

Owning the LAN in 2018

Defeating MACsec and 802.1x-2010

DEF CON 26

Gabriel “solstice” Ryan



About: Digital Silence

Denver-based security consulting firm:

- Penetration testers who give a !@# \$
- Red teaming
- Penetration Testing
- Reverse-engineering / advanced appsec / research

Twitter (for those of you who are into that sort of thing): [@digitalsilence](https://twitter.com/digitalsilence)



About: Gabriel Ryan (a.k.a. solstice)

Co-Founder / Senior Security Assessment Manager @ Digital Silence

- Former Gotham Digital Science, former OGSystems
- Red teamer / Researcher / New Dad

Twitter: [@s0lst1c3](https://twitter.com/s0lst1c3)

LinkedIn: [ms08067](https://www.linkedin.com/in/ms08067)

Email: gabriel@digitalsilence.com



DIGITAL SILENCE

Introduction to 802.1x



What is 802.1x?

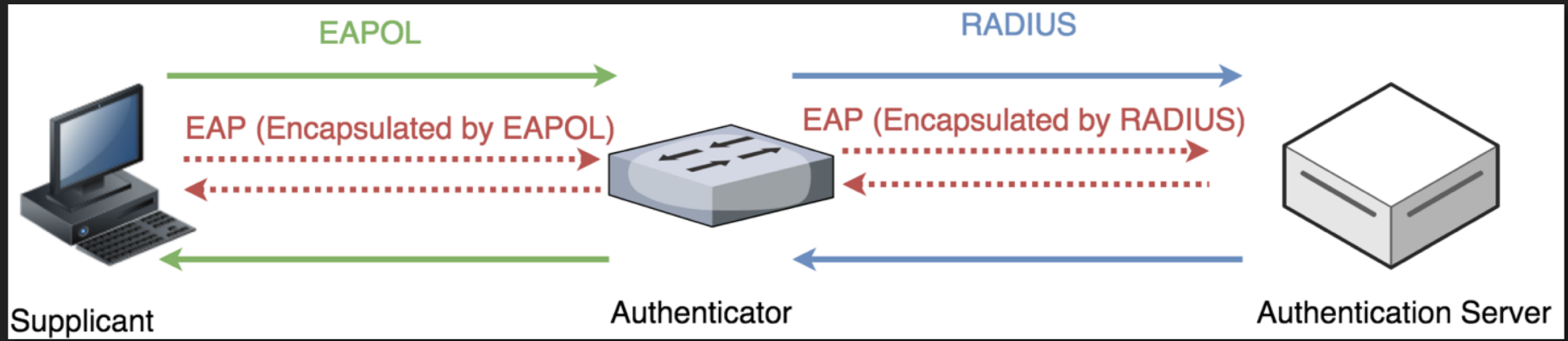
- Authentication protocol
- Used to protect a local area network (LAN) or wireless local area network (WLAN) with rudimentary authentication



802.1.x defines an exchange between three parties:

- **supplicant** – the client device that wishes to connect to the LAN [1][2][9]
- **authenticator** – a network device such as a switch that provides access to the LAN [1][2][9]
- **authentication server** – a host that runs software that implements RADIUS or some other Authorization, Authentication, and Accounting (AAA) protocol [1][2][9]





- authenticator can be thought of as a gatekeeper
- supplicant connects to a switch port and provides the authenticator with its credentials ^{[1][2][9]}
- authenticator forwards credentials to the authentication server ^{[1][2][9]}
- Authentication server validates the credentials, and either allows or denies access the network ^{[1][2][9]}

802.1x is (typically) a four step sequence:

1. Initialization
2. Initiation
3. EAP Negotiation
4. Authentication

[1][2][9]



Ports have two states:

- Authorized – traffic is unrestricted
- Unauthorized – traffic is restricted to 802.1x

[1][2][9]



DIGITAL SILENCE

Step 1: Initialization

1. Supplicant connects to switch port, which is disabled
2. Authenticator detects new connection, enables switch port in unauthorized state

[1][2][9]



Step 2: Initiation

1. (optional) Supplicant sends EAPOL-Start frame [1][2][9]
2. Authenticator responds with EAP-Request-Identity frame [1][2][9]
3. Supplicant sends EAP-Response-Identity frame (contains an identifier such as a username) [1][2][9]
4. Authenticator encapsulates EAP-Response-Identity in a RADIUS Access-Request frame and forwards it to Authentication Server [1][2][9]

Step 3: EAP Negotiation

Long story short:
supplicant and
authentication
server haggle until
they decide on an
EAP method that
they're both
comfortable with.

[1][2][9]



DIGITAL SILENCE

Step 4: Authentication

- Specific details of how authentication should work are dependent on the EAP method chosen by the authentication server and supplicant [1][2][9]
- Always will result in a EAP-Success or EAP-Failure message [1][2][9]
- Port is set to authorized state if EAP-Success, otherwise remains unauthorized [1][2][9]

What is EAP?



Extensible Authentication Protocol (EAP):

It's an authentication *framework*:

- Not really a protocol, only defines message formats
- Individual EAP implementations are called "EAP methods"
- Think of it as a black box for performing authentication

Notable EAP methods...



EAP-MD5



EAP-PEAP



I KNOW IT'S SOAP



BUT IT LOOKS DELICIOUS

quickmeme.com



DIGITAL SILENCE

EAP-TLS



Brief History of Wired Port Security



Brief History of Wired Port Security

- 2001** – the 802.1x-2001 standard is created to provide rudimentary authentication for LANs ^[1]
- 2004** – the 802.1x-2004 standard is created as an extension of 802.1x-2001 to facilitate the use of 802.1x in WLANs extended 802.1x-2001 for use in WLAN ^[2]

Brief History of Wired Port Security

- 2005** — Steve Riley demonstrates that 802.1x-2004 can be bypassed by inserting a hub between supplicant and authenticator [3]
- Interaction limited to injecting UDP packets (TCP race condition) [4]

Brief History of Wired Port Security

2011 – “Abb” of Gremwell Security creates Marvin: [5]

- Bypasses 802.1x by introducing rogue device directly between supplicant and switch [5]
- No hub necessary: rogue device configured as a bridge [5]
- Full interaction with network using packet injection [5]



Brief History of Wired Port Security

2011 – Alva Duckwall's 802.1x-2004 bypass: [4]:

- Transparent bridge used to introduce rogue device between supplicant and switch [4]
- No packet injection necessary: network interaction granted by using iptables to source NAT (SNAT) traffic originating from device [4]
- More on this attack later...



Brief History of Wired Port Security

2017 – Valérien Legrand creates Fenrir: ^[6]:

- Works similarly to Duckwall's tool, but implements NATing in Python using Scapy (instead of making calls to iptables / arptables / ebtables) ^[6]
- Modular design, support for responder, etc...



Improvements to Bridge-Based Bypass Techniques



Let's look at Duckwall's 802.1x bypass more closely...

- Uses transparent bridge to silently introduce rogue device between supplicant and authenticator ^[4]
- Network interaction achieved by using iptables to source NAT (SNAT) traffic originating from device ^[4]
- Hidden SSH service created on rogue device by forwarding traffic to the supplicant's IP address on a specified port to bridge's IP address on port 22 ^[4]



Linux kernel will not forward EAPOL packets over a bridge. Existing tools deal with this problem by either:

- patching the Linux kernel
- Relying on high level libraries such as Scapy

Problems with both of these approaches:

- Relying on Kernel patches can become unwieldy: no publicly available Kernel patches for modern kernel versions
- Relying on high level tools such as Scapy can make the bridge slow under heavy loads ^[17]_[18]

Fortunately, the situation has dramatically improved since Duckwall's contribution:

```
65     os.system('ifconfig %s down' % self.name)
66
67     def enable_8021x_forwarding(self):
68
69         os.system('echo 8 > /sys/class/net/%s/bridge/group_fwd_mask' % self.name)
70
71     def enable_ip_forwarding(self):
72
73         os.system('echo 1 > /proc/sys/net/ipv4/ip_forward')
```

- as of 2012, EAPOL bridging can be enabled using the proc file system ^[11]
- that means no more patching :D ^[11]



Improvement: Support for Side Channel Interaction

When Duckwall created his original 802.1x bypass, he had to figure out how to provide the attacker with access to the rogue device:

- The year was 2011 – cellular modems were unsophisticated, slow, and expensive
- Solution: create hidden SSH service



Problems with this approach:

- Relies on assumption that egress filtering can be bypassed
- Relies on pushing traffic through the target network, creating an opportunity for detection



Our updated implementation:

- Relies on a side channel interface to provide attacker with connectivity

We had to add / modify some firewall rules to get it to work, but totally worth it.

```
[solstice@ops11-2] - [~/silentbridge] - [7528]  
[ ] $ ./silentbridge --create-bridge --upstream eno2 --phy eno1 --egress-port 443
```

```
100 print '[*] Initiate radio silence...'  
101  
102 # start dark - but make an exception for our side channel  
103 core.firewalls.iptables.allow_outbound(sidechannel, port=egress_port)  
104 core.firewalls.arptables.allow_outbound(sidechannel)  
105 core.firewalls.iptables.drop_all()  
106 core.firewalls.arptables.drop_all()  
107  
108 time.sleep(2)  
109  
110 print '[*] Bringing the bridge up with a non-routable IP...'  
111
```



Demo: Improvements to Bridge-Based Bypass Techniques



All traditional 802.1x bypasses (hub, injection, or bridge based) take advantage of the same fundamental security issues that affect 802.1x-2004:

[3][4][6][7]

- The protocol does not provide encryption
- The protocol does not support authentication on a packet-by-packet basis

Introduction to MACsec and 802.1x-2010



These security issues are addressed in 802.1x-2010, which uses MACsec to provide: [7]

- Layer 2 encryption performed on a hop-by-hop basis
- Packet-by-packet integrity checks

Support for hop-by-hop encryption particularly important: [7]

- Protects against bridge-based attacks
- Allows network administrators with a means to inspect data in transit

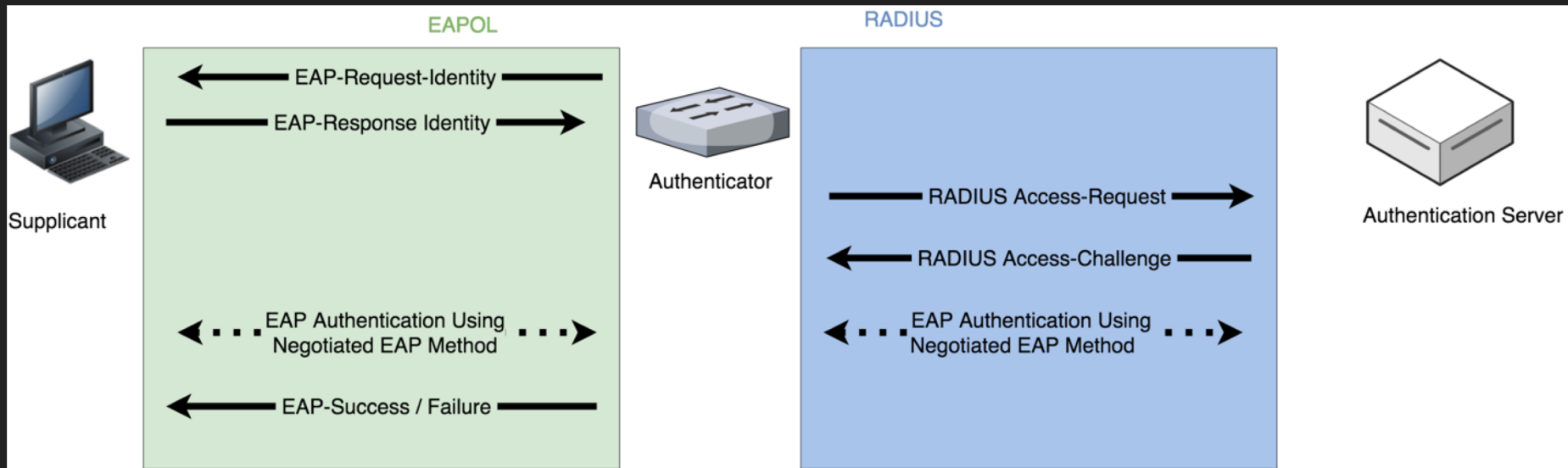


The 802.1x-2010 protocol works in three stages:

[7][8][9]

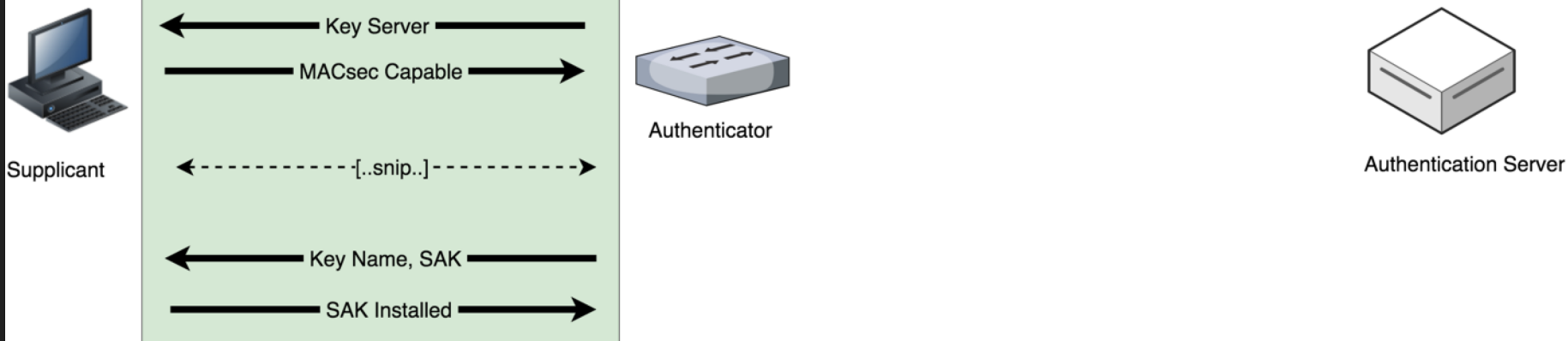
1. Authentication and Master Key Distribution
2. Session Key Agreement
3. Session Secure



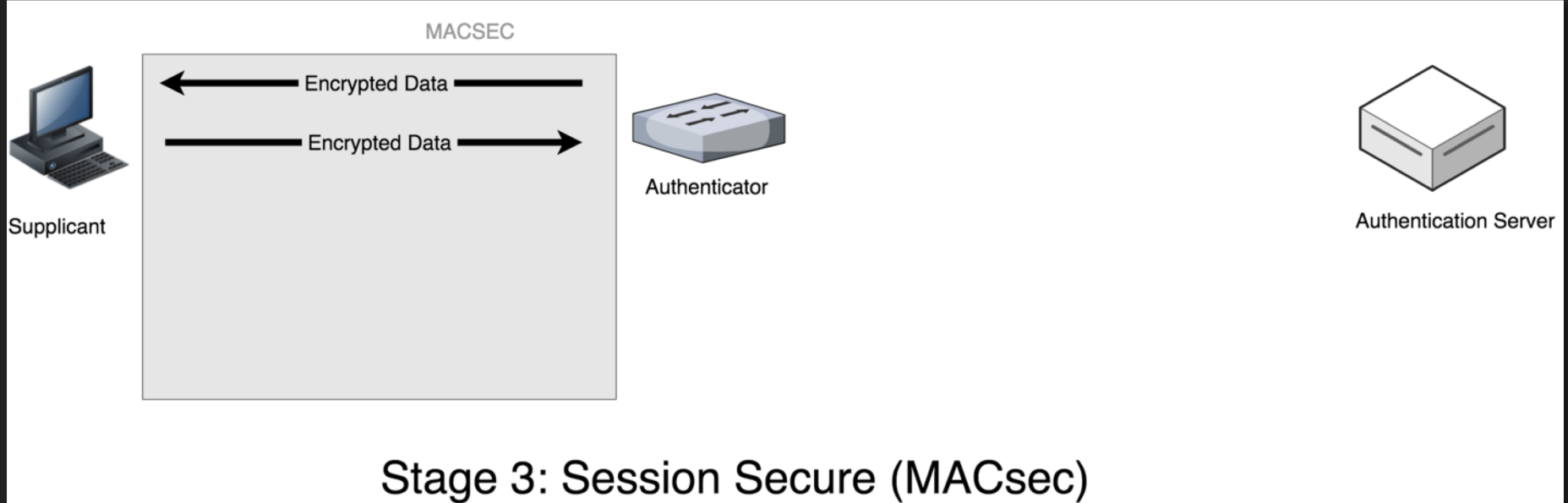


Stage 1: Authentication (802.1x)

EAPOL-MKA



Stage 2: Session Key Agreement (MKA)



Things to think about...

“IEEE Std 802.11 specifies media-dependent cryptographic methods to protect data transmitted using the 802.11 MAC over wireless networks. Conceptually these cryptographic methods can be considered as playing the same role within systems and interface stacks as a MAC Security Entity.” – IEEE 802.1x-2010 Standard – Section 6.6 ^[9]



Parallels between MACsec and WPA



2003 – WPA1 is released

Hop-by-hop Layer 2 Encryption:

- access point to station

Authentication provided by:

- Extensible Authentication Protocol (EAP)
- Pre-Shared Key (as a fallback / alternative)

Shift of focus due to WPA

Injection-based Attacks no longer possible due to Layer 2 encryption

Focus shifts to attacking authentication mechanism

- Pre-Shared Key (PSK) – WPA Handshake Capture and Dictionary Attack
- EAP – Rogue AP attacks against weak EAP methods



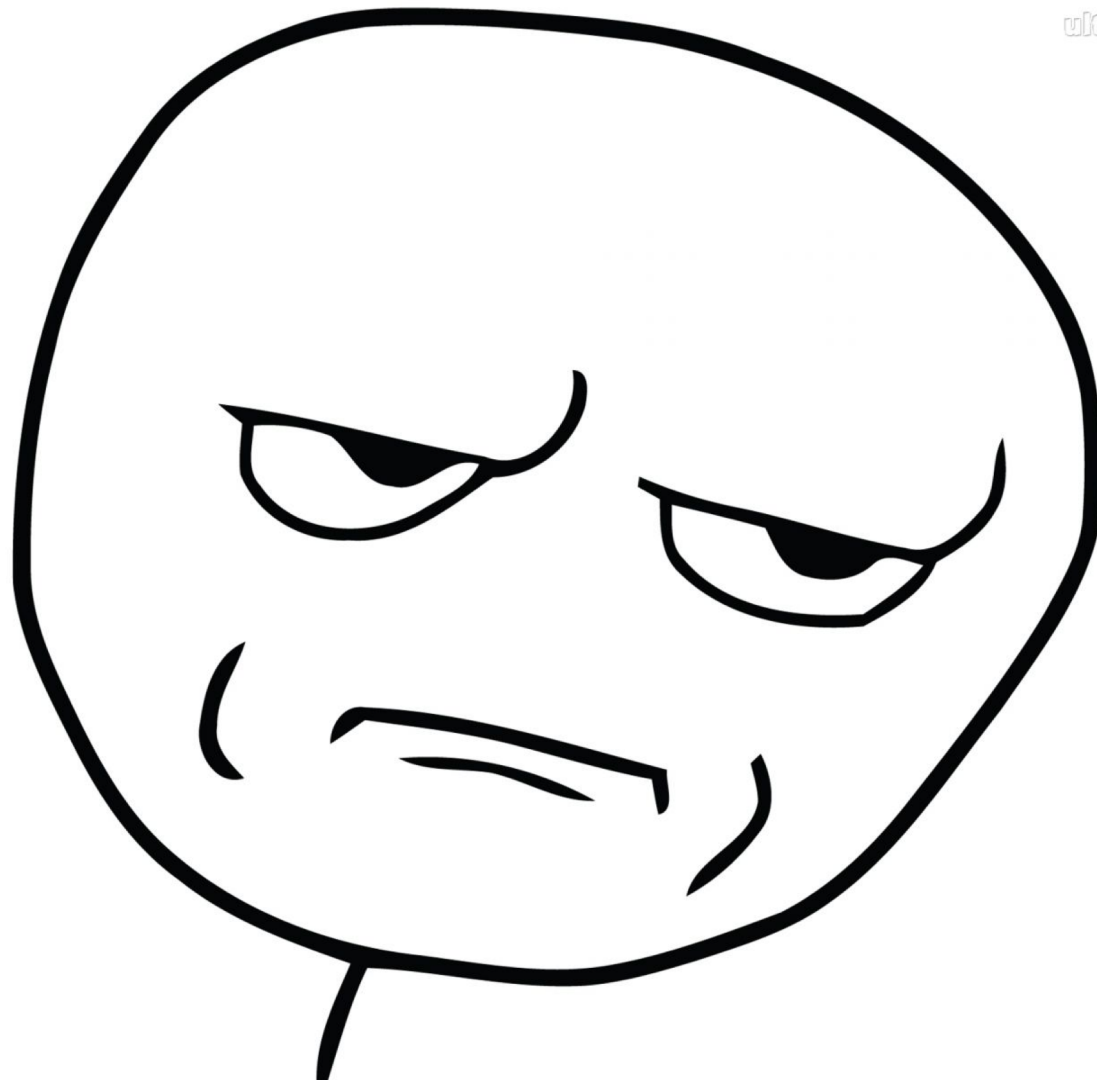
2010 – 802.1x-2010 is released

Hop-by-hop Layer 2 Encryption using MACsec:

- device to switch / switch to switch

Authentication provided by:

- Extensible Authentication Protocol (EAP)
- Pre-Shared Key (as a fallback / alternative)



Shift of focus due to MACsec

Bridge and injection-based attacks no longer possible due to Layer 2 encryption

First thing that comes to mind: try attacking the authentication mechanism:

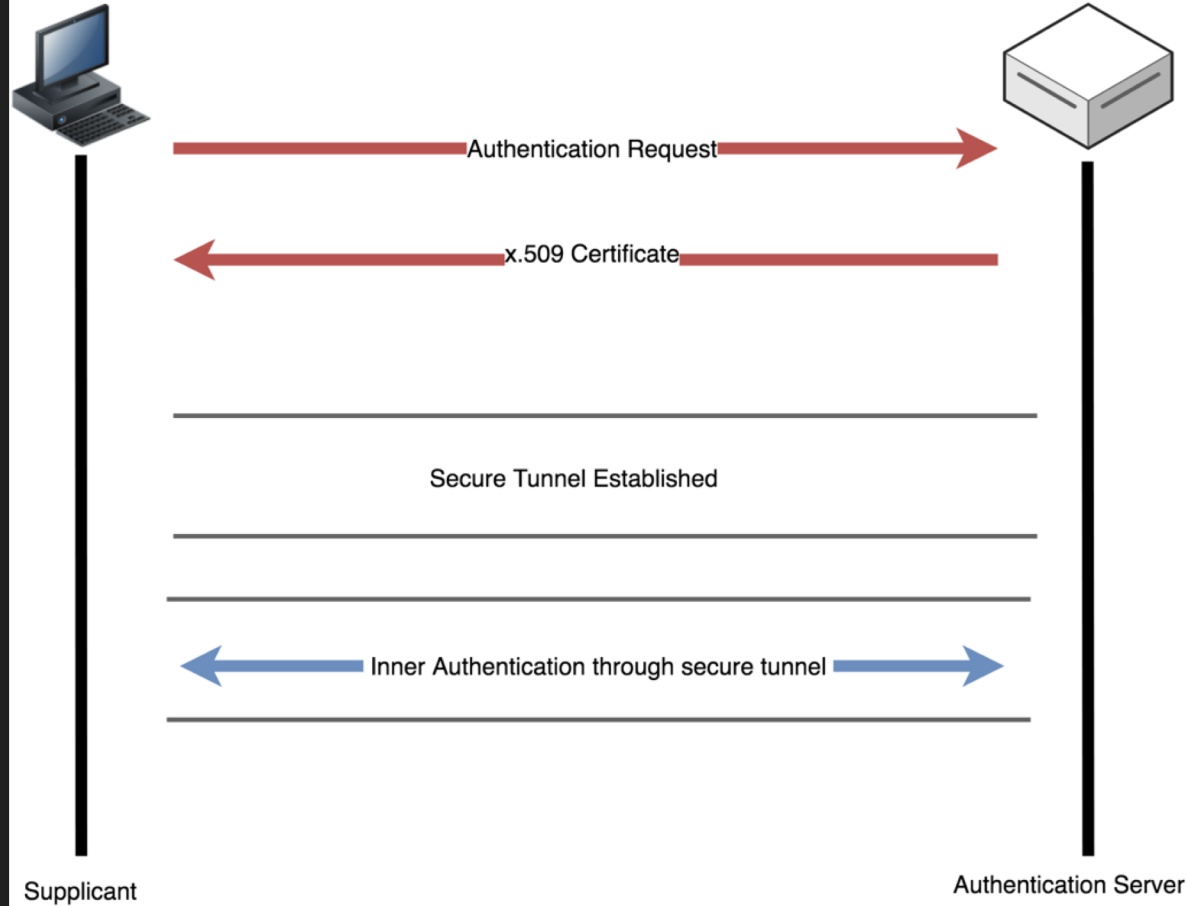
- Pre-Shared Key (PSK) – some kind of dictionary attack??? (still working on that)
- EAP – attacks against weak EAP methods (main takeaway of this talk)



Attacks Against WPA2-EAP



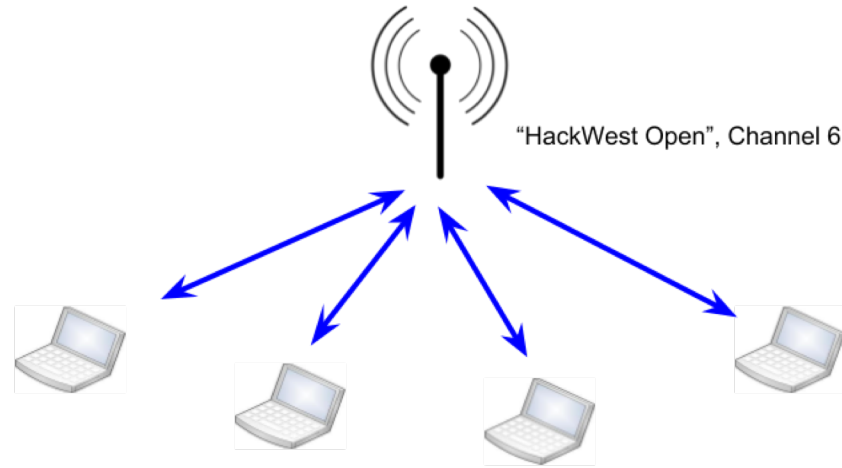
Core Stages of EAP-PEAP
(Forwarded Between Authentication Server and Supplicant by Authenticator)



EAP-PEAP: Security Issues

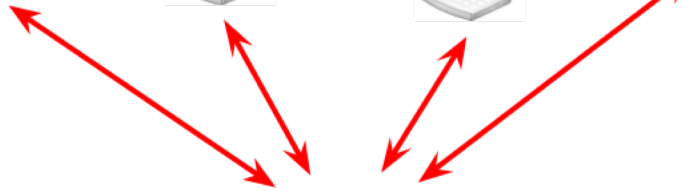
Brad Antoniewicz and Josh Wright in 2008: ^[13]

- attacker can use a rogue access point attack to force the supplicant to authenticate with a rogue authentication server ^{[13][20]}





"HackWest Open", Channel 6



Rogue Access Point:
"HackWest Open", Channel 6



DIGITAL BUSINESS

EAP-PEAP: Security Issues

Brad Antoniewicz and Josh Wright in 2008: ^[13]

- So long as the supplicant accepts the certificate presented by the attacker's authentication server, the supplicant will transmit an EAP challenge and response to the attacker ^{[13][21]}
- can be cracked to obtain a plaintext username and password ^{[13][21]}

EAP-PEAP: Security Issues

MS-CHAPv2 is the strongest Inner Authentication protocol available for use with EAP-PEAP and EAP-TTLS:

- vulnerable to a cryptographic weakness discovered by Moxie Marlinspike and David Hulton in 2012 ^[22]
- MS-CHAPv2 challenge and response can be reduced to a single 56-bits of DES encryption ^{[22][23]}
- The 56-bits can be converted into a password-equivalent NT hash within 24 hours with a 100% success rate using FPGA-based hardware ^{[22][23]}



Back to 802.1x-2010...



Most important takeaway about 802.1x-2010 (from an attacker's perspective): [7]

- It still uses EAP to authenticate devices to the network
- EAP is only as secure as the EAP method used




Supported EAP methods:

The 802.1x-2010 standard allows any EAP method so long as it: [7]

- Supports mutual authentication
- Supports derivation of keys that are at least 128 bits in length
- Generates an MSK of at least 64 octets

Plenty of commonly seen weak EAP methods that meet these requirements (EAP-PEAP, EAP-TTLS, etc).





**I THINK YOU SEE
WHERE THIS IS
GOING**

starz



DIGITAL SILENCE

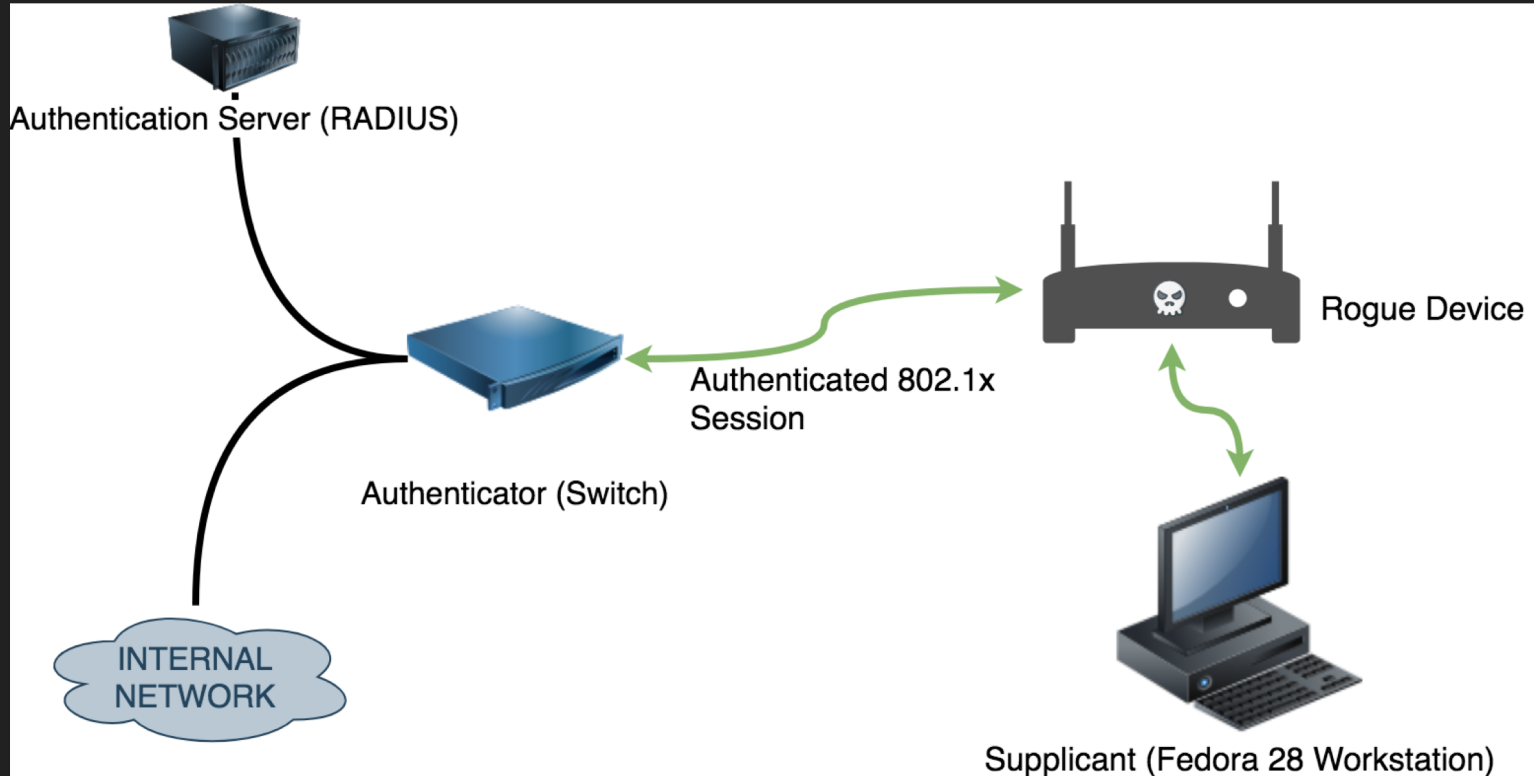
Defeating MACsec Using Rogue Gateway Attacks



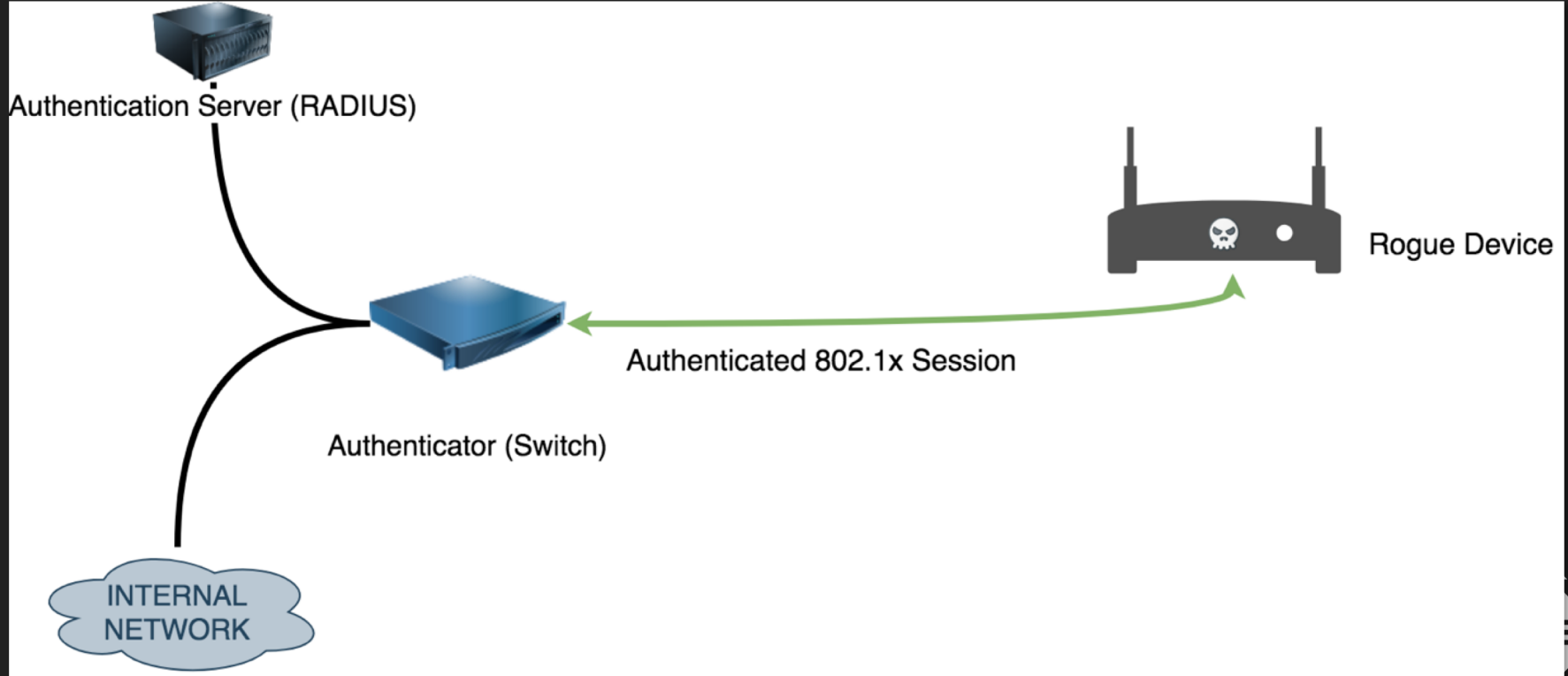
Goal: Rogue Gateway Attack

- Force the supplicant to authenticate with attacker's device
- Crack hashes, authenticate with the network

802.1x-2004: MITM style bypass



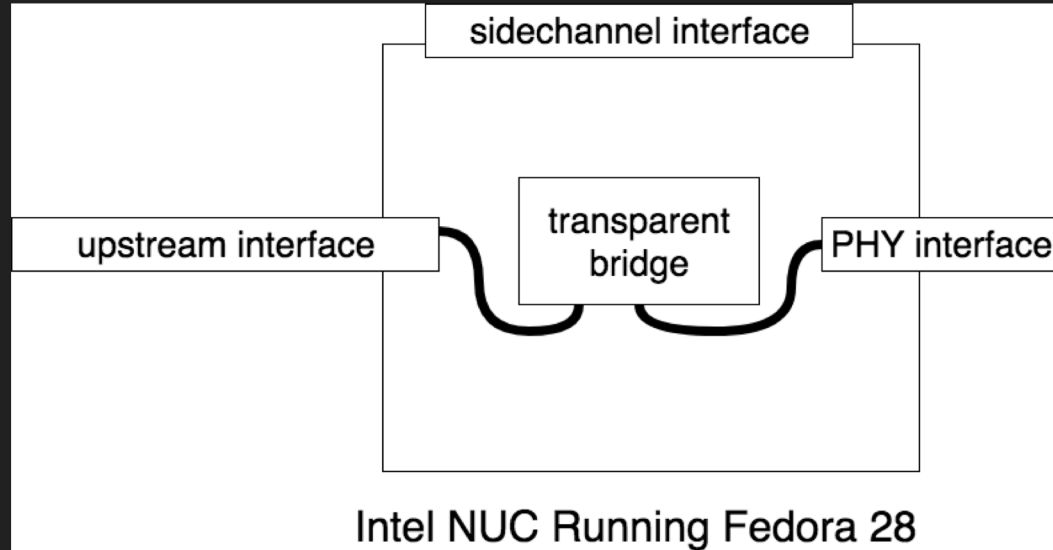
802.1x-2010: Direct Access



Let's build a rogue device...



Step 1: Device Core



Need a way to divert traffic to the rogue device....





DIGITAL SILENCE

Mechanical A/B Ethernet Splitters



FRONT



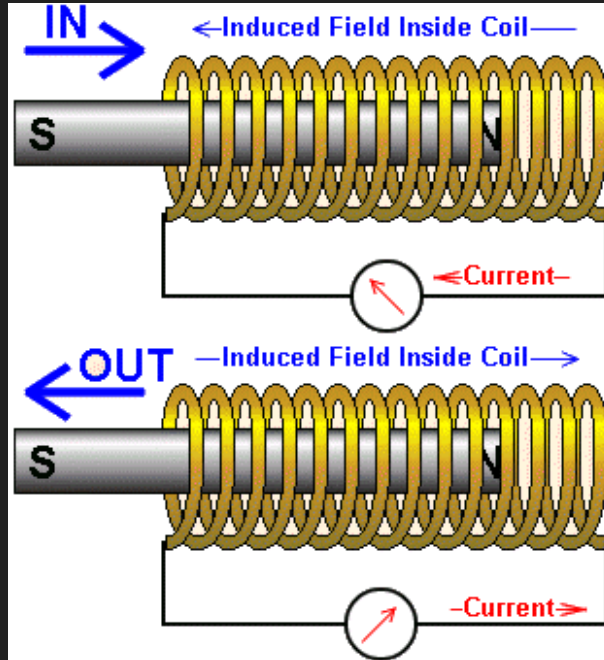
BACK

Need a way of manipulating the push switch:

- Using relays will lead to impedance issues
- Option B: use solenoids

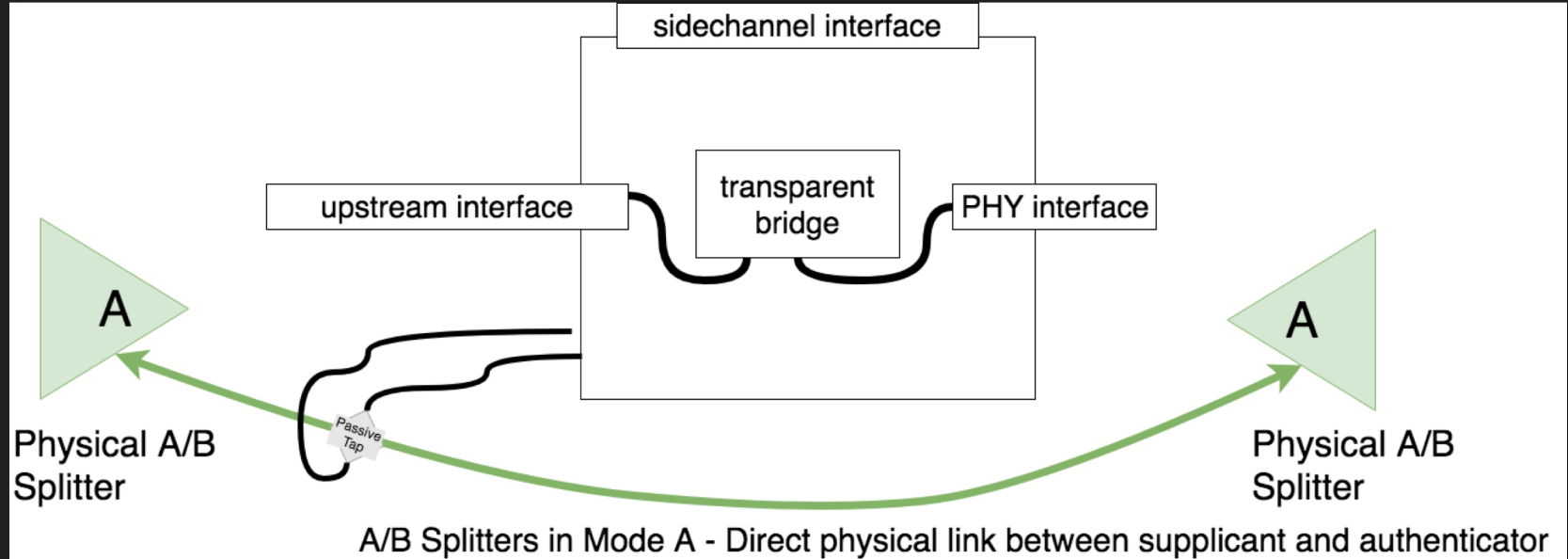
Solenoids:

Pull Solenoid

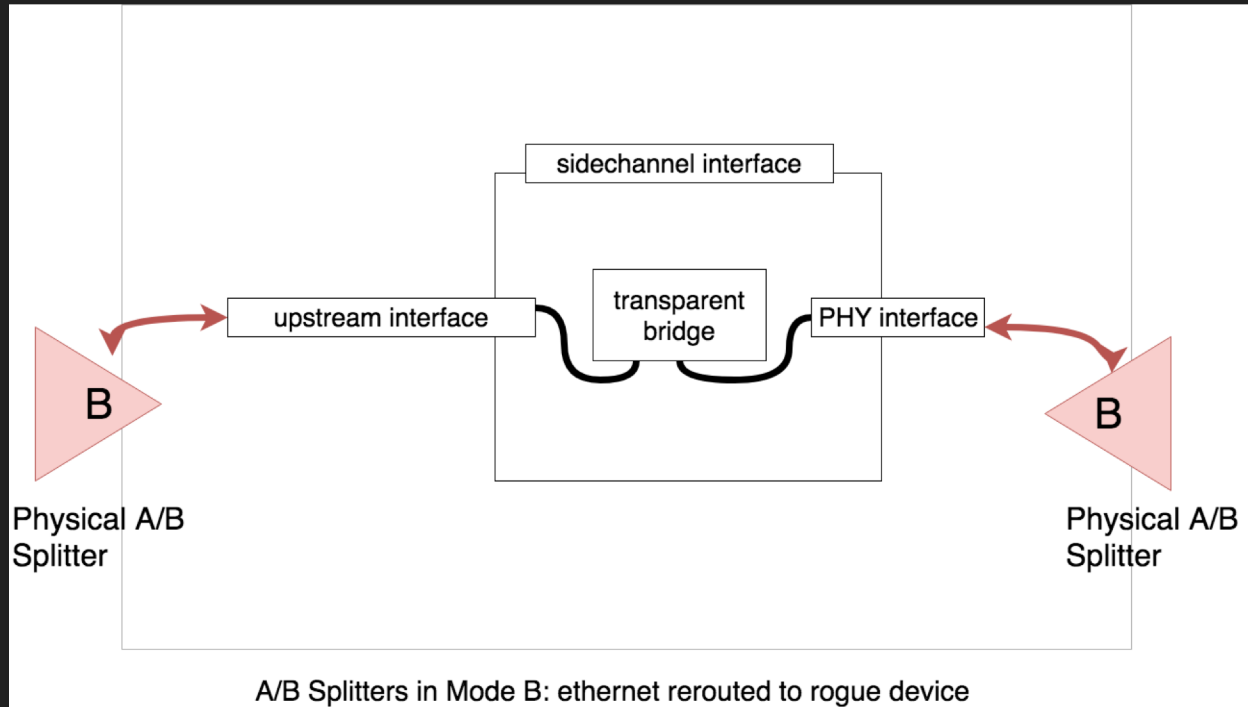


Push Solenoid

Mode A: Full bypass with passive tap



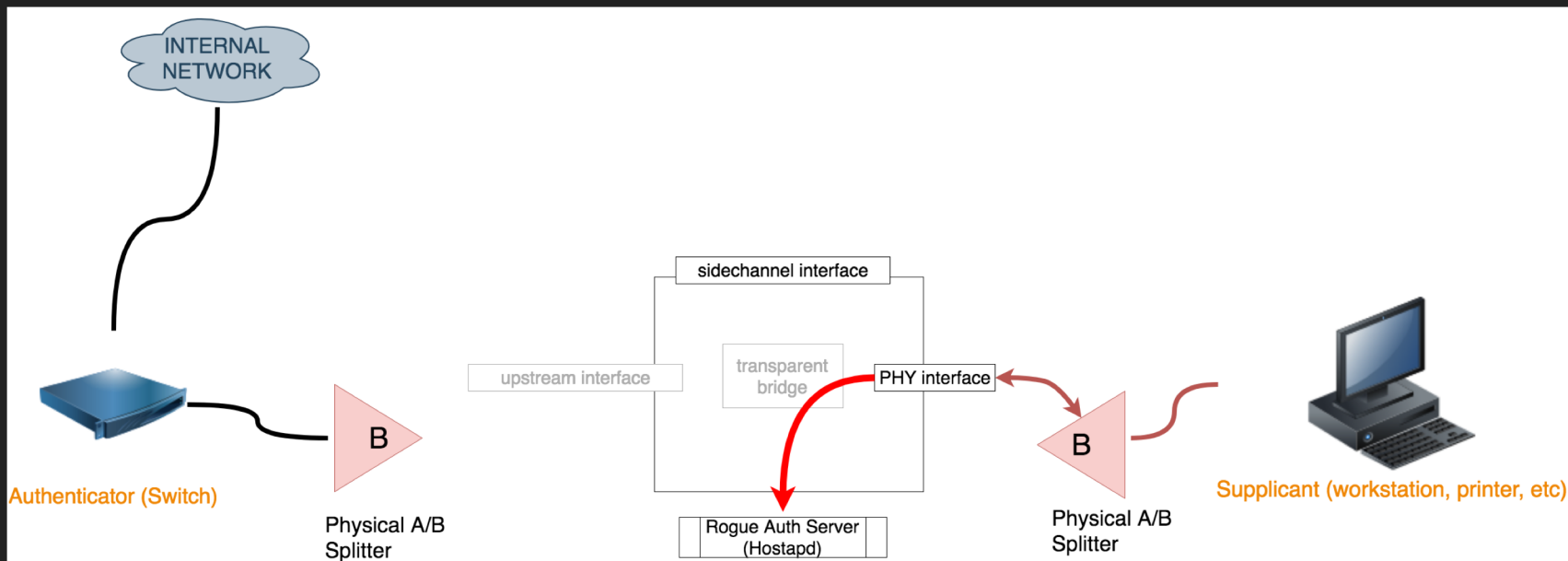
Mode B: Link is routed to upstream & PHY interfaces



Implementing the attack...



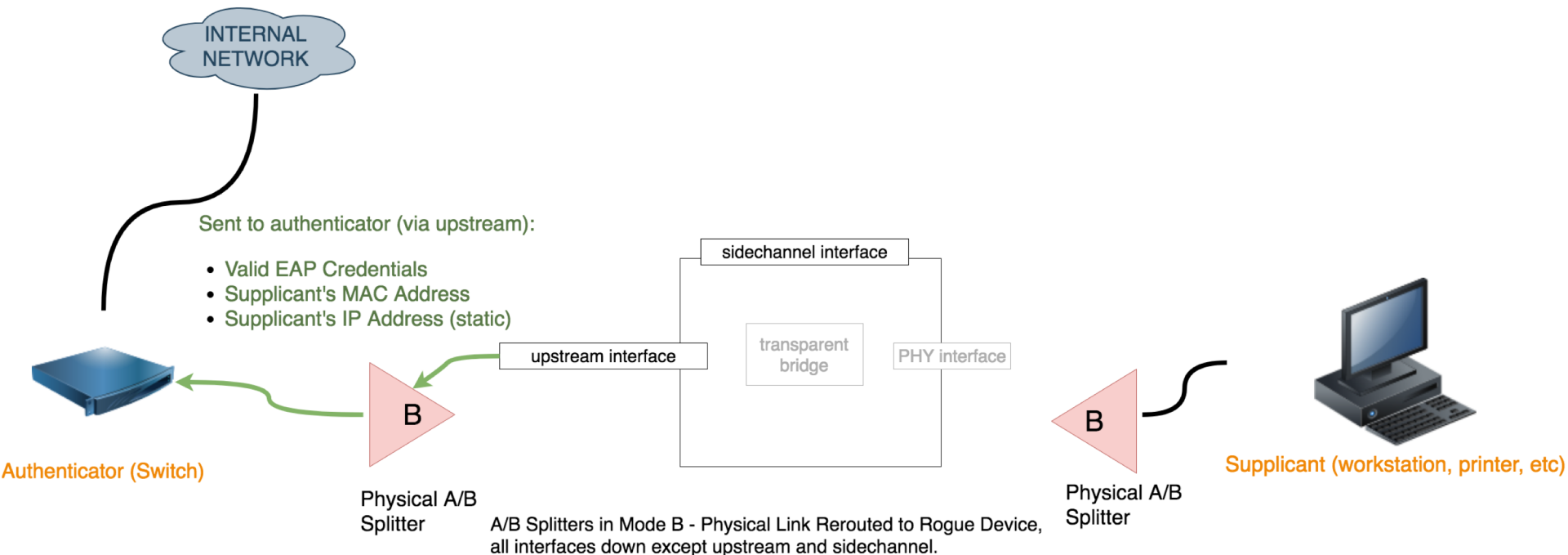
Step 1: Route supplicant to rogue auth server



A/B Splitters in Mode B, which routes physical link to rogue device.
All interfaces down except sidechannel and PHY.
Rogue auth server (hostapd) listening on PHY.



Step 2: Authenticate using stolen EAP credentials



Demo: Defeating MACsec Using Rogue Gateway Attacks



```
[root@localhost ~]# cd /dev/pts/1
[home/solstice/silentbridge]>
```

Wed 12:01



solstice@localhost:~

File Edit View Search Terminal Help

```
64 bytes from 8.8.8.8: icmp_seq=1280 ttl=119 time=14.9 ms
64 bytes from 8.8.8.8: icmp_seq=1281 ttl=119 time=13.8 ms
64 bytes from 8.8.8.8: icmp_seq=1282 ttl=119 time=13.7 ms
64 bytes from 8.8.8.8: icmp_seq=1283 ttl=119 time=14.3 ms
64 bytes from 8.8.8.8: icmp_seq=1284 ttl=119 time=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=1285 ttl=119 time=13.8 ms
64 bytes from 8.8.8.8: icmp_seq=1286 ttl=119 time=13.8 ms
64 bytes from 8.8.8.8: icmp_seq=1287 ttl=119 time=15.0 ms
64 bytes from 8.8.8.8: icmp_seq=1288 ttl=119 time=13.4 ms
64 bytes from 8.8.8.8: icmp_seq=1289 ttl=119 time=14.4 ms
64 bytes from 8.8.8.8: icmp_seq=1290 ttl=119 time=13.8 ms
64 bytes from 8.8.8.8: icmp_seq=1291 ttl=119 time=29.3 ms
64 bytes from 8.8.8.8: icmp_seq=1292 ttl=119 time=14.7 ms
64 bytes from 8.8.8.8: icmp_seq=1293 ttl=119 time=23.9 ms
64 bytes from 8.8.8.8: icmp_seq=1294 ttl=119 time=13.8 ms
64 bytes from 8.8.8.8: icmp_seq=1295 ttl=119 time=20.5 ms
64 bytes from 8.8.8.8: icmp_seq=1296 ttl=119 time=14.4 ms
64 bytes from 8.8.8.8: icmp_seq=1297 ttl=119 time=14.2 ms
64 bytes from 8.8.8.8: icmp_seq=1298 ttl=119 time=14.5 ms
64 bytes from 8.8.8.8: icmp_seq=1299 ttl=119 time=14.4 ms
64 bytes from 8.8.8.8: icmp_seq=1300 ttl=119 time=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=1301 ttl=119 time=14.1 ms
64 bytes from 8.8.8.8: icmp_seq=1302 ttl=119 time=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=1303 ttl=119 time=13.8 ms
64 bytes from 8.8.8.8: icmp_seq=1304 ttl=119 time=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=1305 ttl=119 time=13.3 ms
64 bytes from 8.8.8.8: icmp_seq=1306 ttl=119 time=14.2 ms
64 bytes from 8.8.8.8: icmp_seq=1307 ttl=119 time=14.3 ms
64 bytes from 8.8.8.8: icmp_seq=1308 ttl=119 time=14.1 ms
```

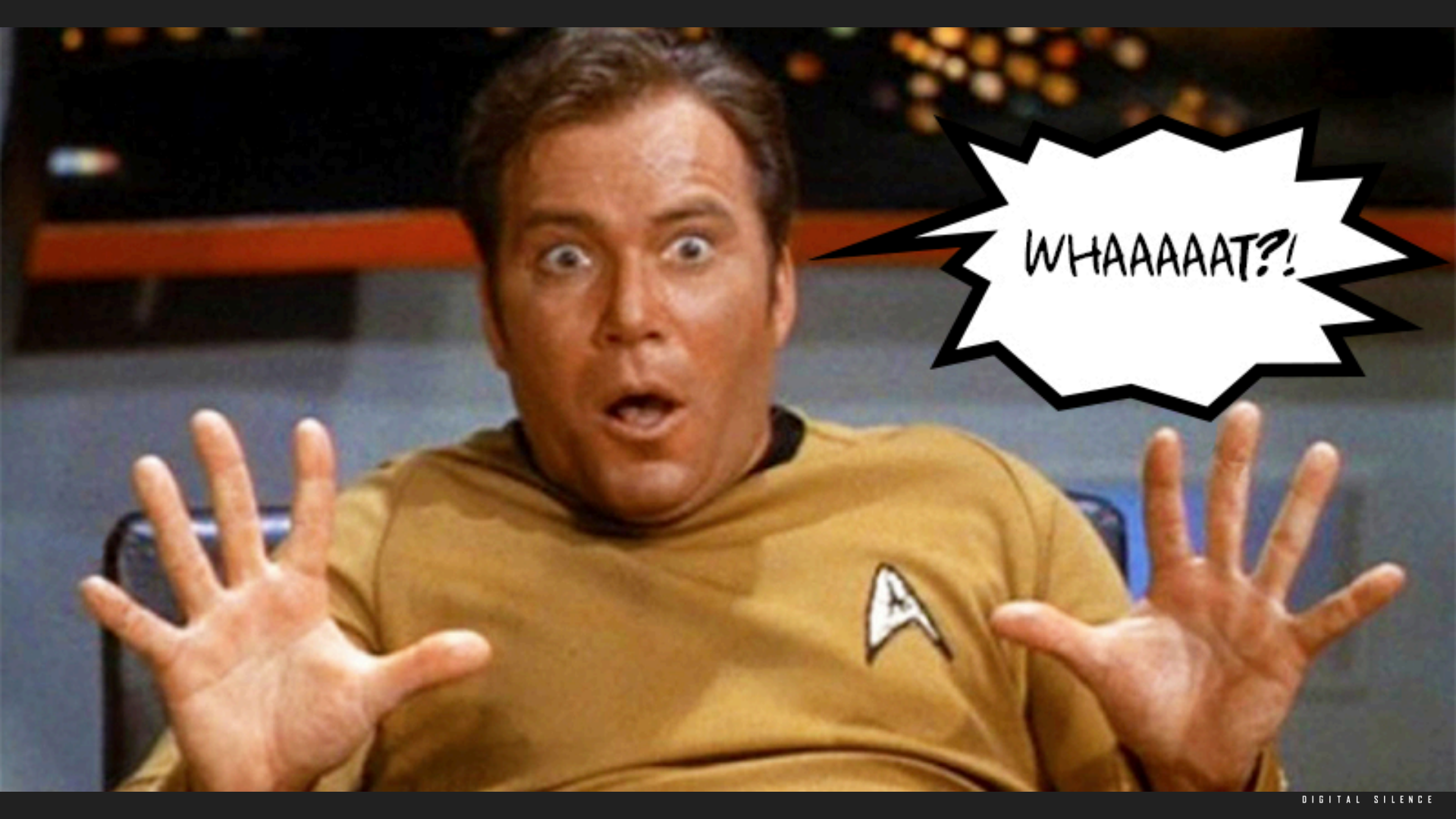
Quick Detour: MAC Filtering and MAC Authentication Bypass (MAB)





Fun fact: not all devices support 802.1x....





WHAAAAAT?!

Not all devices support 802.1x:

- Enterprise organizations with 802.1x protected networks need to deploy them anyways
- Solution: disable 802.1x on the port used by the device – this is known as a **port security exception**
- 802.1x usually replaced with MAC filtering or some other weak form of access control

Port security exceptions:

- Historically, very prevalent due to widespread lack of 802.1x support by peripheral devices (printers, IP cameras, etc)
- Low hanging fruit for attackers – much easier than trying to actually bypass 802.1x using a bridge or hub

Port security exceptions are slowly
dying....



Support for 802.1x by peripheral device manufacturers has increased dramatically:

- Legacy hardware phased out, replaced with 802.1x capable models

Port security exceptions:

- Have become less prevalent
- Are not quite the low-hanging fruit that they used to be

Improved adoption of 802.1x **does not** imply strong port security for peripheral devices:

- 802.1x-2010 support not a reality yet for peripheral devices
- 802.1x-2004 can be bypassed using bridges, injections, etc
- Adoption for secure EAP methods can be expected to be lower than domain joined devices



What about attacking EAP?



Makes sense as an alternative to relying on port security exceptions:

- Adoption of secure EAP methods already low across all device types
- Adoption of secure EAP methods can be expected to be lower for peripheral devices



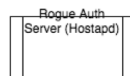
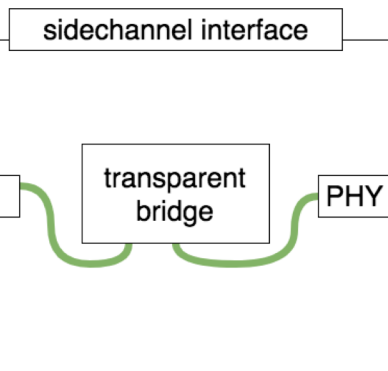
Rogue Gateway Attack Against 802.1x- 2004



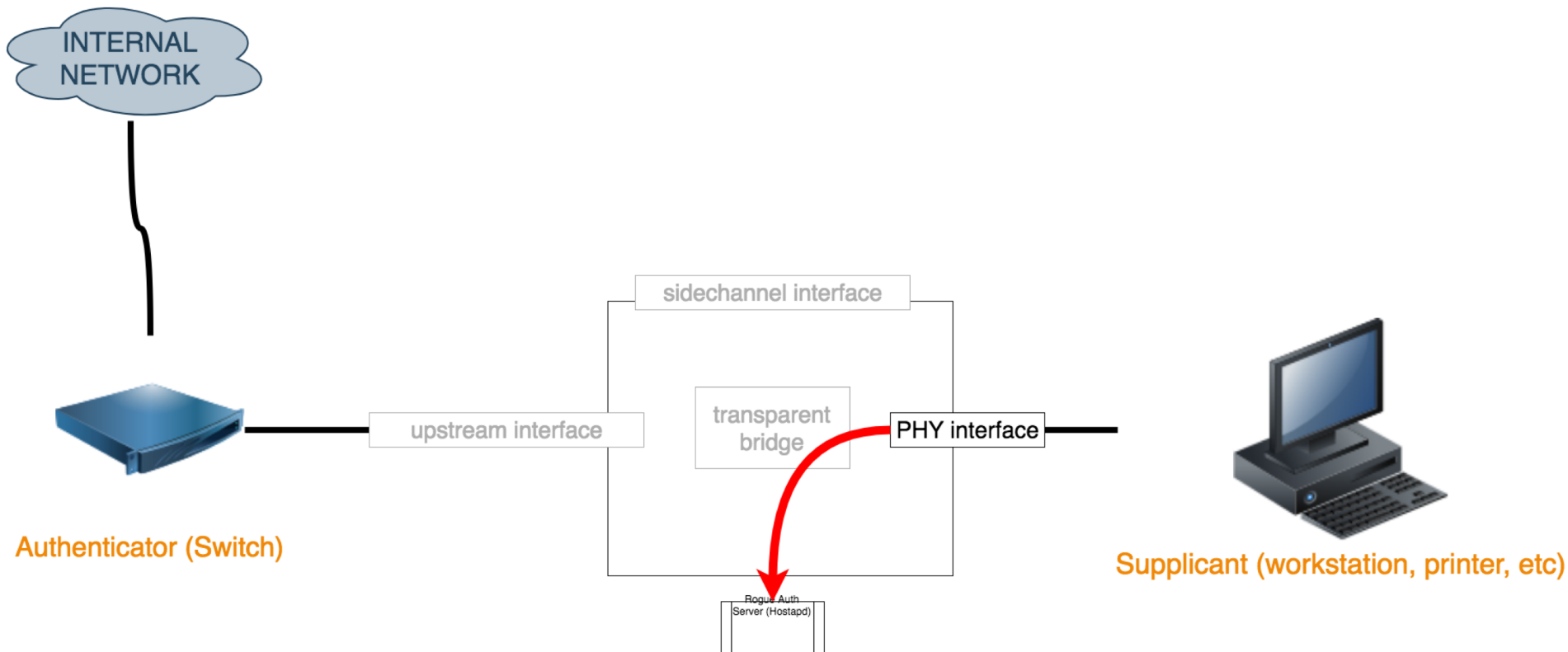


Authenticator (Switch)

upstream interface



Supplicant (workstation, printer, etc)



All interfaces down except sidechannel and PHY.
Rogue auth server (hostapd) listening on PHY.

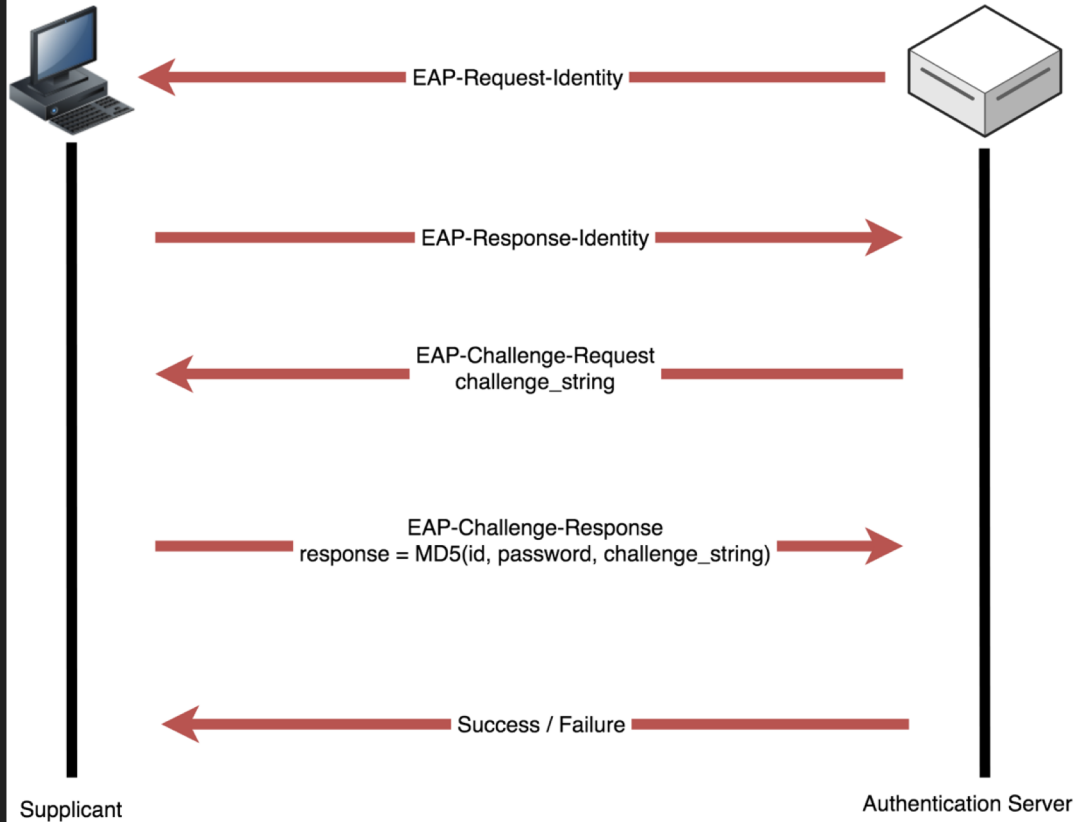
EAP-MD5



EAP-MD5 is widely used to protect peripheral devices such as printers:

- Easy to setup and configure
- Still better than MAC filtering

EAP-MD5 (Forwarded Between Authentication Server and Supplicant by Authenticator)



Entire process occurs over plaintext (bad bad bad bad bad)

Brad Antoniewicz and Josh Wright in 2008: ^[13]

- attacker can capture MD5-Challenge-Request and MD5-Challenge-Response by passively sniffing traffic ^[13]
- Dictionary attack can be used to obtain a password using captured data ^[13]

Fanbao Liu and Tao Xie in 2012: ^[19]

- EAP-MD5 credentials can be recovered even more efficiently using length-recovery attack ^[19]



Leveraging what we know about how to attack EAP-MD5 and 802.1x-2004:

1. Use bridge-based approach to place rogue device between supplicant and authenticator
2. Wait for the supplicant to authenticate, and sniff the EAP-MD5-Challenge and EAP-MD5-Response when it does ^[13]
3. Crack credentials, connect to network using Bait n' Switch



One major drawback to this approach:

- We must wait for the supplicant to reauthenticate with the switch



Realistically, this will not happen unless supplicant is unplugged

- disabling a virtual network interface is not enough
- Using mechanical splitters is an option, but the less overhead the better



EAP-MD5 Forced Reauthentication Attack Against 802.1x-2004



First two steps of the EAP authentication process: [1][2][9]

1. (optional) supplicant sends the authenticator an EAPOL-Start frame
2. The authenticator sends the supplicant an EAP-Request-Identity frame

Problem: supplicant has no way of verifying if incoming EAP-Request-Identity frame has been sent in response to an EAPOL-Start.



What this means: we can force reauthentication by sending an EAPOL-Start frame to the authenticator as if it came from the supplicant (MAC spoofing):

- Result: authenticator will send EAP-Request-Identity frame to the actual supplicant, kickstarting the reauthentication process
- Both the authenticator and supplicant believe that the other party has initiated the reauthentication attempt

Demo: Forced Reauthentication




```
root@solstice: ~ (ssh) #1
root@food: freeradius -X (ssh)
Waking up in 1.3 seconds.
Cleaning up request 384 ID 88 with timestamp +6161
Cleaning up request 385 ID 89 with timestamp +6161
Cleaning up request 386 ID 90 with timestamp +6161
Cleaning up request 387 ID 91 with timestamp +6161
Cleaning up request 388 ID 92 with timestamp +6161
Cleaning up request 389 ID 93 with timestamp +6161
Cleaning up request 390 ID 94 with timestamp +6162
Cleaning up request 391 ID 95 with timestamp +6162
Cleaning up request 392 ID 96 with timestamp +6162
Cleaning up request 393 ID 97 with timestamp +6162
Waking up in 2.2 seconds.
Cleaning up request 394 ID 98 with timestamp +6164
Cleaning up request 395 ID 99 with timestamp +6164
Cleaning up request 396 ID 100 with timestamp +6164
Cleaning up request 397 ID 101 with timestamp +6164
Cleaning up request 398 ID 102 with timestamp +6164
Cleaning up request 399 ID 103 with timestamp +6164
Cleaning up request 400 ID 104 with timestamp +6164
Cleaning up request 401 ID 105 with timestamp +6164
Cleaning up request 402 ID 106 with timestamp +6164
Cleaning up request 403 ID 107 with timestamp +6164
Ready to process requests.
[]

root@solstice: ~
root@solstice:~# scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python Crypto lib. Won't be able to decrypt WEP.
INFO: Can't import python Crypto lib. Disabled certificate manipulation tools
INFO: Can't import python ecdsa lib. Disabled certificate manipulation tools
Welcome to Scapy (2.3.3)
>>> sendp(Ether(src='a8:60:b6:18:82:38', dst='01:80:c2:00:00:03')/EAPOL(type=1, iface="br0"))
```

```
root@food: hostapd hostapd-wired-base.conf (ssh)
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: Sending EAP Packet (identifier 63)
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: received EAP packet (code=2 id=63 len=43) from STA: EAP Response-PEAP (25)
eth0: RADIUS Sending RADIUS message to authentication server
eth0: RADIUS Next RADIUS client retransmit in 3 seconds
eth0: RADIUS Received 133 bytes from RADIUS server
eth0: RADIUS Received RADIUS message
eth0: STA a8:60:b6:18:82:38 RADIUS: Received RADIUS packet matched with a pending request, round trip time 0.00 sec
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: decapsulated EAP packet (code=1 id=64 len=75) from RADIUS server: EAP-Request-PEAP (25)
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: Sending EAP Packet (identifier 64)
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: received EAP packet (code=2 id=64 len=107) from STA: EAP Response-PEAP (25)
eth0: RADIUS Sending RADIUS message to authentication server
eth0: RADIUS Next RADIUS client retransmit in 3 seconds
eth0: RADIUS Received 149 bytes from RADIUS server
eth0: RADIUS Received RADIUS message
eth0: STA a8:60:b6:18:82:38 RADIUS: Received RADIUS packet matched with a pending request, round trip time 0.00 sec
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: decapsulated EAP packet (code=1 id=65 len=91) from RADIUS server: EAP-Request-PEAP (25)
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: Sending EAP Packet (identifier 65)
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: received EAP packet (code=2 id=65 len=43) from STA: EAP Response-PEAP (25)
eth0: RADIUS Sending RADIUS message to authentication server
eth0: RADIUS Next RADIUS client retransmit in 3 seconds
eth0: RADIUS Received 101 bytes from RADIUS server
eth0: RADIUS Received RADIUS message
eth0: STA a8:60:b6:18:82:38 RADIUS: Received RADIUS packet matched with a pending request, round trip time 0.00 sec
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: decapsulated EAP packet (code=1 id=66 len=43) from RADIUS server: EAP-Request-PEAP (25)
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: Sending EAP Packet (identifier 66)
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: received EAP packet (code=2 id=66 len=43) from STA: EAP Response-PEAP (25)
eth0: RADIUS Sending RADIUS message to authentication server
eth0: RADIUS Next RADIUS client retransmit in 3 seconds
eth0: RADIUS Received 169 bytes from RADIUS server
eth0: RADIUS Received RADIUS message
eth0: STA a8:60:b6:18:82:38 RADIUS: Received RADIUS packet matched with a pending request, round trip time 0.00 sec
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: old identity 'testing' updated with User-Name from Access-Accept 'testing'
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: decapsulated EAP packet (code=3 id=66 len=4) from RADIUS server: EAP Success
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: Sending EAP Packet (identifier 66)
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: authorizing port
eth0: STA a8:60:b6:18:82:38 IEEE 802.1X: authenticated - EAP type: 25 (PEAP)
[]
```



EAP-MD5 Forced Reauthentication Attack:

1. Introduce rogue device into the network between authenticator and supplicant
2. Start transparent bridge and passively sniff traffic
3. Force reauthentication by sending spoofed EAPOL-Start frame to the authenticator
4. Captured and crack EAP-MD5-Challenge and EAP-MD5-Response



Demo: EAP-MD5 Forced Reauthentication Attack



```
2. root@localhost: /home/solstice/silentbridge (ssh)
X ..silentbridge (ssh) %1 X ~ (zsh) %2
[ root@localhost ] [ /dev/pts/7 ] [ master ⚡ ]
[ /home/solstice/silentbridge ] > |
```



DIGITAL SILENCE

Proposed Mitigation – safety-bit in the EAP-Request-Identity frame:

- set to 1 when the frame was sent in response to an EAPOL-Start frame
- Checked when supplicant receives an EAP-Request-Identity frame
- Authentication process aborted if safety bit set to 1 and supplicant did not recently issue EAPOL-Start frame



Closing Thoughts



Closing Thoughts

Our contributions:

- Rogue Gateway and Bait n Switch – Bypass 802.1x-2010 by attacking its authentication mechanism
- Updated & improved existing 802.1x-2004 bypass techniques
- EAP-MD5 Forced Reauthentication attack – improved attack against EAP-MD5 on wired networks

Closing Thoughts

Key takeaways (1 of 2):

- Port security is still a positive thing (keep using it!)
- Port security is *not* a substitute for a layered approach to network security (i.e. deploying 802.1x does not absolve you from patch management responsibilities)



Closing Thoughts

Key takeaways (2 of 2):

- Benefits provided by 802.1x can be undermined due to continued use of EAP as authentication mechanism
- Improved 802.1x support by peripheral device manufacturers largely undermined by lack of support for 802.1x-2010 and low adoptions / support rates for strong EAP methods

Blog post & whitepaper:
<https://www.digitalsilence.com/blog/>

Tool:
github.com/s0lst1c3/silentbridge



References:

[1] <http://www.ieee802.org/1/pages/802.1x-2001.html>

[2] <http://www.ieee802.org/1/pages/802.1x-2004.html>

[3] <https://blogs.technet.microsoft.com/steriley/2005/08/11/august-article-802-1x-on-wired-networks-considered-harmful/>

[4] <https://www.defcon.org/images/defcon-19/dc-19-presentations/Duckwall/DEFCON-19-Duckwall-Bridge-Too-Far.pdf>

[5] <https://www.gremwell.com/marvin-mitm-tapping-dot1x-links>



References:

[6] https://hackinparis.com/data/slides/2017/2017_Legrand_Valerian_802.1x_Network_Access_Control_and_Bypass_Techniques.pdf

[7] https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/identity-based-networking-services/deploy_guide_c17-663760.html

[8] <https://1.ieee802.org/security/802-1ae/>

[9] <https://standards.ieee.org/findstds/standard/802.1X-2010.html>

[10] <http://www.ieee802.org/1/files/public/docs2013/ae-seaman-macsec-hops-0213-v02.pdf>



References:

[11] https://www.gremwell.com/linux_kernel_can_forward_802_1x

[12] <https://www.intel.com/content/www/us/en/support/articles/000006999/network-and-i-o/wireless-networking.html>

[13] http://www.willhackforsushi.com/presentations/PEAP_Shmocon2008_Wright_Antoniewicz.pdf

[14] https://link.springer.com/content/pdf/10.1007%2F978-3-642-30955-7_6.pdf

[15] <https://support.microsoft.com/en-us/help/922574/the-microsoft-extensible-authentication-protocol-message-digest-5-eap>



References:

[16] <https://tools.ietf.org/html/rfc3748>

[17] <https://code.google.com/archive/p/8021xbridge/source/default/commits>

[18] <https://github.com/mubix/8021xbridge>

[19] <https://hal.inria.fr/hal-01534313/document>

[20] <https://sensepost.com/blog/2015/improvements-in-rogue-ap-attacks-mana-1%2F2/>



References:

[21] <https://tools.ietf.org/html/rfc4017>

[22] <http://web.archive.org/web/20160203043946/https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/>

[23] <https://crack.sh/>

[24] <https://tools.ietf.org/html/rfc5216>

[25] https://4310b1a9-a-93739578-s-sites.googlegroups.com/a/riosec.com/home/articles/Open-Secure-Wireless/Open-Secure-Wireless.pdf?attachauth=ANoY7cqwbbsU93t3gE88UC_qqtG7cVvms7FRutz0KwK1oiBcEJMIQuUmpGSMMD7oZGyGmt4M2HaBhHFb07j8Gvmb_HWIE8rSfLKDvB0AI80u0cYwSNi5ugTP1JtFXsy1yZn8-85icVc32PpzxLJwRinf2UGzNbEdO97Wsc9xcjnc8A8MaFkPbUV5kwsMYHaxMiWwTcE-A8Dp49vv-tmk86pNMaeUeumBw_5vCZ6C3Pvc07hVbyTOsjqo6C6WpfVhd_M0BNW0RQtl&attredirects=0



References:

[26] <https://txlab.wordpress.com/2012/01/25/call-home-ssh-scripts/>

[27] <https://txlab.wordpress.com/2012/03/14/improved-call-home-ssh-scripts/>

[28] https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3560/software/release/12-2_37_se/command/reference/cr1/cli3.html#wp1948361

[29] https://www.juniper.net/documentation/en_US/junos/topics/concept/port-security-persistent-mac-learning.html

[30] <https://tools.ietf.org/html/rfc3579>



References:

[31] <https://tools.ietf.org/html/rfc5281>

