve):**

```
ClientChallengesAP()
    NetToken_c ← ParseNetworkInfo()
    (CK_public, CK_private) ← GenerateRSAKeys()
    ValidPack_c ← Enc((CK_private, NetToken_c)
    PostToBlockchain(SHA3(CK_public), SHA3(ValidPack_c))
    SendToAP(CK_public, ValidPack_c)
    Listen()
    if noresponse then
        Disconnect()
    else
        ClientResolvesAP()
```

```
APResolvesClient(CK_public, ValidPack_c)
    VerifyOnChain(SHA3(CK_public), SHA3(ValidPack_c))
    if mismatch then
        break
    NetToken_c ← Dec(CK_public, ValidPack_c)
    Validate(ParseNetworkInfo(),NetToken_c)
    if mismatch then
        break
    else
        APRespond()
```

- **AP to Client exchange (Response/Resolve):**

```
APRespond()
    NetToken_a ← ParseNetworkInfo()
    (AK_public, AK_private) ← GenerateRSAKeys()
    ValidPack_a ← Enc((AK_private, NetToken_a)
    PostToBlockchain(SHA3(AK_public), SHA3(ValidPack_a))
    SendToClient(AK_public, ValidPack_a)
```

```
ClientResolvesAP(AK_public, ValidPack_a)
    VerifyOnChain(SHA3(AK_public), SHA3(ValidPack_a))
    if mismatch then
        Disconnect()
    NetToken_a ← Dec(AK_public, ValidPack_a)
    Validate(ParseNetworkInfo(),NetToken_a)
    if mismatch then
        Disconnect()
    else
        AlertTrusted()
```

Figure 3: TrustedAP off-chain exchange pseudocode

Implementation

The current version of TrustedAP uses the following programming languages and software:

- **Solidity (Language):** Encodes Ethereum blockchain smart contracts for on-chain operations
- **JavaScript, HTML, web3.js:** Establishes a UI and interfaces it with smart contract functions
- **Truffle:** Compiles and executes Solidity code for emulated and actual Ethereum (incl. testnet) blockchain interaction
- **Node.js:** Manages libraries for web3, encryption, etc. and provides local webservers for relevant endpoints
- **MetaMask:** Maintains wallets for Ethereum and testnet ETH for on-chain operational gas and payable functions

Evaluation and Future Work

TrustedAP's proof of concept is successful, and it relies exclusively on well-known, cryptographically strong software libraries. Future work includes improving the features named thereafter:

- **Proof of concept:** The on-chain core and UI of TrustedAP is operational with supporting source code[1].
- **Cryptographic strength:** TrustedAP uses Ethereum blockchain DSA signing and OpenSSL RSA and SHA3 library functions[1].
- **Resistance to offensive attack:** The *uniqueness* of network data passed and validated between clients and APs determines threat capability thresholds for future ETAs against TrustedAP, e.g.:
    - User specified/random key: Man in the Middle/network replay attack (MitM)+ETA (early version)
    - Destination IP+Mac address: 2-way MitM (requires rogue AP *and* client)+ETA (current version)
- **Liveness, Safety, and Timing:** Can TrustedAP achieve a 0% false negative rate (connect to an untrusted AP) *and* a low-percentage false negative rate (refuse a trusted AP connection) under a threshold that out-competes attacker software?
- **Transparency and affordability**: Can TrustedAP achieve a low-impact end-user UI and hands-free AP operation while remaining affordable to parties that implement it?

Acknowledgements

This research is supported in part by the National Science Foundation under Grant No. DGE – 1754085.

References

1. P. Fox. 2021. TrustedAP. GitHub repository. https://github.com/pcfox-buf/TrustedAP
2. A. Matos, D. Romão, and P. Trezentos. 2012. Secure hotspot authentication through a Near Field Communication side-channel. In 2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). 807–814. https://doi.org/10.1109/WiMOB.2012.6379169
3. O. Nakhila, E. Dondyk, M. F. Amjad, and C. Zou. 2015. User-side Wi-Fi Evil Twin Attack detection using SSL/TCP protocols. In 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC). 239–244. https://doi.org/10.1109/CCNC.2015.7157983