Owning the LAN in 2018

Defeating MACsec and 802.1x-2010

DEF CON 26

Gabriel “solstice” Ryan

Disclaimers & Updates

These slides are an early version designed for pre-release prior to DEF CON 26. All content will be updated by the time of the presentation at DEF CON 26 in August. Final versions of all content will be available at:

- https://www.digitalsilence.com/blog/

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_

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

Co-Founder / Senior Security Assessment Manager @ Digital Silence

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

Twitter: @s0lst1c3

LinkedIn: ms08067

Email: gabriel@digitalsilence.com

Introduction to 802.1x

The 802.1x authentication protocol:

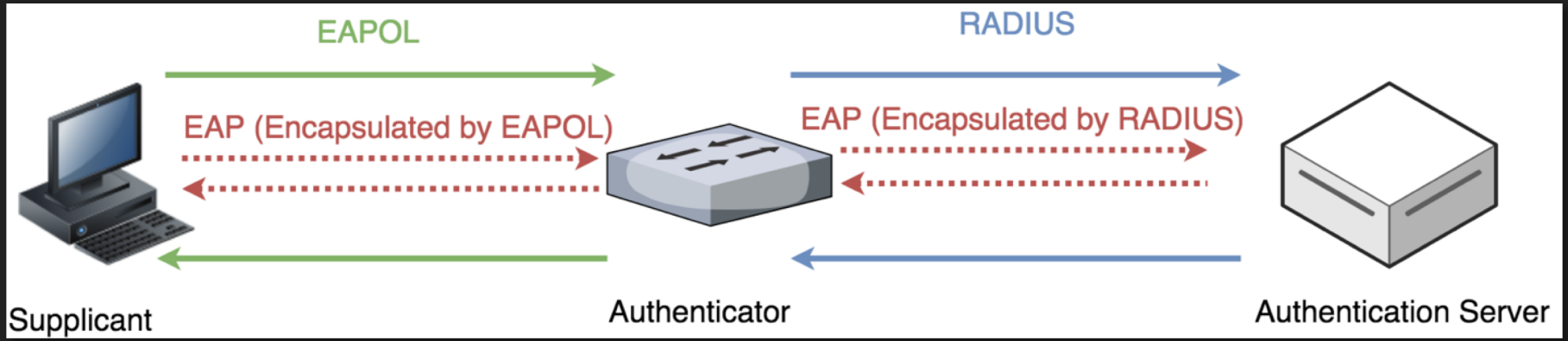
- Authentication protocol
- Used to protect a local area network (LAN) or wireless local area network (WLAN) with rudimentary authentication

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]

DIGITAL SILENCE

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]

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?

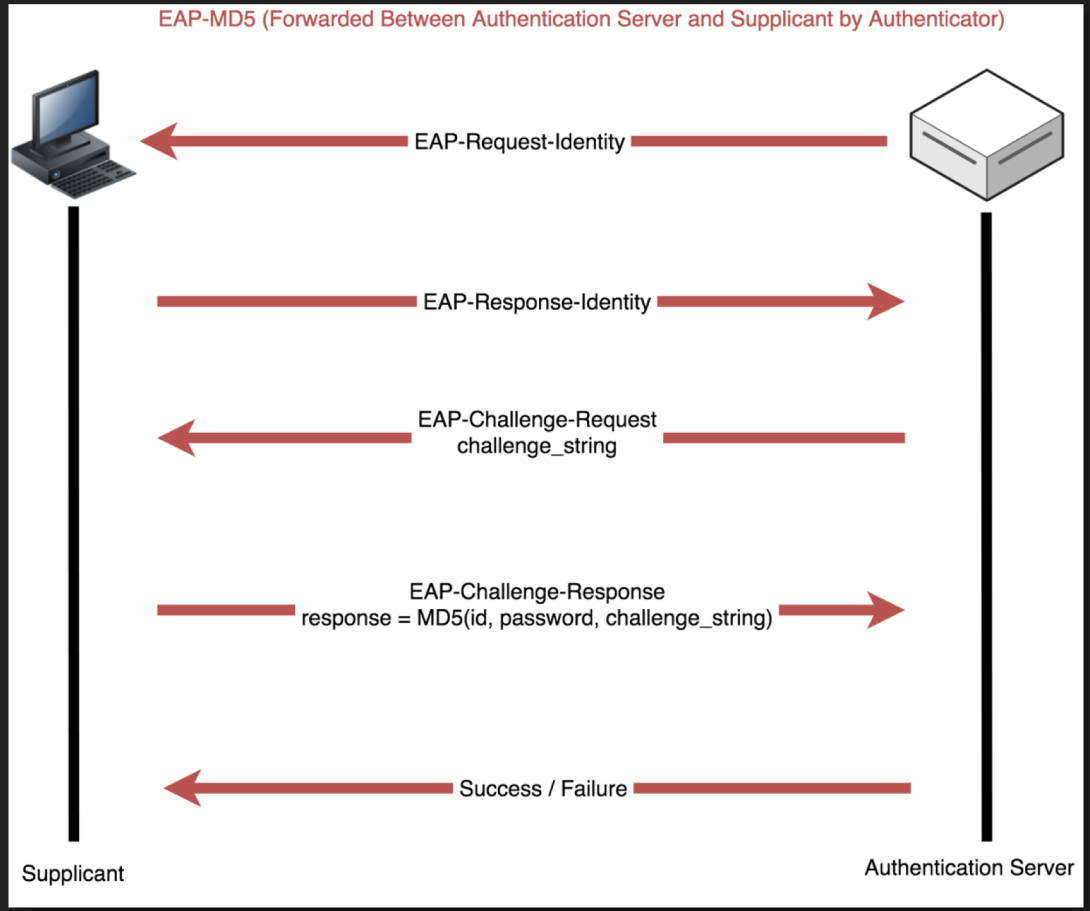
Extensible Authentication Protocol (EAP):

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-MD5: Security Issues

Entire process occurs over plaintext (bad bad bad bad bad)
Brad Antoniewicz and Josh Wright in 2008: [13]

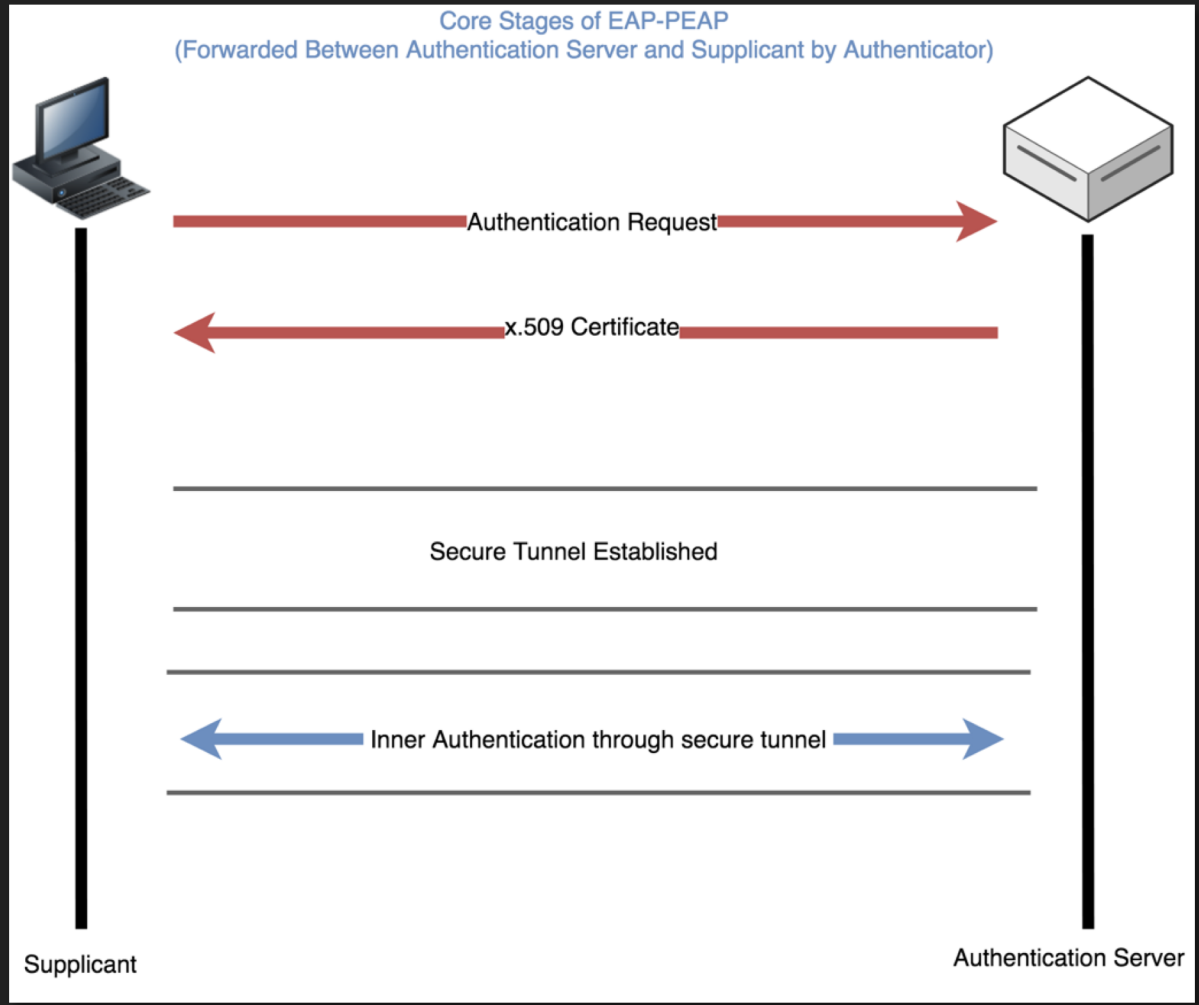
- attacker can capture MD5-Challenge-Request and MD5-Challenge-Response by passively sniffing traffic [13]
- Dictionary attack can be used to obtain a password using captured data [13]

EAP-MD5: Security Issues

Fanbao Liu and Tao Xie in 2012: [19]

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

EAP-PEAP



EAP-PEAP: Security Issues

Remember – EAP also used for wireless authentication. 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]
- 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]

DIGITAL SILENCE

EAP-TLS

EAP-TLS

EAP-TLS was introduced by RFC 5216 in response to weaknesses in EAP methods such as EAP-PEAP and EAP-TTLS:

- strength lies in use of mutual certificate-based authentication during the outer authentication process [24]
- prevents the kinds of MITM attacks that affect weaker EAP methods [24]
- Poor adoption rate [25]

Brief History of Wired Port Security

Brief History of Wired Port Security

2001 – the 802.1x-2001 standard is created to provided 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 – Valérian 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…

MAC Filtering and MAC Authentication Bypass (MAB)

DIGITAL SILENCE

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

Demo: MAB

Current State of Wired Port Security

Relatively new technology: 802.1x-2010

Uses MACsec to provide:

- hop-by-hop Layer 2 encryption [3][4][6][7]
- Packet-by-packet integrity check [3][4][6][7]

Mitigates bridge-based attacks that affect 802.1x-2004 [7]

802.1x-2010: Vendor Support

Largest manufacturers of enterprise networking hardware now support 802.1x-2010 and MACsec:

- Limited to high end equipment
- Full support for all 802.1x-2010 features varies by make and model

802.1x-2010: Adoption Rates

Loaded question, since adoption rates for 802.1x itself remain low. With that said:

- MACsec and 802.1x-2010 are just beginning to take off
- adoption rate steadily increasing

Improvements in Peripheral Device Security

Most printer manufacturers offer at least one affordable model that supports 802.1x:

- Legacy hardware phased out, replaced with 802.1x capable models
- Port security exceptions becoming less prevalent (many sad red teamers)

Improvements in Peripheral Device Security

Goal of this project: tip the scales back in favor of attackers

- explore ways in which 802.1x-2010 and MACsec can be bypassed
- Address the reduced prevalence of port security exceptions by identifying alternative methods for attacking peripheral devices

Improvements to Bridge-Based Bypass Techniques

Classical Bridge-based 802.1x Bypass

- Developed by Alva Duckwall in 2014 [4]
- 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]

Improvement: Leveraging Native EAPOL Forwarding

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

Improvement: Leveraging Native EAPOL Forwarding

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]

Improvement: Leveraging Native EAPOL Forwarding

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]

DIGITAL SILENCE

Improvement: Bypassing Sticky MAC

Most modern authenticators use some form of Sticky MAC:

- dynamically associates the MAC address of the supplicant to the switch port once the supplicant has successfully authenticated [28][29]
- if another MAC address is detected on the switch port, a port security violation occurs and the port is blocked [28][29]

Improvement: Bypassing Sticky MAC

Our updated implementation:

- sets the bridge and PHY interfaces to the MAC address of the authenticator
- sets the upstream interface to the MAC address of the supplicant

```
82    core.firewalls.ebtables.flush()
83    core.firewalls.arptables.flush()
84
85    print '[*] Creating the bridge...'
86
87    # create the bridge
88    bridge = core.utils.bridge.Bridge(bridge_iface, switch_mac)
89    bridge.create()
90    bridge.enable_8021x_forwarding()
91    bridge.enable_ip_forwarding()
92    bridge.add_iface(phy, mac=switch_mac)
93    bridge.add_iface(upstream, mac=client_mac)
94
95    print '[*] bringing both sides of the bridge up...'
96
97    # bring both sides of bridge up
98    bridge.all_ifaces_up()
```

Improvement: Support for Side Channel Interaction

In Duckwall's original bypass, outbound ARP and IP traffic is initially blocked while the transparent bridge is initialized [4]:

- Prevents us from using a side channel device, such as an LTE modem

```
49  #start dark
50  arptables -A OUTPUT -j DROP
51  iptables -A OUTPUT -j DROP
52
53
54  # bring up the bridge with our bridge IP
55  ifconfig $BRINT $BRIP up promisc
56
57  # creat to source NAT the $COMPMAC
58  # for traffic leaving the device
59  # from the bridge mac address
60  ebtables -t nat -A POSTROUTING -s $SWMAC -o $SWINT -j snat --to-src $COMPMAC
61  ebtables -t nat -A POSTROUTING -s $SWMAC -o $BRINT -j snat --to-src $COMPMAC
62
```

Improvement: Bypassing Sticky MAC

Our updated implementation:

- added a firewall exception that allows outbound traffic from our side channel interface only
- allows the user to specify a desired egress destination port

```
[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
```

DIGITAL SILENCE

Demo: Improvements to Bridge-Based Bypass Techniques

Introduction to MACsec and 802.1x-2010

Introduction to 802.1x-2010 and MACsec

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 802.1x-2010 and MACsec

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

Introduction to 802.1x-2010 and MACsec

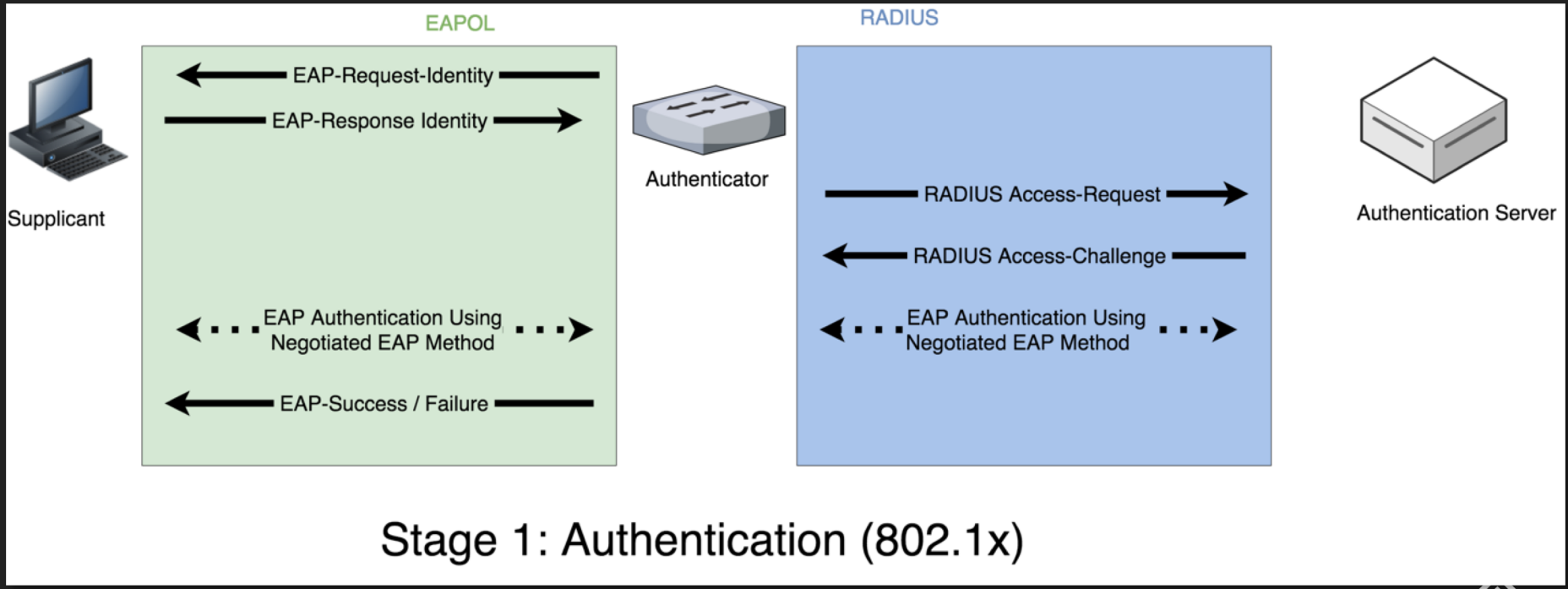
Support for hop-by-hop encryption particularly important: [7]

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

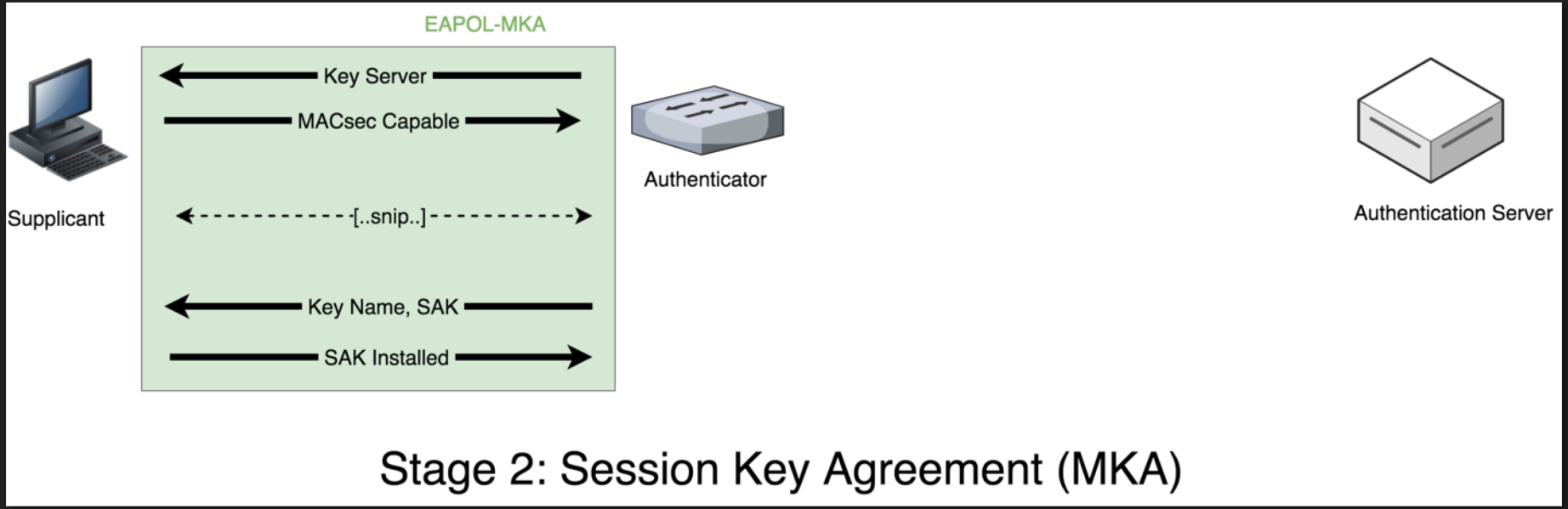
Introduction to 802.1x-2010 and MACsec

The 802.1x-2010 protocol works in three stages: [7][8][9]

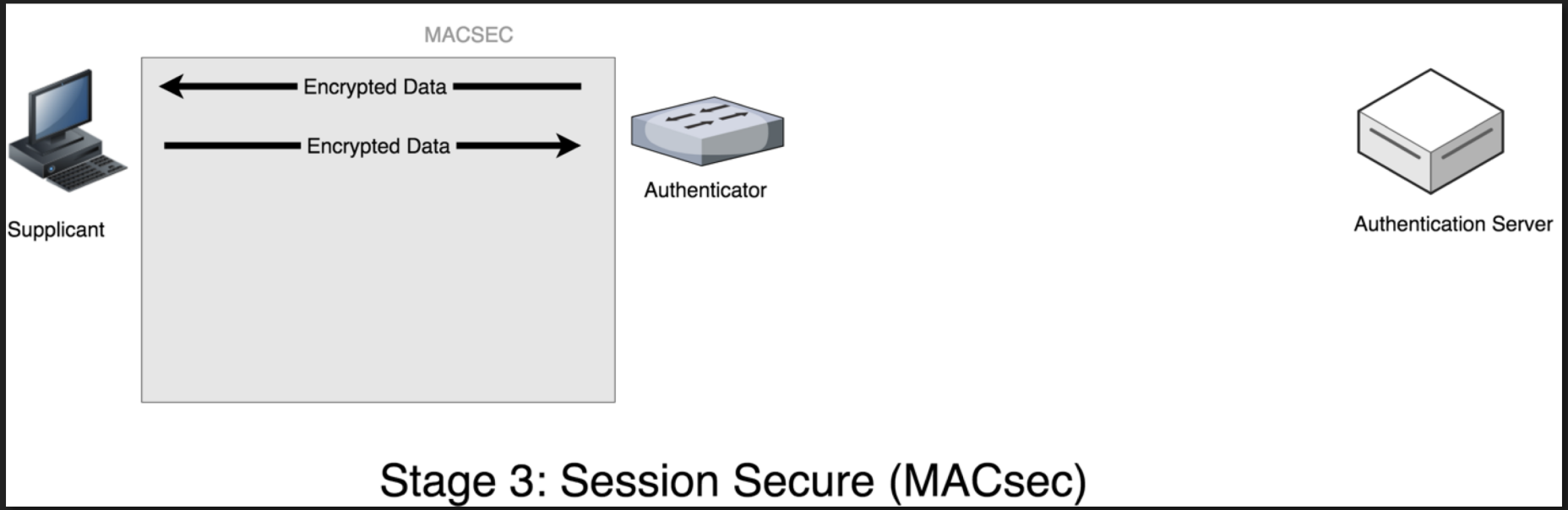
1. Authentication and Master Key Distribution
2. Session Key Agreement
3. Session Secure



Stage 1: Authentication (802.1x)



Stage 2: Session Key Agreement (MKA)



Stage 3: Session Secure (MACsec)

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

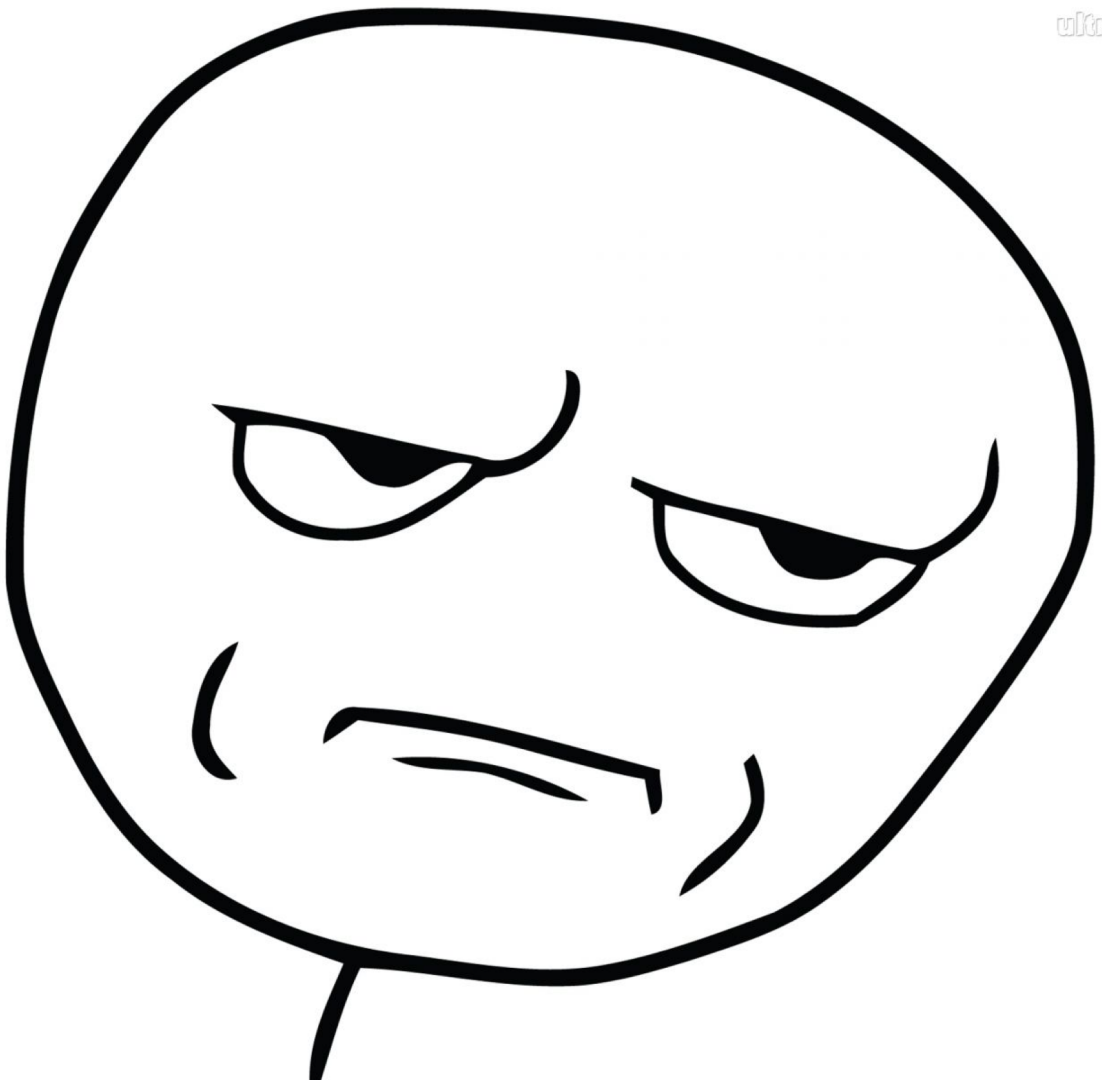
2010 – 802.1x-2010 is released

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

Focus shifts to attacking 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)

Defeating MACsec Using Rogue Gateway Attacks

Defeating MACsec Using Rogue Gateway Attacks

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:

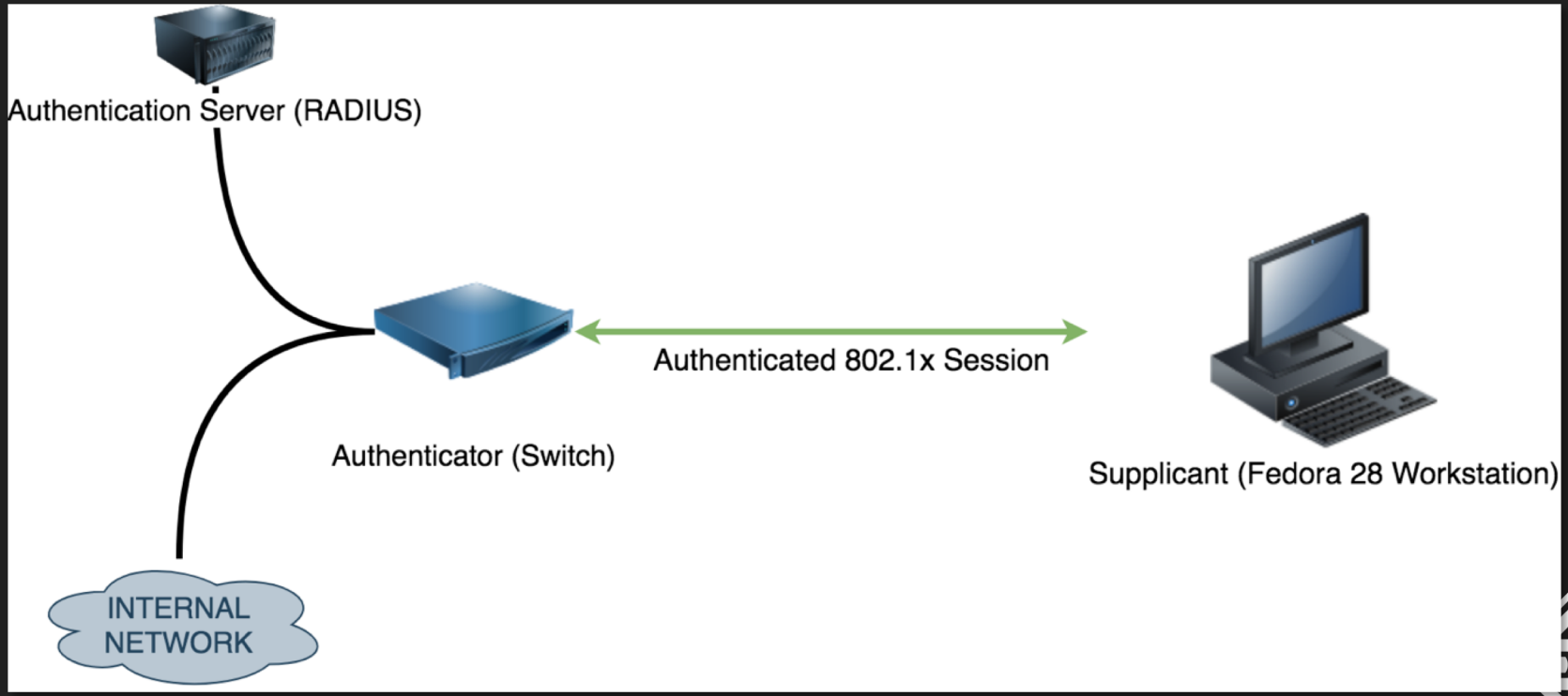
The 802.1x-2010 standard allows any EAP method so long as it: [7]

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

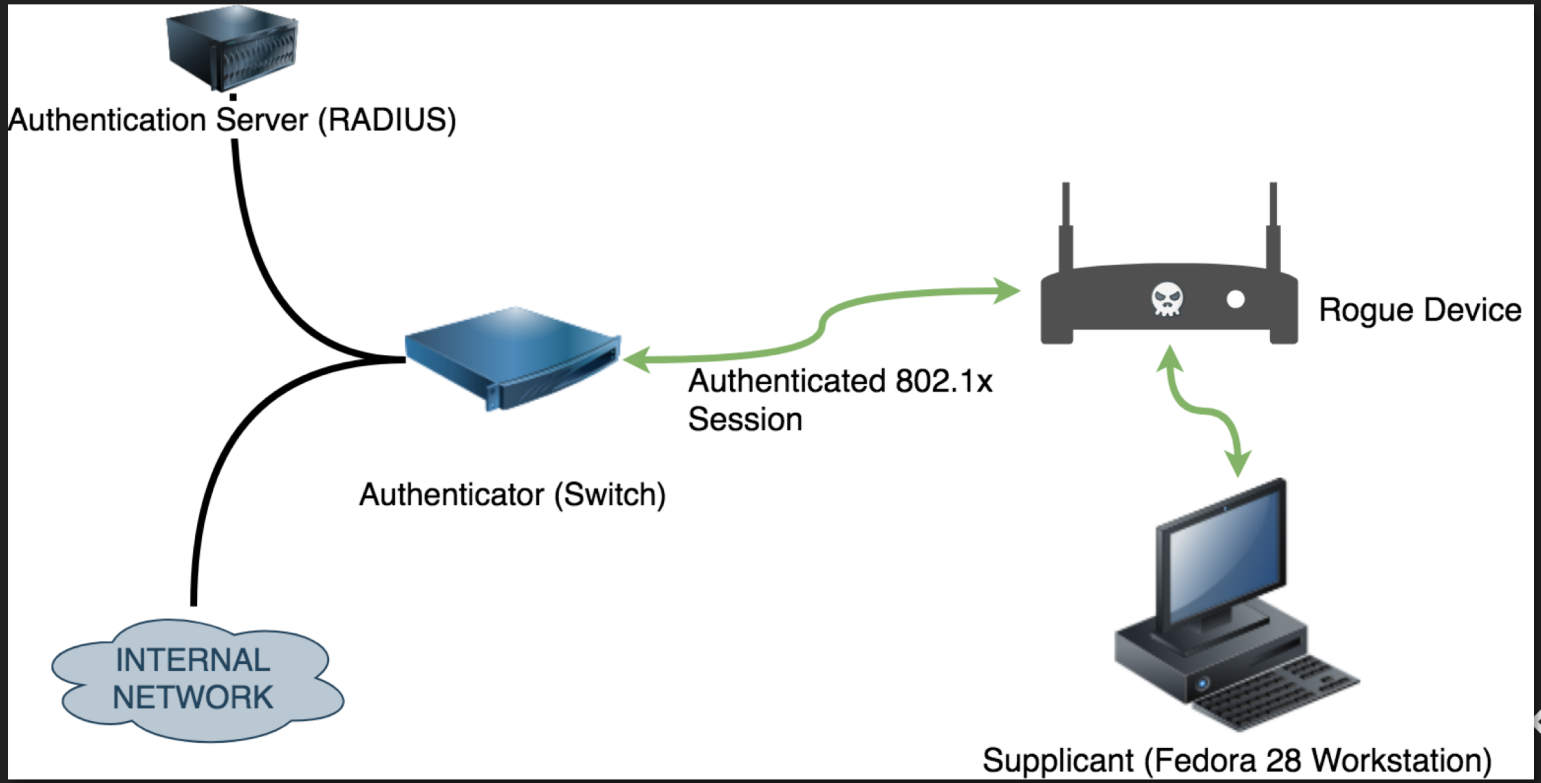
Plenty of commonly seen weak EAP methods that meet these requirements (EAP-PEAP, EAP-TTLS, etc).

DIGITAL SILENCE

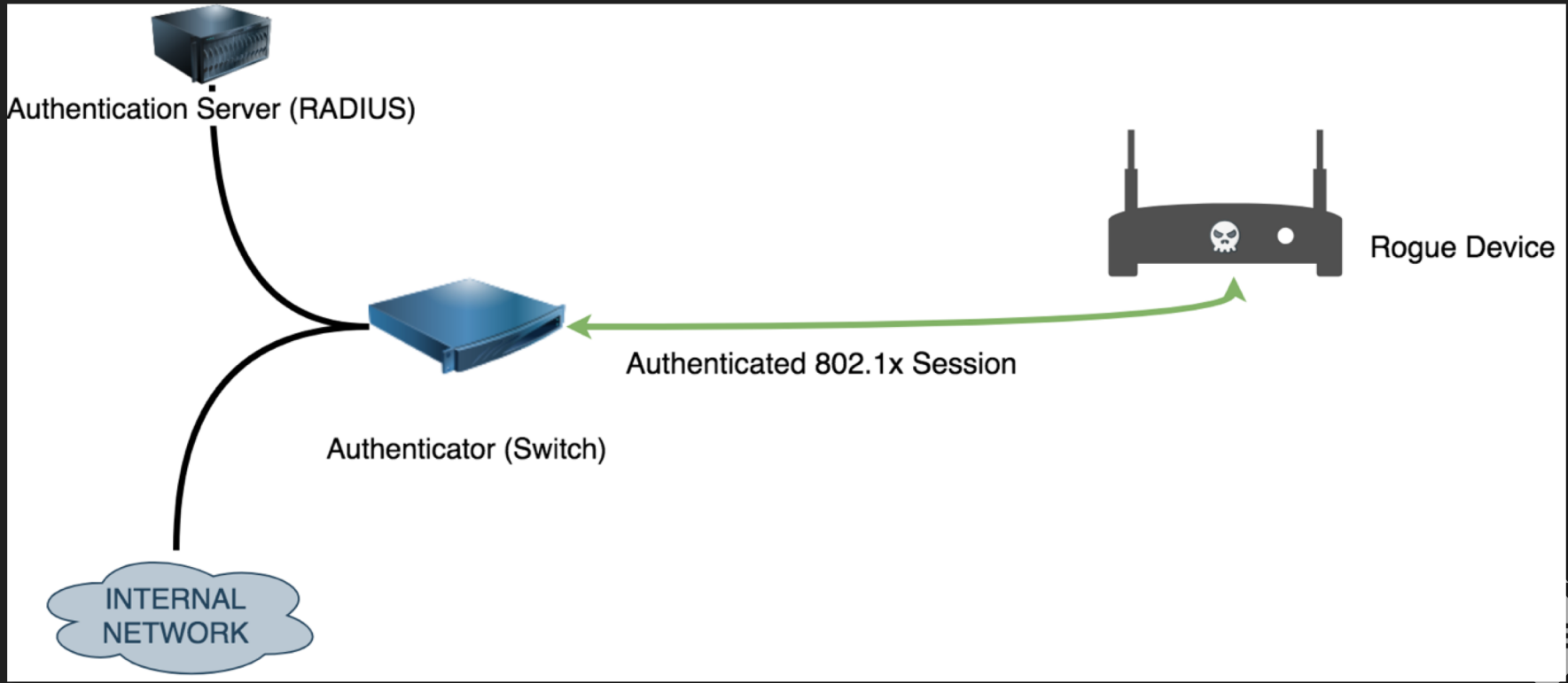
Our lab environment:



Option 1: MITM style bypass

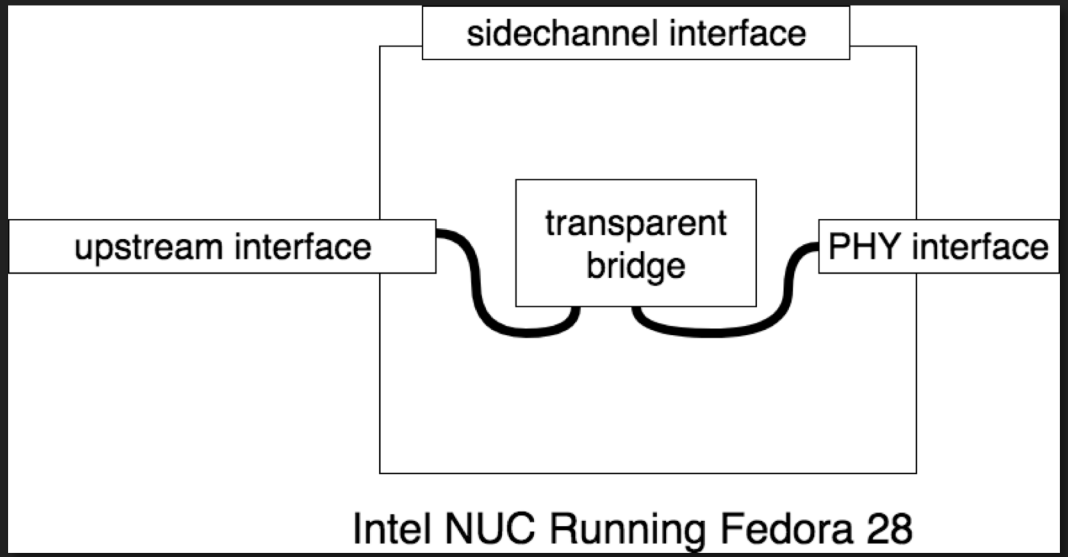


Option 2: Direct Access



Let’s build a rogue device…

Step 1: Device Core



Step 2: Mechanically Assisted Bypass

FRONT

BACK

DIGITAL SILENCE

Step 2: Mechanically Assisted Bypass

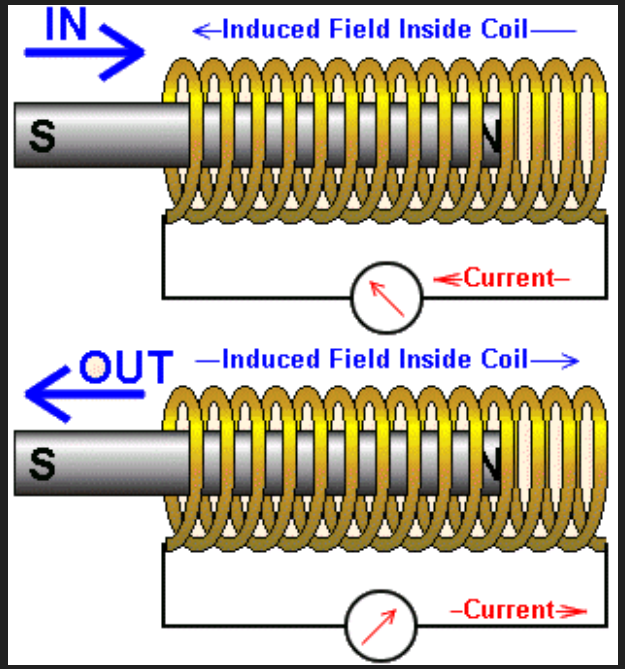
Need a way of manipulating the push switch:

- Using relays will lead to impedance issues unless you're an electrical engineer (which I am certainly not…)
- Option B: use solenoids

Step 2: Mechanically Assisted Bypass

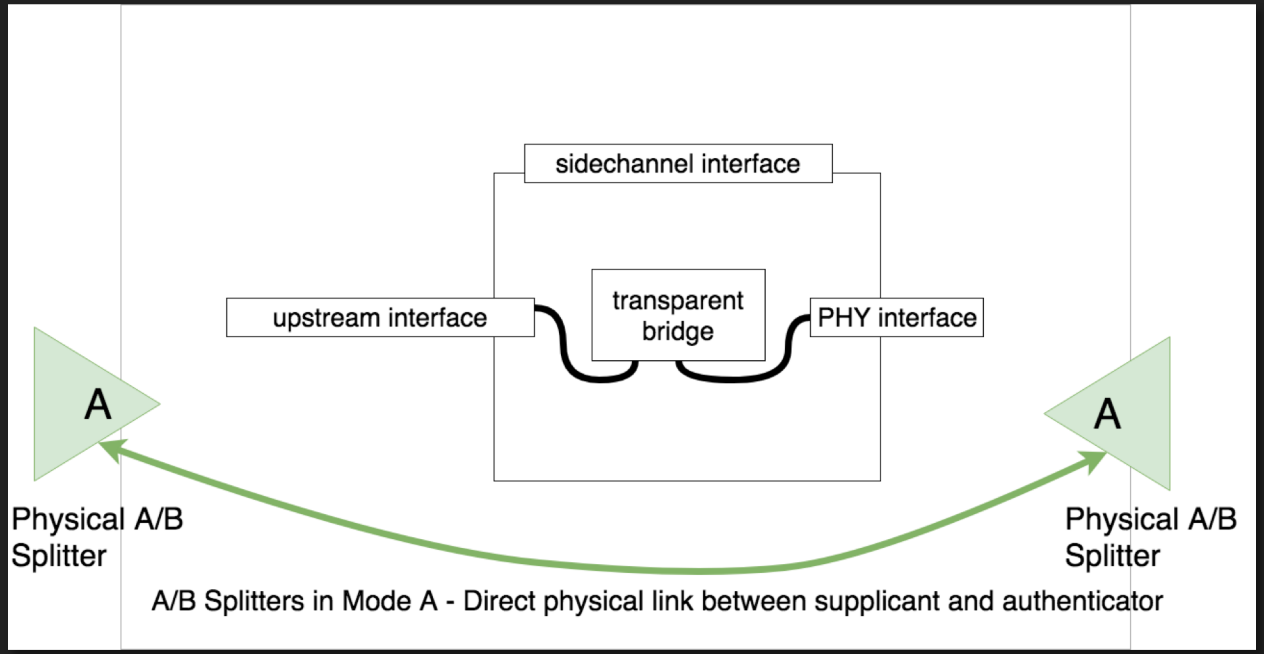
Pull Solenoid

Push Solenoid

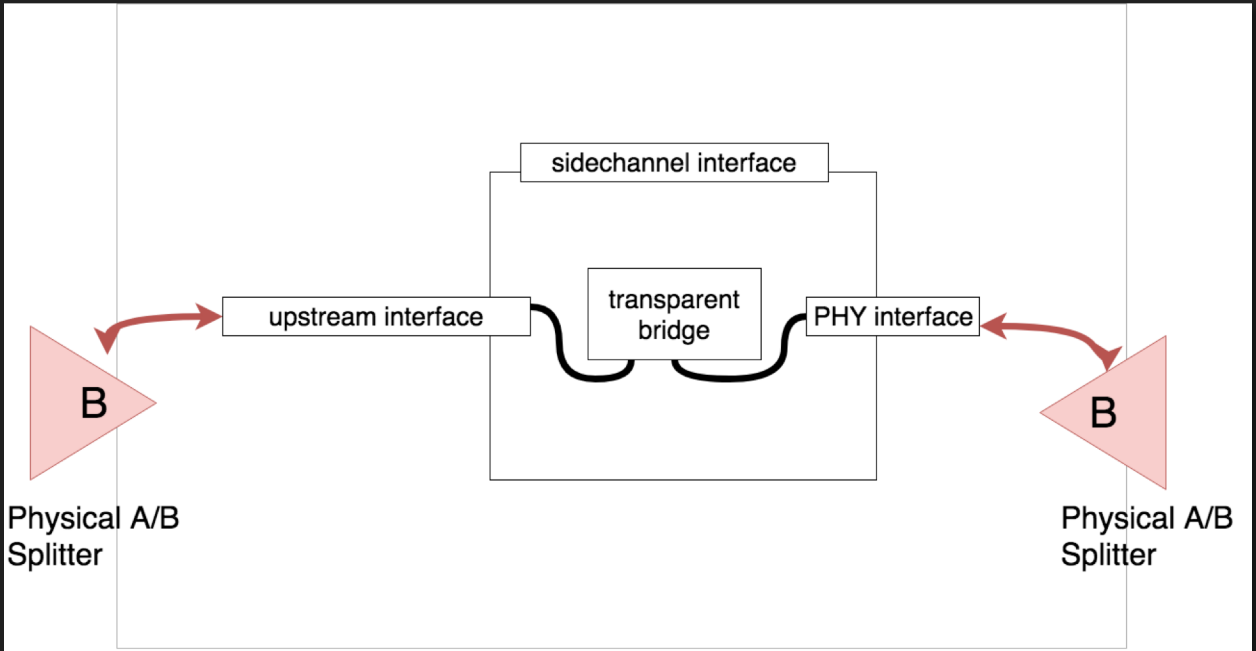


DIGITAL SILENCE

Step 2: Mechanically Assisted Bypass



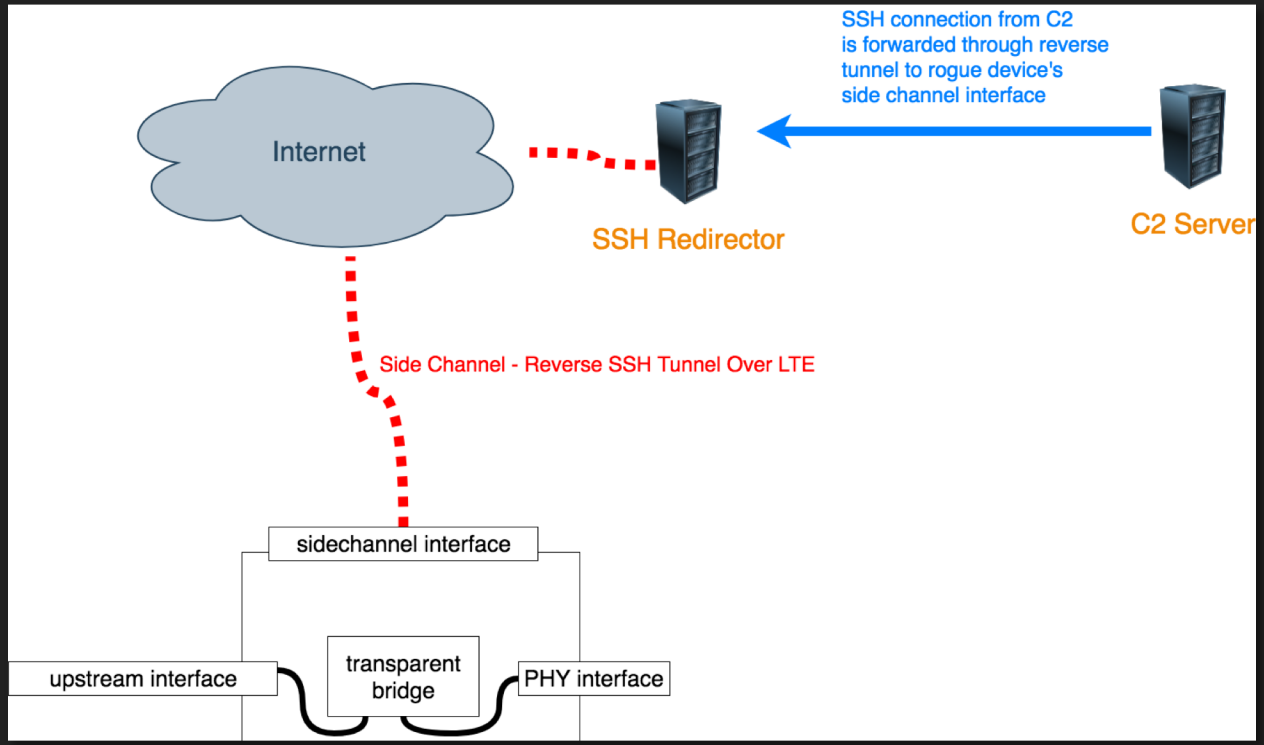
Step 2: Mechanically Assisted Bypass



A/B Splitters in Mode B: ethernet rerouted to rogue device

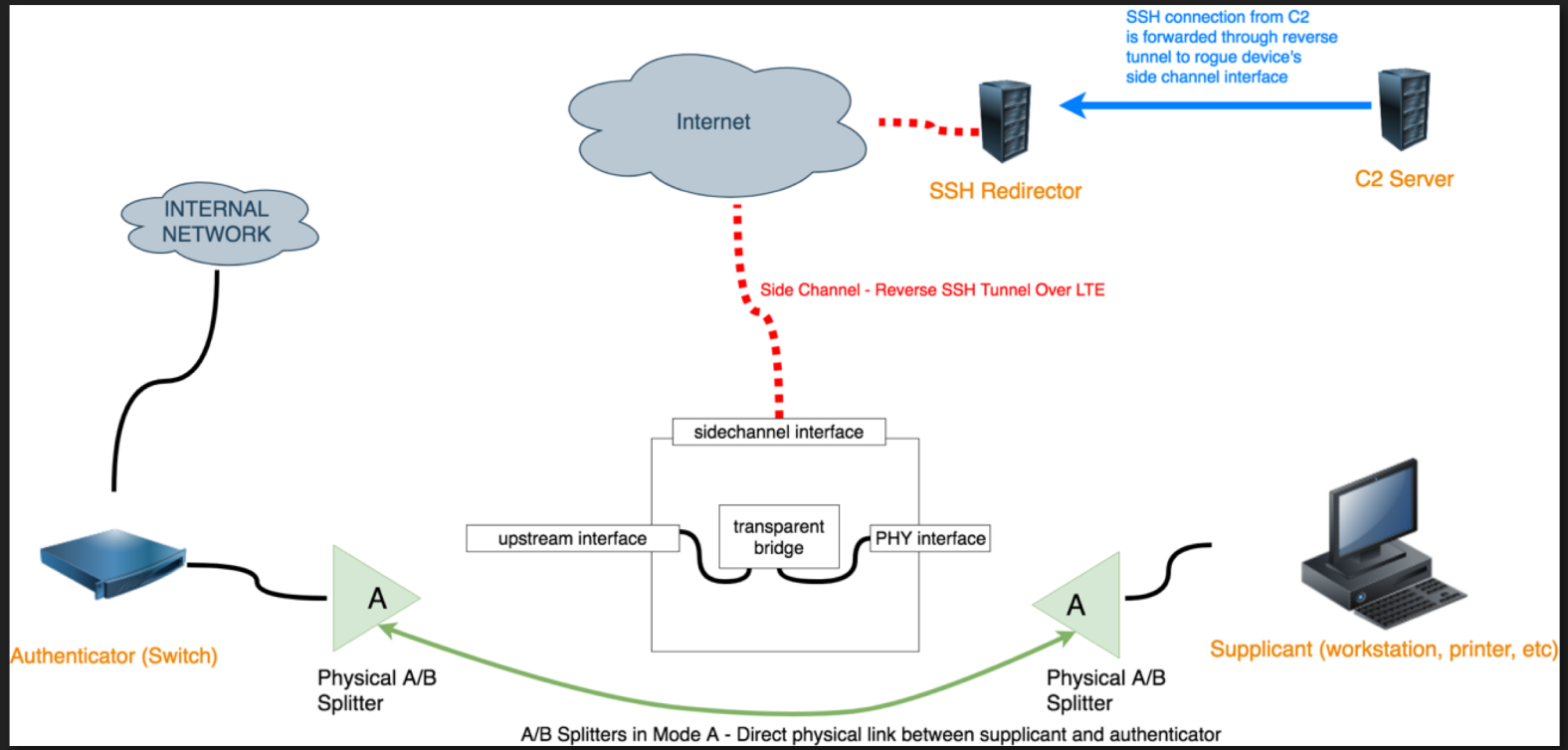
DIGITAL SILENCE

Step 3: Establish a Side Channel



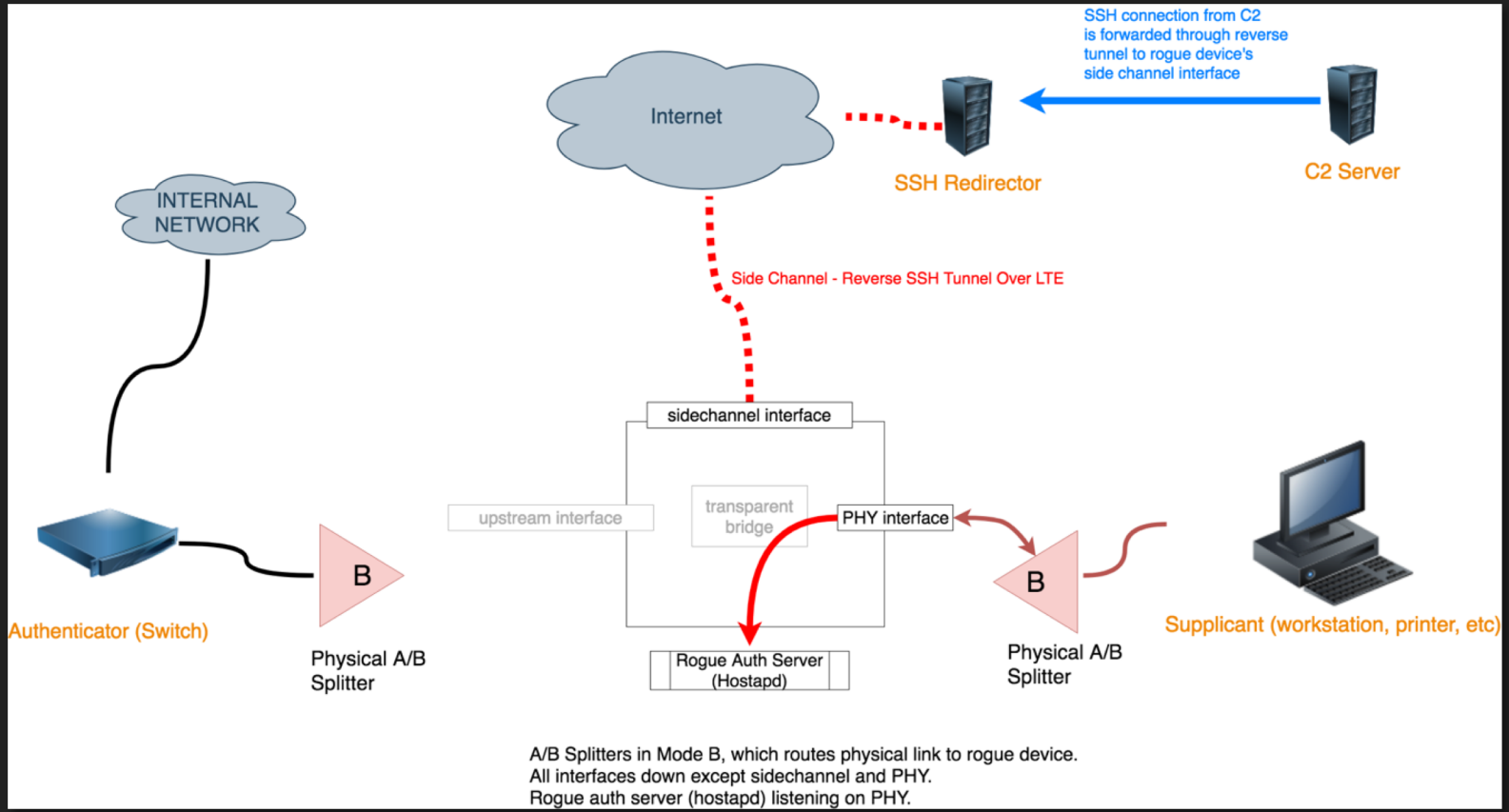
DIGITAL SILENCE

Step 4: Plant the Device



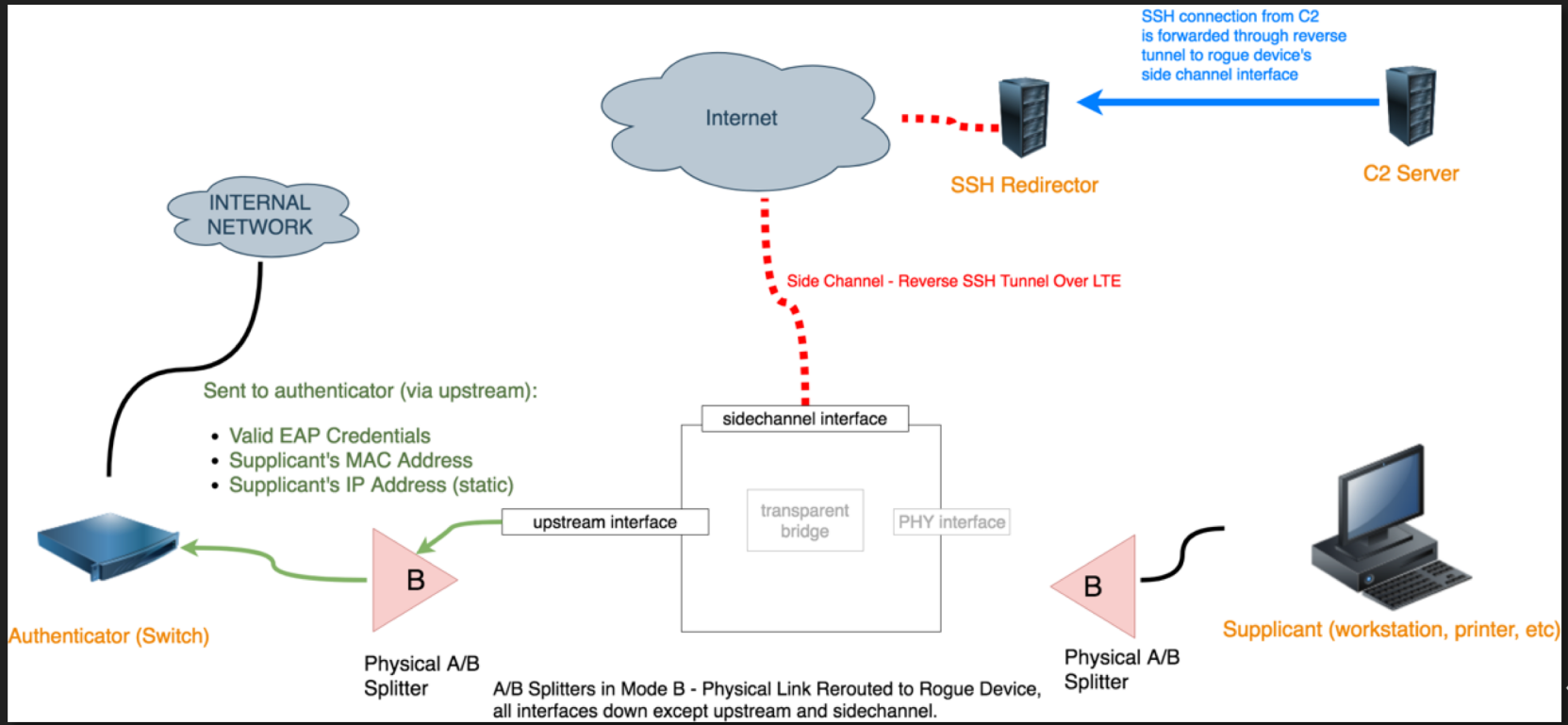
DIGITAL SILENCE

Step 5: Rogue Gateway Attack



DIGITAL SILENCE

Step 6: Bait n Switch



DIGITAL SILENCE

Demo: Defeating MACsec Using Rogue Gateway Attacks

Dealing With Improvements to Peripheral Device Security

Improvements to Peripheral Device Security

Improved 802.1x support by peripheral devices:

- bypassing port security by looking for policy exceptions has become difficult

Improvements to Peripheral Device Security

Important caveat:

- Improved adoption of 802.1x does not necessarily imply strong port security for peripheral devices
- Back to EAP: most EAP methods have major security issues

Improvements to Peripheral Device Security

Adoption rates for secure EAP methods very poor across all device types:

- Secure EAP methods are often challenging to deploy at scale

We can expect these adoption rates to be even lower for peripheral devices:

- Already painful to configure
- Not always centrally manageable through Group Policy (besides, is a domain joined printer really a good idea?)

Improvements to Peripheral Device Security

What this means:

- Peripheral devices are still a viable attack vector for bypassing port security

EAP-MD5 Forced Reauthentication Attack

EAP-MD5 Forced Reauthentication Attack

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

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

EAP-MD5 Forced Reauthentication Attack

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

EAP-MD5 Forced Reauthentication Attack

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

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.

EAP-MD5 Forced Reauthentication Attack

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

EAP-MD5 Forced Reauthentication Attack

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

EAP-MD5 Forced Reauthentication Attack

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

Leveraging Rogue Gateway Attacks Against Peripheral Devices

Rogue Gateway Attacks Against Peripheral Devices

Other commonly used weak EAP methods used by peripheral devices include EAP-TTLS and EAP-PEAP: [13]

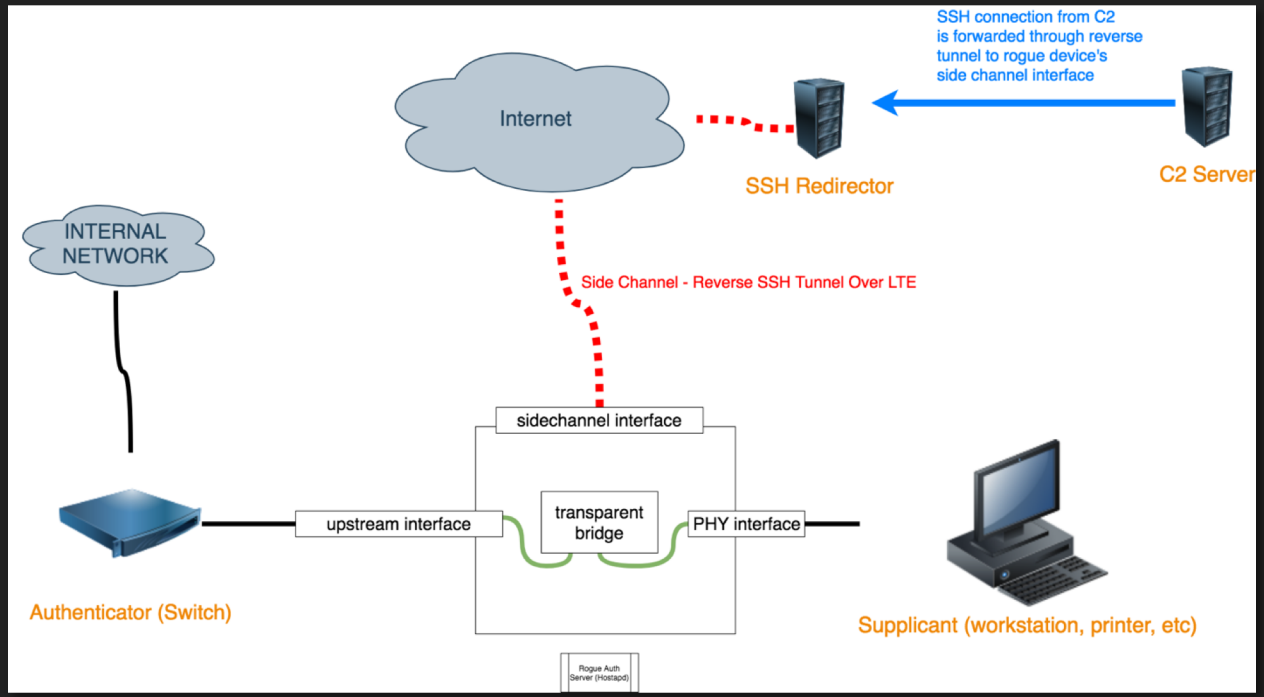
- Attacks are considerably more involved compared to EAP-MD5
- Authentication occurs through an encrypted tunnel, so a MITM is necessary to capture the challenge and response [13]

Rogue Gateway Attacks Against Peripheral Devices

Solution – use Rogue Gateway Attack:

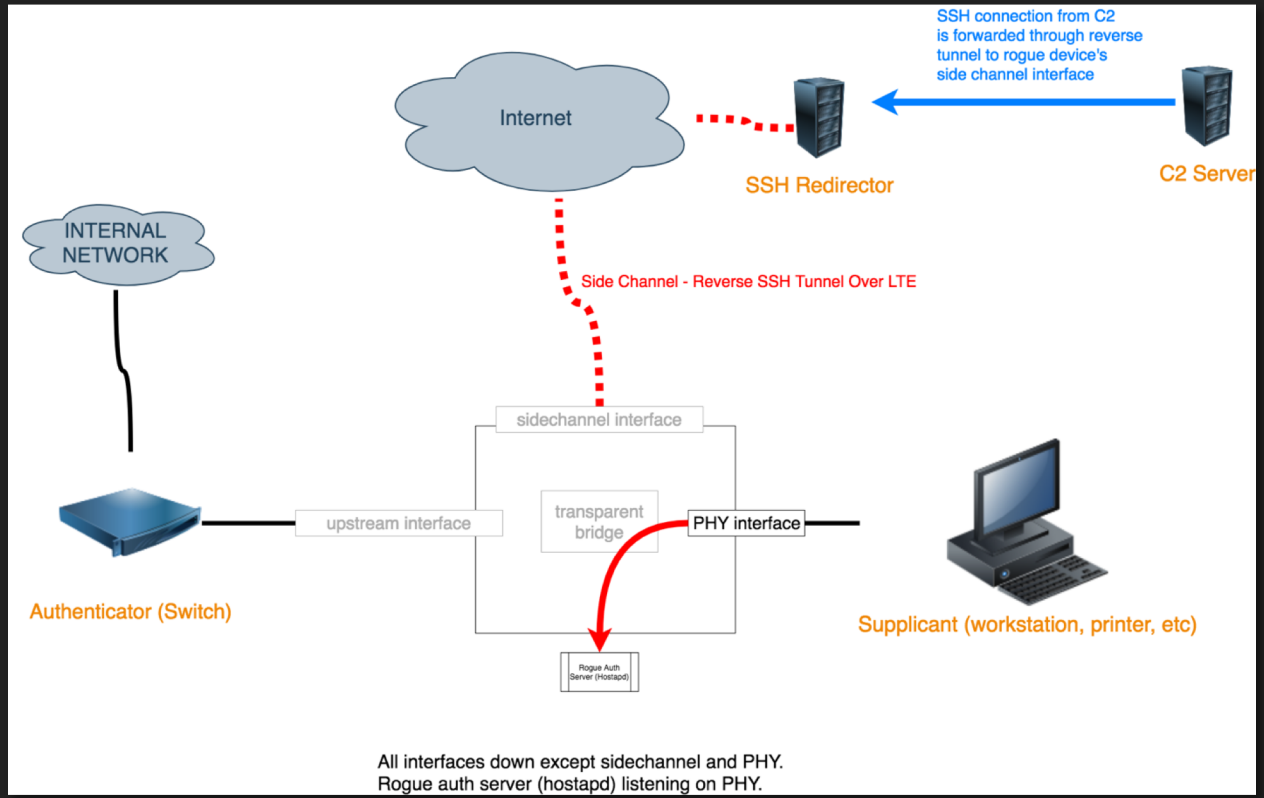
- No mechanical splitters needed this time: attack can be implemented in software using transparent bridge

Step 1: Plant the Device



DIGITAL SILENCE

Step 2: Perform the Attack



Closing Thoughts

Closing Thoughts

Our contributions:

- Rogue Gateway and Bait n Switch – Bypass 802.1x-2011 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_Shmoocon2008_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=ANoY7cqwzbsU93t3gE88UC_qqtG7cVvms7FRutz0KwK1oiBcEJMlQuUmpGSMMD7oZGyGmt4M2HaBhHFb07j8Gvmb_HWIE8rSfLKDvB0Al80u0cYwSNi5ugTP1JtFXsy1yZn8-85icVc32PpzxLJwRinf2UGzNbEdO97Wsc9xcjnc8A8MaFkPbUV5kwsMYHaxMiWwTcE-A8Dp49vv-tmk86pNMaeUeumBw_5vCZ6C3Pvc07hVbyTOsjqo6C6WpfVhd_M0BNW0RQtI&attredirects=0

DIGITAL SILENCE

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