Adrian Vollmer

THE PENTEST EXPERTS.

Lauschgerät

Gets in your victim’s traffic and out of yours

About me

- Used to be a cosmologist
- Pentester since 2015 at SySS
- Specialized on Windows networks
- Wrote *Seth* and presented at Black Hat USA Arsenal, Hacktivity, DACH-Security
- @mr_mitm

About Lauschgerät

- Helps you with all kinds of MitM attacks
- In particular bypassing 802.1X network access control and TLS inspection
- Written in Python and Bash
- Physical (Raspi or similar) or virtual
- Manageable via web interface (Flask)
- Modular concept
- `https://github.com/SySS-Research/Lauschgeraet`
- Focus: Attacks on the client; painless; automated

Microsoft Security Bulletin MS15-011 - Critical

10/11/2017 • 8 minutes to read •

Vulnerability in Group Policy Could Allow Remote Code Execution (3000483)

Published: February 10, 2015 | Updated: March 11, 2015

Version: 1.1

Executive Summary

This security update resolves a privately reported vulnerability in Microsoft Windows. The vulnerability could allow remote code execution if an attacker convinces a user with a domain-configured system to connect to an attacker-controlled network. An attacker who successfully exploited this vulnerability could take complete control of an affected system. An attacker could then install programs; view, change, or delete data; or create new accounts with full user rights.

This security update is rated Critical for all supported editions of Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows RT, Windows 8.1, Windows Server 2012 R2, and Windows RT 8.1. For more information, see the **Affected Software** section.

The security update addresses the vulnerability by improving how domain-configured systems connect to domain controllers prior to Group Policy accepting configuration data. For more information about the vulnerability, see the **Vulnerability Information** section.

To be protected from the vulnerability described in this bulletin, additional configuration by a system administrator is required in addition to deploying this security update. For more information about this update, see Microsoft Knowledge Base Article 3000483.

Wagging the Dog: Abusing Resource-Based Constrained Delegation to Attack Active Directory

28 January 2019 • Elad Shamir • 41 min read

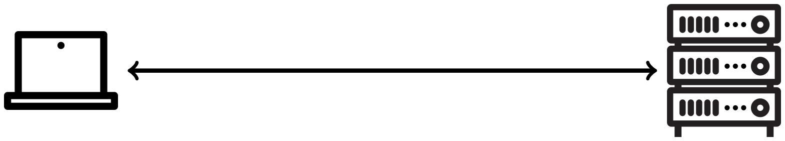
Back in March 2018, I embarked on an arguably pointless crusade to prove that the TrustedToAuthForDelegation attribute was meaningless, and that “protocol transition” can be achieved without it. I believed that security wise, once constrained delegation was enabled (msDS-AllowedToDelegateTo was not null), it did not matter whether it was configured to use “Kerberos only” or “any authentication protocol”.

I started the journey with Benjamin Delpy’s (@gentilkiwi) help modifying Kekeo to support a certain attack that involved invoking S4U2Proxy with a silver ticket without a PAC, and we had partial success, but the final TGS turned out to be unusable. Ever since then, I kept coming back to it, trying to solve the problem with different approaches but did not have much success. Until I finally accepted defeat, and ironically then the solution came up, along with several other interesting abuse cases and new attack techniques.

TL;DR

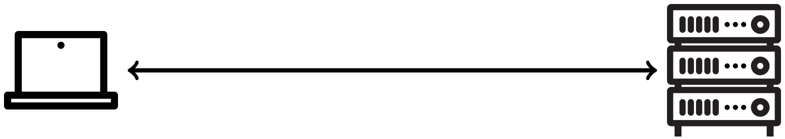
Setup

Before:

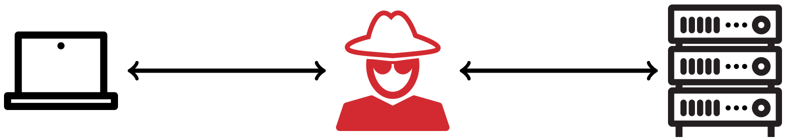


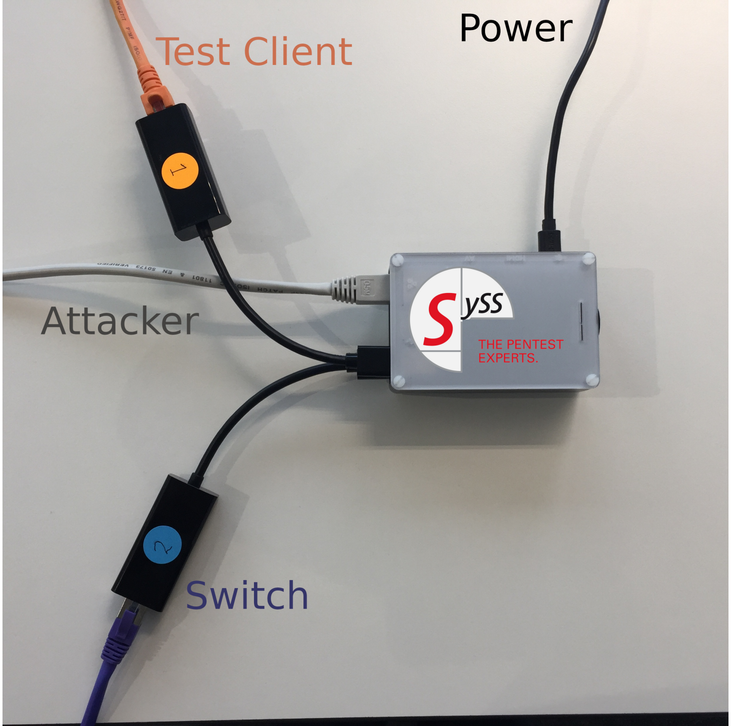
Setup

Before:



After:




Power
Test Client
Attacker
SySS
THE PENTEST EXPERTS.
Switch

Sailing the Seven ~~Seas~~Layers

1. Unplug the network cable
2. Source-NAT Ethernet frames
3. Source-NAT IP packets
4. Redirect specific TCP connections
5. Perform TLS inspection
6. ???
7. Parse and modify HTTP messages

802.1X bypass – Why not just use a hub?

- ➔ Reminder: (wired) 802.1X is certificate-based authentication at layer 2 – ethernet port does not work until it's authenticated
- ➔ Theoretically, a hub should work: let the legitimate client authenticate the port and use the same MAC address
- ➔ The problem is a race condition: if one ACK is received by the wrong client, it sends a RST
- ➔ Could disconnect the legitimate client, but port needs to be re-authorized regularly

802.1X bypass – better idea

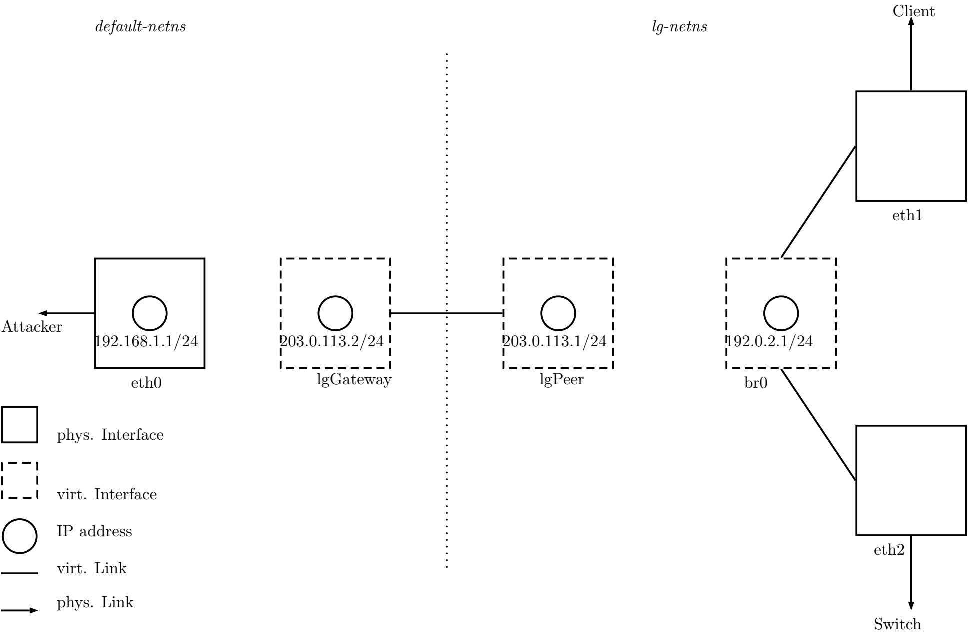
- Create new network name space
- Put two interfaces there (**`eth1`** and **`eth2`**)
- Create network bridge (**`br0`**)
- Adjust source addresses (SNAT) with **`iptables`** and **`ebtables`**
- Manually fix ARP table
- Inject own traffic by routing it via an IP address on **`br0`**

802.1X bypass – better idea

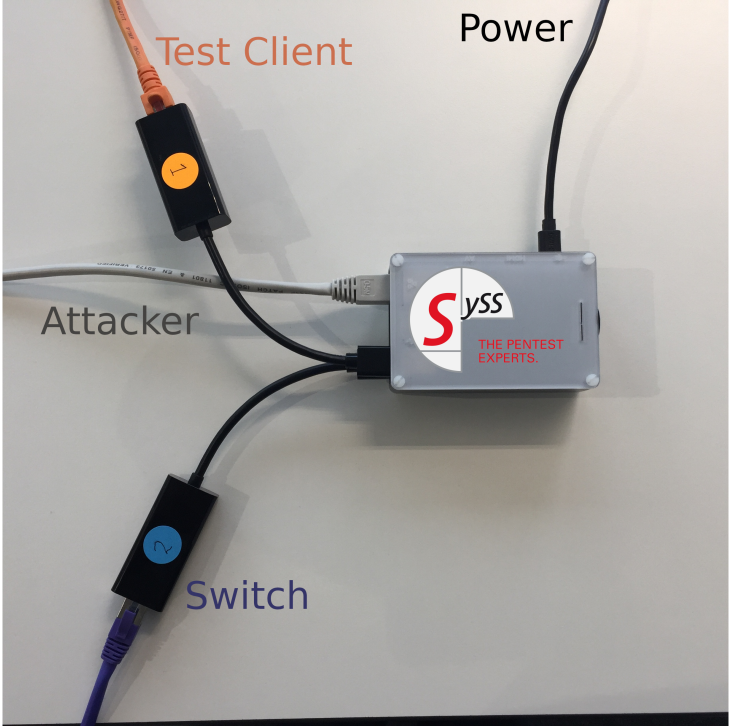
- ➔ Create new network name space
- ➔ Put two interfaces there (`eth1` and `eth2`)
- ➔ Create network bridge (`br0`)
- ➔ Adjust source addresses (SNAT) with `iptables` and `ebtables`
- ➔ Manually fix ARP table
- ➔ Inject own traffic by routing it via an IP address on `br0`

A Bridge Too Far. Defeating Wired 802.1X with a Transparent Bridge Using Linux (Alva Lease 'Skip' Duckwall IV)[1]

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



default-netns
lg-netns
Client
eth1
Attacker
192.168.1.1/24
eth0
203.0.113.2/24
lgGateway
203.0.113.1/24
lgPeer
192.0.2.1/24
br0
eth2
Switch
phys. Interface
virt. Interface
IP address
virt. Link
phys. Link


Power
Test Client
Attacker
SySS
THE PENTEST EXPERTS.
Switch

802.1X bypass – a caveat

Standard mandates: EAPoL packets must not traverse a network bridge

802.1X bypass – a caveat

Standard mandates: EAPoL packets must not traverse a network bridge

➔ Solution 1: patch the Linux kernel

802.1X bypass – a caveat

Standard mandates: EAPoL packets must not traverse a network bridge

- ➔ Solution 1: patch the Linux kernel
- ➔ Solution 2: use something like scapy to forward them manually

802.1X bypass – a caveat

Standard mandates: EAPoL packets must not traverse a network bridge

- ➔ Solution 1: patch the Linux kernel
- ➔ Solution 2: use something like scapy to forward them manually
- ➔ Solution 3: `echo 8 > /sys/class/net/br0/bridge/group_fwd_mask` (since kernel 3.2[2])

[2] `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=515853ccecc6987dfb8ed809dd8bf8900286f29e`

802.1X bypass – the search for the gateway

We need the IP and MAC address of the client and the MAC address of the gateway. How can we determine that from just observing traffic?

802.1X bypass – the search for the gateway

We need the IP and MAC address of the client and the MAC address of the gateway. How can we determine that from just observing traffic?

- ➔ Sniffing DHCP responses could work, but sometimes clients use a static IP config
- ➔ Instead, see where DNS or Kerberos requests go:

```
tcpdump -i br0 -w "$TCPDUMP_FILE" -c1 \
    "udp dst port 53 or tcp dst port 88" 2> /dev/null
```

- ➔ They're usually on a different subnet and thus go via the gateway
- ➔ Don't forget to statically set the ARP entries
- ➔ ... including a fake entry with a bogus gateway

802.1X bypass – injecting traffic

Because attacker and victim client are using the same IP and MAC, incoming packets can't be distinguished.
Solution: Use `iptables` to fix source ports in the range 61000-62000 for attacker traffic:

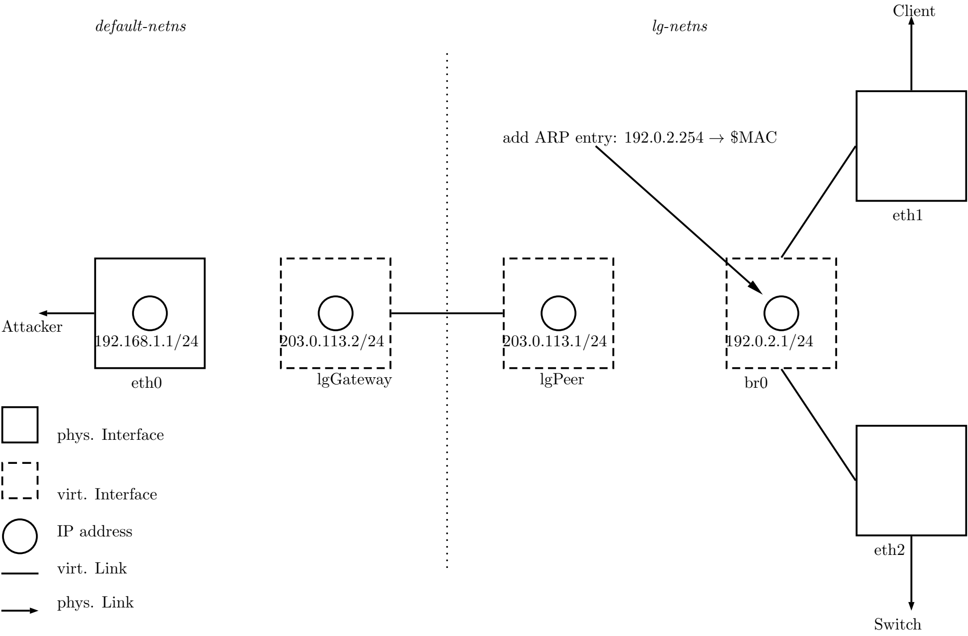
```
iptables -t nat -A POSTROUTING -o br0 -s $ATTACKER_NET \
    -p tcp -j SNAT --to $CLIENT_IP:61000-62000
ebtables -t nat -A POSTROUTING -s $SWITCH_MAC -o br0 \
    -j snat --to-src $CLIENT_MAC
```

802.1X bypass – injecting traffic

Because attacker and victim client are using the same IP and MAC, incoming packets can't be distinguished.
Solution: Use `iptables` to fix source ports in the range 61000-62000 for attacker traffic:

```
iptables -t nat -A POSTROUTING -o br0 -s $ATTACKER_NET \
    -p tcp -j SNAT --to $CLIENT_IP:61000-62000
ebtables -t nat -A POSTROUTING -s $SWITCH_MAC -o br0 \
    -j snat --to-src $CLIENT_MAC
```

Ephemeral Source Ports:
XP/2003: 1025 – 5000
Vista/7/2008/10: 49152 - 65535 (as recommended by IANA)
many Linux kernels: 32768 to 60999


default-netns
lg-netns
Client
add ARP entry: 192.0.2.254 → $MAC
eth1
Attacker
192.168.1.1/24
eth0
203.0.113.2/24
lgGateway
203.0.113.1/24
lgPeer
192.0.2.1/24
br0
eth2
Switch
phys. Interface
virt. Interface
IP address
virt. Link
phys. Link

Injecting and modifying traffic

Injecting: Just add a route via our gateway IP
Modifying: Add `iptables` rule
Example: `1.2.3.4:443` → `203.0.113.1:443` (a malicious service)

```
iptables -t nat -A PREROUTING -i br0 \
    -p tcp --dport 443 --destination 1.2.3.4 \
    -j DNAT --to-destination 203.0.113.1:443
```

TLS Inspection

- Most interesting traffic is encrypted
- Need TLS proxy, e.g. `https://github.com/ickerwx/tcpproxy`
- Desirable features:
  - Automatically find original destination
  - Create a new cert which looks identical to the original
  - Watch clear text traffic in Wireshark

TLS Inspection – original destination

Goal: Automatically find original destination

Easy, if connection was redirected with `iptables`:

```
SO_ORIGINAL_DST = 80
sockaddr_in = conn.getsockopt(socket.SOL_IP,
                              SO_ORIGINAL_DST,
                              16)
_, port, a, b, c, d = struct.unpack('!HHBBBB',
                                    sockaddr_in[:8])
print('Original destination was: %d.%d.%d.%d:%d' %
      (a, b, c, d, port))
```

TLS Inspection – clone cert

Goal: Create a certificate that looks like the original as much as possible

- ➔ Create a new key pair
- ➔ Parse ASN.1 structure
- ➔ Replace public key
- ➔ Modify serial number, issuer and CA key identifier
- ➔ (Only the issuer is human readable)
- ➔ Re-sign key with new private key

`https://github.com/SySS-Research/clone-cert`

TLS Inspection – Wireshark

Goal: Have an extra interface for the decrypted traffic

Problem: libpcap does not see traffic on virtual interfaces

Solution: Redirect traffic over an interface in another network namespace

TLS Inspection – Wireshark

```
            S1  S2            S5  S6
src ---->----o  o              o  o---->---- orig.dst
                 \            /
                  \          /            namespace 1
=====================================================
                    \      /              namespace 2
                     \    /
                      o  o
                     S3  S4

o = TCP socket
```

TLS Inspection – tlseraser

- ➔ Written in Python
- ➔ Automatic detection of TLS handshake
- ➔ Modular concept for tampering with traffic
- ➔ One example module will be in the demo

`https://github.com/AdrianVollmer/tlseraser`

Demo

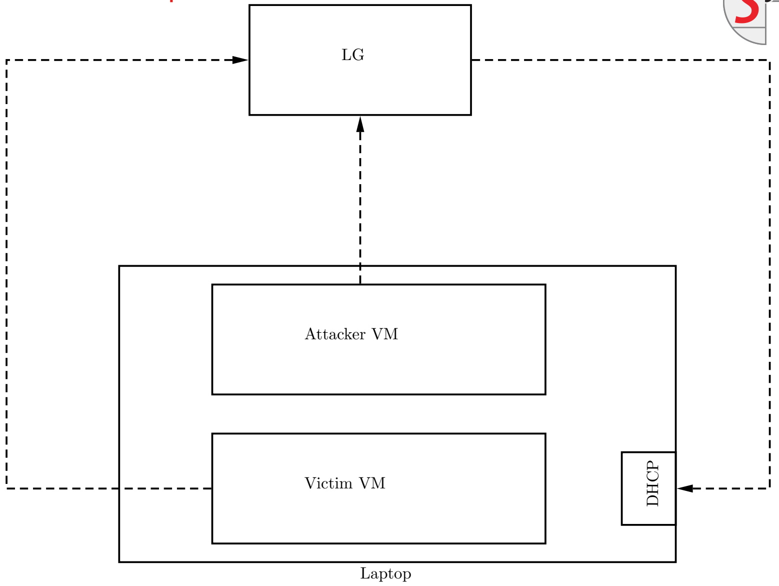
Demo time! However ...

Four devices is two too much:

- ➔ Victim client
- ➔ Victim switch
- ➔ Lauschgerät
- ➔ Attacker laptop

Will show you the physical setup, but virtual is recommended (unless you want to attack wifi devices)

Demo Setup



For the record

- https://github.com/SySS-Research/Lauschgeraet
- https://github.com/AdrianVollmer/tlseraser
- https://github.com/SySS-Research/clone-cert
- https://www.gremwell.com/marvin-mitm-tapping-dot1x-links
- https://www.defcon.org/images/defcon-19/dc-19-presentations/Duckwall/DEFCON-19-Duckwall-Bridge-Too-Far.pdf
- https://github.com/ickerwx/tcpproxy

THE PENTEST EXPERTS

WWW.SYSS.DE