CHAPTER 3

ENUMERATING A VOIP NETWORK

It pays to be obvious, especially if you have a reputation for subtlety.

—Isaac Asimov

Now that the hacker has developed a list of active IP addresses and services in your VoIP environment, the next logical step is to probe those services aggressively in search of known weaknesses and vulnerabilities. This process is called *enumeration* and is more intrusive and noisy than the reconnaissance techniques we have covered so far. In the last chapter, we compared scanning to a masterful art thief walking around the Louvre checking for doors. Enumeration can best be compared to that same thief going one step further and rattling door knobs loudly until he finds an unlocked one.

The goal of enumeration is to leverage the target's open services to glean sensitive information that can assist in launching further attacks. For example, an effective enumeration technique covered in this chapter involves brute forcing VoIP PBXs and phones in order to generate a list of valid phone extensions. Gleaning the phone extensions that are active on a VoIP network is necessary for attacks such as INVITE floods and REGISTER hijacking, which are covered in Chapters 12 and 13, respectively.

Enumerating common VoIP infrastructure support services, such as TFTP and SNMP, can also often unearth a treasure trove of sensitive configuration information. As you saw in the Google hacking exercise in Chapter 1, many VoIP phones come installed with active web servers on them by default so that an administrator can easily configure them. Unfortunately, these web interfaces can reveal very sensitive device and network configuration details given the right enumeration techniques.

This chapter will discuss some of the enumeration techniques relevant to SIP-based devices, as well as targeting the highly exposed VoIP support services such as TFTP, SNMP, and others. The chapter begins, however, with review of SIP and RTP.

SIP 101

The majority of techniques covered in this chapter, and in the rest of this book, assume a basic understanding of the *Session Initiation Protocol (SIP)* (http://www.cs.columbia .edu/sip/). While it goes beyond the scope of this book to delve thoroughly into the complete workings of SIP, it will be helpful to review some of the basics.

Simply put, SIP allows two speaking parties to set up, modify, and terminate a phone call between the two of them. SIP is a text-based protocol and is most similar, at first glance, to the HTTP protocol. SIP messages are composed of specific requests and responses that are detailed here.

SIP URIs

A SIP *Uniform Resource Indicator (URI)* is how users are addressed in the SIP world (RFC 3261). The general format of a SIP URI is

```
sip:user:password@host:port;uri-parameters?headers
```

Some example SIP URIs taken directly from the RFC are

```
   sip:alice@atlanta.com
   sip:2125551212@example.com
   sip:alice:secretword@atlanta.com;transport=tcp
   sip:+1-212-555-1212:1234@gateway.com;user=phone
   sip:alice@192.0.2.4:5060
   sip:atlanta.com;method=REGISTER?to=alice%40atlanta.com
   sip:alice;day=tuesday@atlanta.com
```

SIP Architecture Elements

There are five logical core components in SIP architecture. Many of the server functions detailed here are often consolidated into one or two server applications.

- **User agents (UA)** Any client application or device that initiates a SIP connection, such as an IP phone, PC softphone, PC instant messaging client, or mobile device. The user agent can also be a gateway that interacts with the PSTN.
- **Proxy server** A proxy server is a server that receives SIP requests from various user agents and routes them to the appropriate next hop. A typical call traverses at least two proxies before reaching the intended callee.
- **Redirect server** Sometimes it is better to offload the processing load on proxy servers by introducing a redirect server. A redirect server directs incoming requests from other clients to contact an alternate set of URIs.
- **Registrar server** A server that processes REGISTER requests. The registrar processes REGISTER requests from users and maps their SIP URI to their current location (IP address, username, port, and so on). For instance, sip:dave@hackingexposed.com might be mapped to something like sip: dave@192.168.1.100:5060, which is the softphone from which I just registered.
- **Location server** The location server is used by a redirect server or a proxy server to find the callee's possible location. This function is most often performed by the registrar server.

A typical SIP-based call flow is best represented by the illustration in the section, "Typical Call Flow," later in this chapter.

SIP Requests

SIP requests can be used in a standalone sense or in a dialog with other SIP requests and responses. The following is a brief overview of the most common requests used in call initiation and teardown:

SIP Request	Purpose	RFC Reference
INVITE	Initiates a conversation.	RFC 3261
BYE	Terminates an existing connection between two users in a session.	RFC 3261
OPTIONS	Determines the SIP messages and codecs that the UA or server understands.	RFC 3261
REGISTER	Registers a location from a SIP user.	RFC 3261
ACK	Acknowledges a response from an INVITE request.	RFC 3261
CANCEL	Cancels a pending INVITE request, but does not affect a completed request (for instance, stops the call setup if the phone is still ringing).	RFC 3261
REFER	Transfers calls and contacts external resources.	RFC 3515
SUBSCRIBE	Indicates the desire for future NOTIFY requests.	RFC 3265
NOTIFY	Provides information about a state change that is not related to a specific session. (For example, Windows Messenger uses a SUBSCRIBE method to get contacts, groups, and allow and block lists from the server. Microsoft Live Communications Server 2003 uses a NOTIFY to transfer this information.)	RFC 3265

SIP Responses

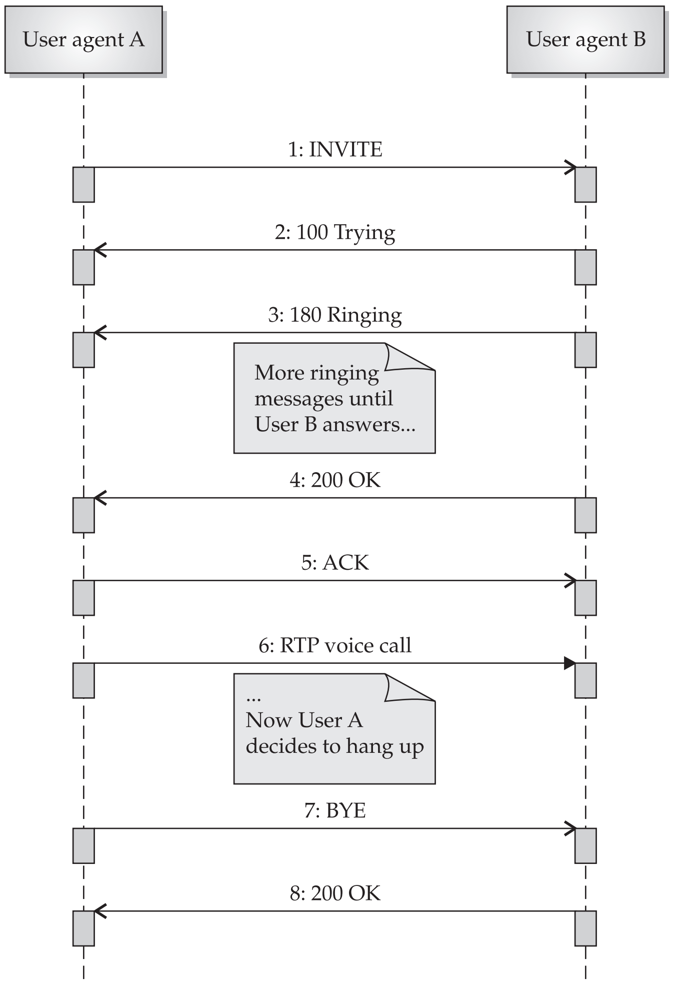
SIP responses (RFC 2543) are three-digit codes much like HTTP (for example, 200 OK, 404 Not Found, and so on). The first digit indicates the category of the response. The entire range of possible responses to a SIP request is as follows:

Response	Category	Codes
1*xx* responses	Information responses	100 Trying 180 Ringing 181 Call Is Being Forwarded 182 Queued 183 Session Progress
2*xx* responses	Successful responses	200 OK

Response	Category	Codes
3*xx* responses	Redirection responses	300 Multiple Choices 301 Moved Permanently 302 Moved Temporarily 303 See Other 305 Use Proxy 380 Alternative Service
4*xx* responses	Request failure responses	400 Bad Request 401 Unauthorized 402 Payment Required 403 Forbidden 404 Not Found 405 Method Not Allowed 406 Not Acceptable 407 Proxy Authentication Required 408 Request Timeout 409 Conflict 410 Gone 411 Length Required 413 Request Entity Too Large 414 Request URI Too Large 415 Unsupported Media Type 420 Bad Extension 480 Temporarily Not Available 481 Call Leg/Transaction Does Not Exist 482 Loop Detected 483 Too Many Hops 484 Address Incomplete 485 Ambiguous 486 Busy Here
5*xx* responses	Server failure responses	500 Internal Server Error 501 Not Implemented 502 Bad Gateway 503 Service Unavailable 504 Gateway Time-out 505 SIP Version Not Supported
6*xx* responses	Global failure responses	600 Busy Everywhere 603 Decline 604 Does Not Exist Anywhere 606 Not Acceptable

Typical Call Flow

Now to see the SIP requests and responses in action, let's look at a fairly standard call setup between two users. The actual example is shown using a Vonage softphone client as User agent A (7035551212) calling User agent B (5125551212).



1. The Vonage user sends an INVITE to User B to initiate a phone call.

The Session Description Protocol (SDP RFC 2327) is used to describe all media codecs supported by the Vonage user.

```
INVITE sip:15125551212@sphone.vopr.vonage.net SIP/2.0
Via: SIP/2.0/UDP 12.39.18.123:5060;rport;branch=z9hG4bK666
12D61E45C460BA4624A77E6E51AA1
From: Vonage User
sip:17035551212@sphone.vopr.vonage.net>;tag=3010128031
To: <sip:15125551212@sphone.vopr.vonage.net>
Contact: <sip:17035551212@12.39.18.123:5060>
Call-ID: 805C3881-E9F6-402E-BBD8-181A2B9C2AC6@12.39.18.123
CSeq: 10814 INVITE
Max-Forwards: 70
Content-Type: application/sdp
User-Agent: X-PRO Vonage release 1105x
Content-Length: 244

v=0
o=17035551212 44428031 44428065 IN IP4 12.39.18.123
s=X-PRO Vonage
c=IN IP4 12.39.18.123
t=0 0
m=audio 8000 RTP/AVP 0 18 101
a=rtpmap:0 pcmu/8000
a=rtpmap:18 G729/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv
```

2. User B receives the request (his phones rings).

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 12.39.18.123:5060;rport;branch=z9hG4bKA53
5C55954034DE8980460B33AC67DDD
From: Vonage User <sip:17035551212@sphone.vopr.vonage.
net>;tag=3010128031
To: <sip:15125551212@sphone.vopr.vonage.net>
Call-ID: 805C3881-E9F6-402E-BBD8-181A2B9C2AC6@12.39.18.123
CSeq: 10815 INVITE
Max-Forwards: 15
Content-Length: 0<F255D>
```

3. While User B's phone is ringing, he sends updates (TRYING, SESSION PROGRESS, and so on).

```
SIP/2.0 183 Session Progress
Via: SIP/2.0/UDP 12.39.18.123:5060;rport;branch=z9hG4bKA53
5C55954034DE8980460B33AC67DDD
From: Vonage User <sip:17035551212@sphone.vopr.vonage.
net>;tag=3010128031
To: <sip:15125551212@sphone.vopr.vonage.
net>;tag=gK0ea08a79
Call-ID: 805C3881-E9F6-402E-BBD8-181A2B9C2AC6@12.39.18.123
CSeq: 10815 INVITE
Contact: <sip:15125551212@216.115.20.41:5061>
Max-Forwards: 15
Content-Type: application/sdp
Content-Length:<F255D>   238

v=0
o=Sonus_UAC 14354 30407 IN IP4 69.59.245.131
s=SIP Media Capabilities
c=IN IP4 69.59.245.132
t=0 0
m=audio 21214 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv
a=maxptime:20
```

4. User B picks up the phone and sends an OK response to the caller.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 12.39.18.123:5060;rport;branch=z9hG4bK493
C01C844624AAE8C1A8CE04A4237E3
From: Vonage User <sip:17035551212@sphone.vopr.vonage.
net>;tag=1667903552
To: Vonage User <sip:17035551212@sphone.vopr.vonage.net>
Call-ID: 6E44DD2552ED417EB0B92A6F3C640E80@sphone.vopr.
vonage.net
CSeq: 1410 REGISTER
Contact: "Vonage User" <sip:17035551212@12.39.18.123:5060>
;expires=20
Content-Length: 0<F255D>
```

5. The Vonage user responds with an ACK acknowledgment.	`ACK sip:15125551212@216.115.20.41:5061 SIP/2.0`
	`Via: SIP/2.0/UDP 12.39.18.123:5060;rport;branch=z9hG4bK6B5`
	`3C0C1ECFD4B7DB26C6CC5F224B292`
	`From: Vonage User <sip:17035551212@sphone.vopr.vonage.`
	`net>;tag=3010128031`
	`To: <sip:15125551212@sphone.vopr.vonage.`
	`net>;tag=1091505090`
	`Contact: <sip:17035551212@12.39.18.123:5060>`
	`Call-ID: 805C3881-E9F6-402E-BBD8-181A2B9C2AC6@12.39.18.123`
	`CSeq: 10815 ACK`
	`Max-Forwards: 70`
	`Content-Length: 0<F255D>`

6. The conversation is established directly between the two parties.	RTP packets are exchanged in both directions carrying the conversation.

7. User B hangs up and sends a BYE message.	`BYE sip:17035551212@12.39.18.123:5060 SIP/2.0`
	`Via: SIP/2.0/UDP 216.115.20.41:5061`
	`Via: SIP/2.0/UDP 69.59.240.166;branch=z9hG4bK07e88f99`
	`From: <sip:15125551212@sphone.vopr.vonage.`
	`net>;tag=1091505090`
	`To: Vonage User <sip:17035551212@sphone.vopr.vonage.`
	`net>;tag=3010128031`
	`Call-ID: 805C3881-E9F6-402E-BBD8-181A2B9C2AC6@12.39.18.123`
	`CSeq: 10816 BYE`
	`Max-Forwards: 15`
	`Content-Length: 0<F255D>`

8. The Vonage user accepts the BYE message, and sends an OK as an acknowledgment.	`SIP/2.0 200 OK`
	`Via: SIP/2.0/UDP 12.39.18.123:5060;rport;branch=z9hG4bKE31`
	`C9EC9A1764679A417E3B5FBBF425A`
	`From: <sip:17035551212@inbound2.vonage.net>;tag=2209518249`
	`To: <sip:15125551212@206.132.91.13>;tag=448318763`
	`Call-ID: E630553E-E44911DA-BC08C530-3979085C@206.132.91.13`
	`CSeq: 10816 BYE`
	`Max-Forwards: 14`
	`Content-Length: 0<F255D>`

Further Reading

This brief summary of SIP is meant only as a refresher and companion to many of the SIP-based attacks discussed throughout the book. For a more thorough reference guide on SIP, we highly recommend reading *SIP Beyond VoIP* by Henry Sinnreich, Alan B. Johnson, and Robert J. Sparks (VON Publishing, 2005).

RTP 101

The *Real-Time Protocol (RTP)* is an IETF standard, documented in RFC 3550. While different vendor systems use various signaling protocols, virtually every vendor uses RTP for the audio. This is important because RTP will be present as a target protocol in any VoIP environment. RTP is a simple protocol, generally riding on top of UDP. RTP provides payload type identification, sequence numbering, timestamping, and delivery monitoring. RTP does not provide mechanisms for timely delivery or other QoS capabilities. It depends on lower layer protocols to do this. RTP also does not assure delivery or order of packets. However, RTP's sequence numbers allow applications, such as an IP phone, to check for lost or out of order packets.

RTP includes the RTP control protocol (RTCP), which is used to monitor the quality of service and to convey information about the participants in an ongoing session. VoIP endpoints should update RTCP, but not all do.

RTP is a binary protocol, which adds the following header to each UDP packet:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The first twelve bytes are present in every RTP packet, while the list of CSRC identifiers is present only when inserted by a mixer. The fields are defined as follows:

- **Version (V): 2 bits** This field identifies the version of RTP. The version defined by RFC 3550 specification is two (2).
- **Padding (P): 1 bit** If the padding bit is set, the packet contains one or more additional padding bytes at the end that are not part of the payload.
- **Extension (X): 1 bit** If the extension bit is set, the fixed header *must* be followed by exactly one header extension, with a format defined in RFC 3550.
- **CSRC count (CC): 4 bits** The CSRC count contains the number of CSRC identifiers that follow the fixed header.
- **Marker (M): 1 bit** The interpretation of the marker is defined by a profile. It is intended to allow significant events such as frame boundaries to be marked in the packet stream.

- **Payload type (PT): 7 bits** This field identifies the format of the RTP payload and determines its interpretation by the application.
- **Sequence number: 16 bits** The sequence number increments by one for each RTP data packet sent and may be used by the receiver to detect packet loss and to restore packet order. The initial value of the sequence number should be random (it should not be 0) to make known–plaintext attacks on encryption more difficult, even if the source itself does not encrypt according to the method because the packets may flow through a translator that does.
- **Timestamp: 32 bits** The timestamp reflects the sampling time of the first byte in the RTP payload. The clock used to calculate the timestamp must have sufficient resolution to allow endpoints to perform synchronization and jitter calculations.
- **SSRC: 32 bits** The SSRC field identifies the synchronization source. This identifier should be chosen randomly, so that no two synchronization sources within the same RTP session should have the same SSRC. Although the probability of multiple sources choosing the same identifier is low, all RTP implementations must be prepared to detect and resolve duplicates.

The presence of the sequence number, timestamp, and SSRC makes it difficult for an attacker to inject malicious RTP packets into a stream. The attacker needs to be performing a man-in-the-middle (MITM) attack or at least be able to monitor the packets, so that the malicious packets include the necessary SSRC, sequence number, and timestamp. If these values are not correct, the target endpoint will ignore the malicious packets.

RTP audio is sampled at a transmitting endpoint over a given time period. A number of samples are collected and then typically compressed by a *compressor/decompressor (codec)*. For example, the ITU has created and published specifications for several popular audio codecs, such as G.711, G.723, G.726, and G.729.

G.711 is the most commonly used codec, particularly for LAN-based VoIP calls. G.711 uses Pulse Code Modulation (PCM) and requires 64 Kbps. Other codecs, such as G.729, which uses Adaptive Differential Pulse Code Modulation (ADPCM), only require 8 Kbps. These codecs are often used over lower-bandwidth links.

Codecs such as G.711 are *waveform coders,* which means they are not aware of human speech characteristics. They sample the audio at a specific frequency, such as 8 KHz (8000 times a second). Codecs such as G.728 are *vocoders* and are aware of human speech characteristics. Only a limited number of sounds are uttered during human speech. A vocoder identifies the phonemes uttered and looks up codes in a table corresponding to that sound. Codes are then transmitted instead of the sampled audio itself. Vocoders are significantly more computationally intensive than waveform coders, but generally require less bandwidth. They do, however, generate poor sounding audio.

G.711 audio is carried as a 160-byte payload within an RTP message. RTP messages are transmitted within UDP packets at a rate of 50 Hz (in other words every 20 milliseconds (ms)). The sequence number field within the RTP header begins at some random number and increases monotonically by 1 with each RTP packet transmitted. For G.711, the timestamp begins at some random number and increases monotonically by 160 with each RTP packet transmitted.

When an audio session is being set up between two VoIP endpoints, SIP signaling messages (for example, INVITE, OK) typically carry a Session Description Protocol (SDP) message within their payload. Session Description Protocol message exchange is the mechanism by which endpoints negotiate, or state, which codec or codecs they care to support for encoding or decoding audio during the session. One codec may be used to compress transmitted audio, and a different codec may be used to decompress received audio.

The codec determines the time quantum over which audio is sampled and the rate that RTP-bearing packets are transmitted. The selected transmission rate is fixed. Whether or not the packets arrive at the fixed rate depends on the underlying network. Packets may be lost, arrive out of order, or be duplicated. Receiving endpoints must take this into account. Endpoints use an audio *jitter buffer* that collects, resequences, fills in gaps, and if necessary, deletes samples in order to produce the highest quality audio playback. The sequence number and timestamp in the RTP header are used for this purpose.

BANNER GRABBING WITH NETCAT

The first step in enumerating a VoIP network involves a technique called *banner grabbing* or *banner scraping*. Banner grabbing is simply a method of connecting to a port on a remote target to identify more information about the associated service running on that port. Banner grabbing is one of the easiest and highest yield attack methods that hackers employ to inventory your VoIP applications and hardware. By connecting to most standard services and applications, they can glean the specific service type (for example, Apache HTTPd, Microsoft IIS, and so on), the service version (for example, Apache HTTPd 1.3.37, Microsoft IIS 6.0, and so on), and tons of other useful information about the target.

Banner Grabbing

Popularity:	7
Simplicity:	7
Impact:	4
Risk Rating:	**6**

Most manual banner grabbing can be easily accomplished using the command-line tool, Netcat (http://netcat.sourceforge.net/). For instance, building off of the Nmap results from the previous chapter, let's try to identify the web server running on 192.168.1.103:

```
[root@attacker] nc 192.168.1.103 80
GET / HTTP/1.1
```

```
HTTP/1.1 400 Bad Request
Date: Sun, 05 Mar 2006 22:15:40 GMT
Server: Apache/2.0.46 (CentOS)
Content-Length: 309
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr />
<address>Apache/2.0.46 (CentOS) Server at 192.168.1.103 Port 80</address>
</body></html>
```

By examining the error response, we can clearly see that the web server type is Apache HTTPd version 2.0.46 running on the operating system CentOS.

Looking again at the Nmap results from the previous chapter, we see that both UDP and TCP port 5060 is active on host 192.168.1.104. Using the same technique with Netcat, we can manually identify information about the (likely SIP) service attached to that port by sending an OPTIONS method with the following four lines (remember that 192.168.1.120 is the attacking IP address):

```
[root@attacker]# nc 192.168.1.104 5060
OPTIONS sip:test@192.168.1.104 SIP/2.0
Via: SIP/2.0/TCP 192.168.1.120;branch=4ivBcVj5ZnPYgb
To: alice <sip:test@192.168.1.104>
Content-Length: 0

SIP/2.0 404 Not Found
Via: SIP/2.0/TCP 192.168.1.120;branch=4ivBcVj5ZnPYgb;received=192.168.1.103
To: alice <sip:test@192.168.1.104>;tag=b27e1a1d33761e85846fc98f5f3a7e58.0503
Server: Sip EXpress router (0.9.6 (i386/linux))
Content-Length: 0
Warning: 392 192.168.1.104:5060 "Noisy feedback tells:  pid=29801 req_src_
ip=192.168.1.120 req_src_port=32773 in_uri=sip:test@192.168.1.104 out_
uri=sip:test@192.168.1.104 via_cnt==1"
```

As you can see, we can easily determine the exact SIP application and version by looking at the error response message (SIP EXpress Router 0.9.6 running on Linux).

Why is it important to identify the versions of these particular services? Outdated versions of applications are often vulnerable to exploits that have been published and archived for anyone to view. For instance, looking at http://secunia.com/advisories/8119/, we read:

```
Description:
IPTel has confirmed vulnerabilities in the SIP (Session Initiation Protocol)
implementation in all versions of SIP Express Router up to 0.8.9.

The vulnerabilities have been identified in the INVITE message used by two
SIP-endpoints during the initial call setup. The impact of successful ex-
ploitation of the vulnerabilities has not been disclosed but could poten-
tially result in a compromise of a vulnerable device.

Solution:
Upgrade to version 0.8.10 and apply patch:
```

Fortunately (unfortunately for the attacker) our version of SER is the latest and greatest at version 0.9.6. Similarly, there are many other vulnerabilities published for Apache, Cisco, and most other services and devices we'll be looking at in this chapter. You can find many of them through simple searches on online vulnerability databases:

- Symantec's SecurityFocus (http://www.securityfocus.com/bid)
- Secunia (http://www.secunia.com)
- Open Source Vulnerability Data Base (http://www.osvdb.org)
- National Vulnerability Database (http://nvd.nist.gov)

Automated Banner Grabbing

Using the previous SIP banner-grabbing example, the tool SiVuS (http://www.vopsecurity.org) can automate this process with a nice graphical interface (see Figure 3-1).

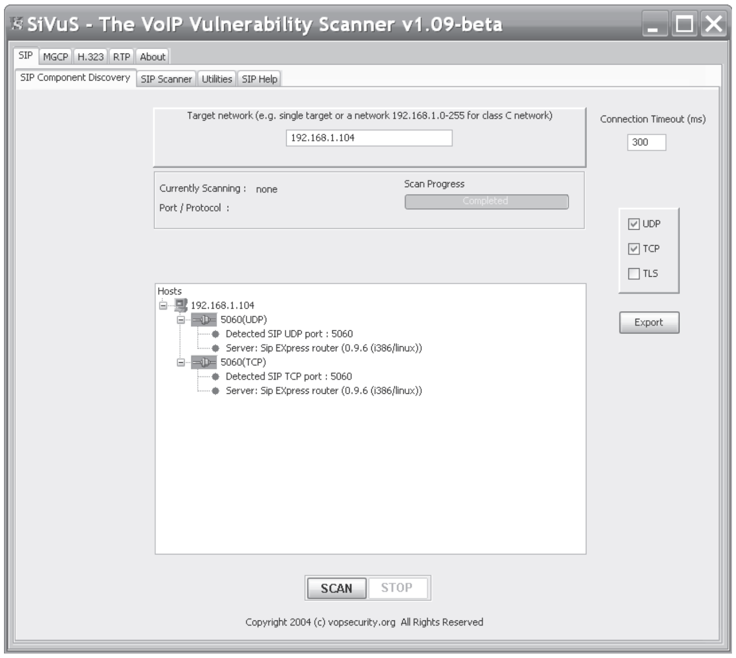
Another SIP scanning/fingerprinting tool called smap by Hendrick Scholz (http://www.wormulon.net/files/pub/smap-blackhat.tar.gz) actually analyzes SIP message responses to determine the type of device it's probing. For example,

```
$ ./smap -o 89.53.17.208/29

smap 0.4.0-cvs <hscholz@raisdorf.net> http://www.wormulon.net/

Host 89.53.17.208:5060: (ICMP OK) SIP timeout
Host 89.53.17.209:5060: (ICMP OK) SIP enabled
AVM FRITZ!Box Fon Series firmware: 14.03.(89|90) (Oct 28 2005)
Host 89.53.17.210:5060: (ICMP timeout) SIP timeout
Host 89.53.17.211:5060: (ICMP OK) SIP enabled
AVM FRITZ!Box Fon Series firmware: 14.03.(89|90) (Oct 28 2005)
Host 89.53.17.212:5060: (ICMP OK) SIP enabled
```

```
AVM FRITZ!Box Fon Series firmware: 14.03.(89|90) (Oct 28 2005)
Host 89.53.17.213:5060: (ICMP timeout) SIP enabled
Siemens SX541 (firmware 1.67)
Host 89.53.17.214:5060: (ICMP OK) SIP enabled
AVM FRITZ!Box Fon Series firmware: 14.03.(89|90) (Oct 28 2005)
Host 89.53.17.215:5060: (ICMP OK) SIP enabled
AVM FRITZ!Box Fon ata 11.03.45

8 hosts scanned, 6 ICMP reachable, 6 SIP enabled
$
```

Many of today's other open-source and commercial vulnerability scanners automate this banner-grabbing functionality along with port scanning, OS identification, service enumeration, and known vulnerability mapping. Just to name a few:

- Nessus (http://www.nessus.org)
- Retina (http://www.eeye.com)
- Saint (http://www.saintcorporation.com/saint/)



Target network (e.g. single target or a network 192.168.1.0-255 for class C network): 192.168.1.104

Connection Timeout (ms): 300

Currently Scanning : none

Port / Protocol :

Scan Progress: Completed

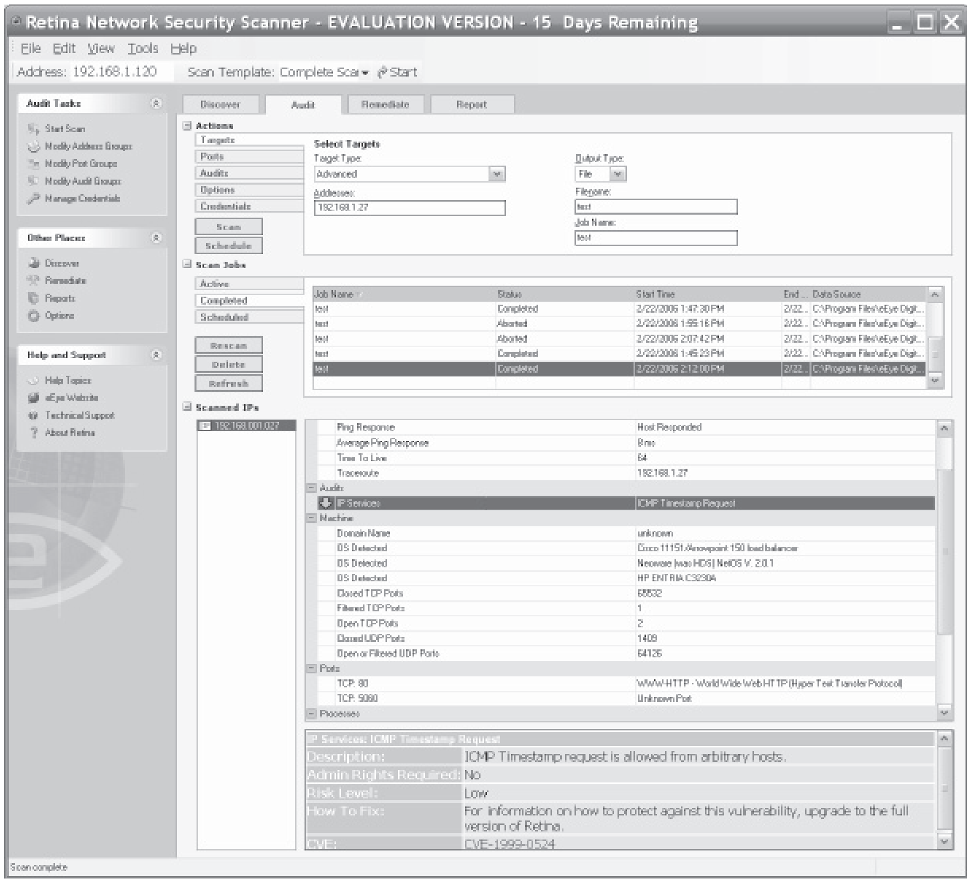
[✓] UDP
[✓] TCP
[ ] TLS

```
Hosts
192.168.1.104
    5060(UDP)
        Detected SIP UDP port : 5060
        Server: Sip EXpress router (0.9.6 (i386/linux))
    5060(TCP)
        Detected SIP TCP port : 5060
        Server: Sip EXpress router (0.9.6 (i386/linux))
```

Copyright 2004 (c) vopsecurity.org All Rights Reserved

Figure 3-1 SiVuS helps us find the same information we found manually with the click of a button.

Figures 3-2, 3-3, and 3-4 show a few of these tools being run on our target deployment's Polycom Soundpoint phone (192.168.1.27).



Address: 192.168.1.120 Scan Template: Complete Sca

Select Targets
Target Type: Advanced
Addresses: 192.168.1.27
Output Type: File
Filename: test
Job Name: test

Job Name	Status	Start Time	End ...	Data Source
test	Completed	2/22/2006 1:47:30 PM	2/22...	C:\Program Files\eEye Digit...
test	Aborted	2/22/2006 1:55:16 PM	2/22...	C:\Program Files\eEye Digit...
test	Aborted	2/22/2006 2:07:42 PM	2/22...	C:\Program Files\eEye Digit...
test	Completed	2/22/2006 1:45:23 PM	2/22...	C:\Program Files\eEye Digit...
test	Completed	2/22/2006 2:12:00 PM	2/22...	C:\Program Files\eEye Digit...

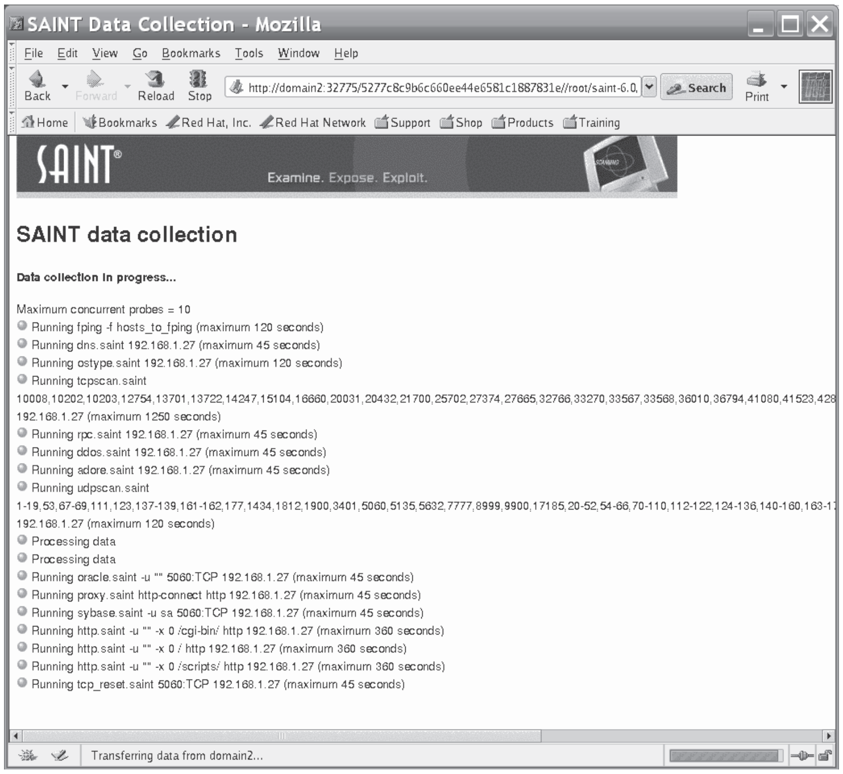
192.168.001.027

Ping Response	Host Responded
Average Ping Response	8 ms
Time To Live	64
Traceroute	192.168.1.27
Audits	
IP Services	ICMP Timestamp Request
Machine	
Domain Name	unknown
OS Detected	Cisco 11151/Arrowpoint 150 load balancer
OS Detected	Neoware (was HDS) NetOS V. 2.0.1
OS Detected	HP ENTRIA C3230A
Closed TCP Ports	65532
Filtered TCP Ports	1
Open TCP Ports	2
Closed UDP Ports	1409
Open or Filtered UDP Ports	64126
Ports	
TCP: 80	WWW-HTTP - World Wide Web HTTP (Hyper Text Transfer Protocol)
TCP: 5060	Unknown Port
Processes	

IP Services: ICMP Timestamp Request

Description:	ICMP Timestamp request is allowed from arbitrary hosts.
Admin Rights Required:	No
Risk Level:	Low
How To Fix:	For information on how to protect against this vulnerability, upgrade to the full version of Retina.
CVE:	CVE-1999-0524

Figure 3-2 The Retina scanner against the Polycom phone


SAINT Data Collection - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://domain2:32775/5277c8c9b6c660ee44e6581c1887831e//root/saint-6.0.
Search Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training
SAINT
Examine. Expose. Exploit.

SAINT data collection

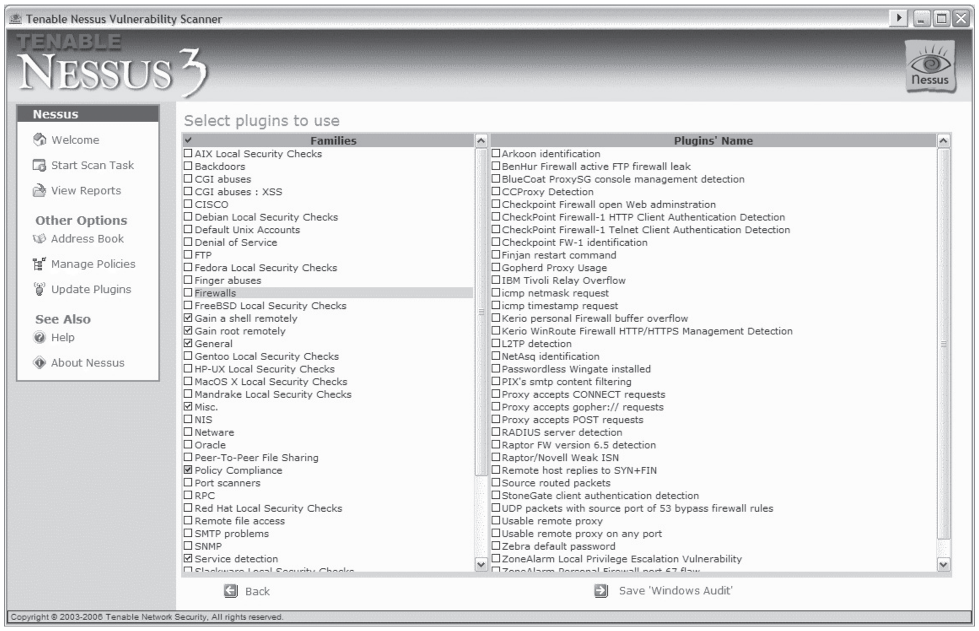
Data collection in progress...

```
Maximum concurrent probes = 10
● Running fping -f hosts_to_fping (maximum 120 seconds)
● Running dns.saint 192.168.1.27 (maximum 45 seconds)
● Running ostype.saint 192.168.1.27 (maximum 120 seconds)
● Running tcpscan.saint
10008,10202,10203,12754,13701,13722,14247,15104,16660,20031,20432,21700,25702,27374,27665,32766,33270,33567,33568,36010,36794,41080,41523,428
192.168.1.27 (maximum 1250 seconds)
● Running rpc.saint 192.168.1.27 (maximum 45 seconds)
● Running ddos.saint 192.168.1.27 (maximum 45 seconds)
● Running adore.saint 192.168.1.27 (maximum 45 seconds)
● Running udpscan.saint
1-19,53,67-69,111,123,137-139,161-162,177,1434,1812,1900,3401,5060,5135,5632,7777,8999,9900,17185,20-52,54-66,70-110,112-122,124-136,140-160,163-1
192.168.1.27 (maximum 120 seconds)
● Processing data
● Processing data
● Running oracle.saint -u "" 5060:TCP 192.168.1.27 (maximum 45 seconds)
● Running proxy.saint http-connect http 192.168.1.27 (maximum 45 seconds)
● Running sybase.saint -u sa 5060:TCP 192.168.1.27 (maximum 45 seconds)
● Running http.saint -u "" -x 0 /cgi-bin/ http 192.168.1.27 (maximum 360 seconds)
● Running http.saint -u "" -x 0 / http 192.168.1.27 (maximum 360 seconds)
● Running http.saint -u "" -x 0 /scripts/ http 192.168.1.27 (maximum 360 seconds)
● Running tcp_reset.saint 5060:TCP 192.168.1.27 (maximum 45 seconds)
```

Transferring data from domain2...

Figure 3-3 The Saint scanner in action against the Polycom phone

There is also a commercial VoIP-specific vulnerability scanning tool called VoIPaudit, developed by VoIPshield SYSTEMS (http://www.voipshield.com/). We did not get a chance to run this tool against our test deployment; however, the developers shared with us a screenshot of the tool for your viewing pleasure (see Figure 3-5).



Select plugins to use

Families	Plugins' Name
☐ AIX Local Security Checks	☐ Arkoon identification
☐ Backdoors	☐ BenHur Firewall active FTP firewall leak
☐ CGI abuses	☐ BlueCoat ProxySG console management detection
☐ CGI abuses : XSS	☐ CCProxy Detection
☐ CISCO	☐ Checkpoint Firewall open Web adminstration
☐ Debian Local Security Checks	☐ CheckPoint Firewall-1 HTTP Client Authentication Detection
☐ Default Unix Accounts	☐ CheckPoint Firewall-1 Telnet Client Authentication Detection
☐ Denial of Service	☐ Checkpoint FW-1 identification
☐ FTP	☐ Finjan restart command
☐ Fedora Local Security Checks	☐ Gopherd Proxy Usage
☐ Finger abuses	☐ IBM Tivoli Relay Overflow
☐ Firewalls	☐ icmp netmask request
☐ FreeBSD Local Security Checks	☐ icmp timestamp request
☑ Gain a shell remotely	☐ Kerio personal Firewall buffer overflow
☑ Gain root remotely	☐ Kerio WinRoute Firewall HTTP/HTTPS Management Detection
☑ General	☐ L2TP detection
☐ Gentoo Local Security Checks	☐ NetAsq identification
☐ HP-UX Local Security Checks	☐ Passwordless Wingate installed
☐ MacOS X Local Security Checks	☐ PIX's smtp content filtering
☐ Mandrake Local Security Checks	☐ Proxy accepts CONNECT requests
☑ Misc.	☐ Proxy accepts gopher:// requests
☐ NIS	☐ Proxy accepts POST requests
☐ Netware	☐ RADIUS server detection
☐ Oracle	☐ Raptor FW version 6.5 detection
☐ Peer-To-Peer File Sharing	☐ Raptor/Novell Weak ISN
☑ Policy Compliance	☐ Remote host replies to SYN+FIN
☐ Port scanners	☐ Source routed packets
☐ RPC	☐ StoneGate client authentication detection
☐ Red Hat Local Security Checks	☐ UDP packets with source port of 53 bypass firewall rules
☐ Remote file access	☐ Usable remote proxy
☐ SMTP problems	☐ Usable remote proxy on any port
☐ SNMP	☐ Zebra default password
☑ Service detection	☐ ZoneAlarm Local Privilege Escalation Vulnerability
☐ Slackware Local Security Checks	☐ ZoneAlarm Personal Firewall port 67 flaw

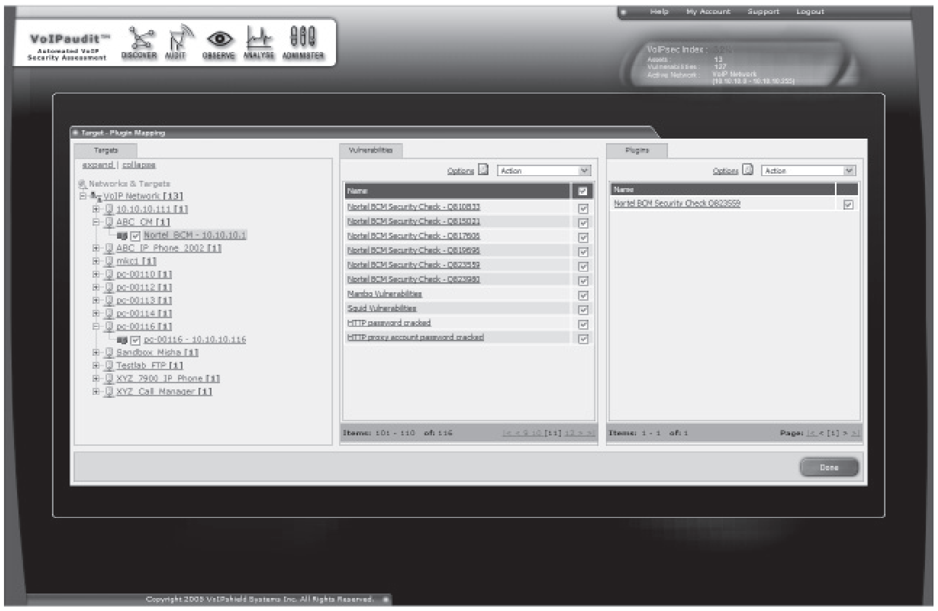
Figure 3-4 Selecting the specific Nessus scanning modules to run against the phone

Banner Grabbing Countermeasures

There's really not much you can do to prevent simple banner grabbing and service identification. Because of the open-source nature of some applications, such as Asterisk or SER or Apache, you can take an extreme approach and hack the source code to change the advertised banners. This is not a long-lasting resolution, however, and will rarely stop a determined hacker who has other techniques at her disposal.

The best solution is to constantly upgrade your applications and services with the latest updates available. Also, you should make sure to disable those services that are not needed in your VoIP environment; for instance, there's probably no good reason to leave telnet services running on your VoIP phone or PBX.

If and whenever possible, restrict access to the remaining administrative services to specific IP addresses. For example, if an administrative web interface to your Asterisk



VoIPsec Index :
Assets : 13
Vulnerabilities : 137
Active Network : VoIP Network (10.10.10.0 - 10.10.10.255)

expand | collapse

```
Networks & Targets
VoIP Network [13]
  10.10.10.111 [1]
  ABC_CM [1]
    Nortel_BCM - 10.10.10.1
  ABC_IP_Phone_2002 [1]
  mkc1 [1]
  pc-00110 [1]
  pc-00112 [1]
  pc-00113 [1]
  pc-00114 [1]
  pc-00116 [1]
    pc-00116 - 10.10.10.116
  Sandbox_Misha [1]
  Testlab_FTP [1]
  XYZ_7900_IP_Phone [1]
  XYZ_Call_Manager [1]
```

Name	✓
Nortel BCM Security Check - D810833	✓
Nortel BCM Security Check - D815021	✓
Nortel BCM Security Check - D817606	✓
Nortel BCM Security Check - D819696	✓
Nortel BCM Security Check - D823559	✓
Nortel BCM Security Check - D823980	✓
Mambo Vulnerabilities	✓
Squid Vulnerabilities	✓
HTTP password cracked	✓
HTTP proxy account password cracked	✓

Items: 101 - 110 of: 116 |< < 9 10 [11] 12 > >|

Name	
Nortel BCM Security Check D823559	✓

Items: 1 - 1 of: 1 Page: |< < [1] > >|

Copyright 2005 VoIPshield Systems Inc. All Rights Reserved.

Figure 3-5 Selecting the VoIP exploit plugins to launch

server listens on TCP port 8111, then apply the appropriate firewall or network switch rules to ensure outsiders cannot arbitrarily connect to that port with their browser.

SIP USER/EXTENSION ENUMERATION

In order to perform some of the attacks we'll be describing in later chapters, it's necessary for a hacker to know some valid usernames or extensions of SIP phones, registration, and/or proxy servers (see the section "SIP 101," at the beginning of the chapter). Short of calling every possible extension from 1–100000 with a traditional wardialer, or even by hand, there are much more effective ways to find this information.

Let's assume the hacker already has a working knowledge of the target company's SIP extension and/or username format based on our Google hacking exercises in Chapter 1. Many organizations might typically start assigning extensions at 100 or 200.

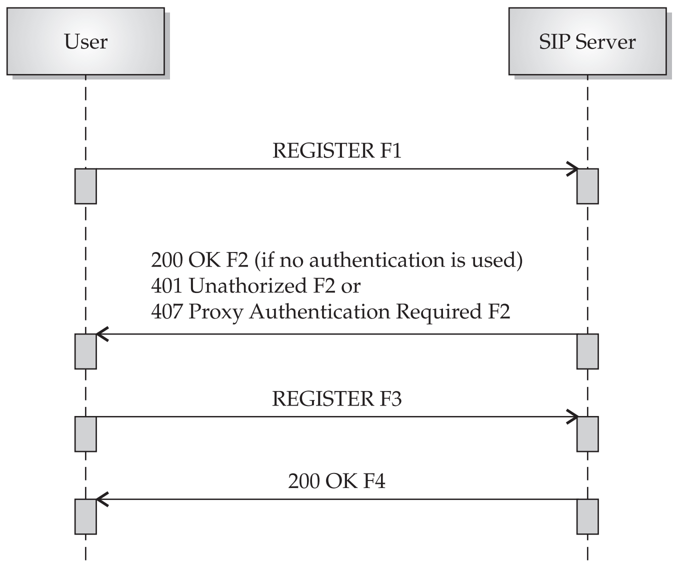
Even without this information, we can still make some pretty good guesses to begin our probing. The following enumeration methods rely on studying the error messages returned with these three SIP methods: REGISTER, OPTIONS, and INVITE. Not all servers and user agents will support all three methods; however, there's a decent chance one or two of them would be enough for our purposes.

For most of the SIP enumeration techniques we cover in the following sections, the SIP proxy or registrar (in other words the location server) is our main target because those are typically the easiest places to glean user registration and presence. The last section, however, also shows that by knowing a SIP phone's IP address, it is possible to interrogate it directly to find out its extension.

REGISTER Username Enumeration

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

A typical SIP REGISTER call flow from a phone to a registration server or proxy server looks like this (see the earlier section, "SIP 101"):


User
SIP Server
REGISTER F1
200 OK F2 (if no authentication is used)
401 Unathorized F2 or
407 Proxy Authentication Required F2
REGISTER F3
200 OK F4

Let's look at an actual example of this normal call flow in action with a valid REGISTER request to our SIP EXpress Router (192.168.1.104) as our Windows XTEN softphone is turned on:

```
Sent to 192.168.1.104:
REGISTER sip:192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.120:5060;rport;branch=z9hG4bK9AE42E04481647949E19
C9C281BD7CDC
From: 506 <sip:506@192.168.1.104>;tag=120975822
To: 506 <sip:506@192.168.1.104>
Contact: "506" <sip:506@192.168.1.120:5060>
Call-ID: 9F7F6FB9AFFA47278BE3CB571B3744D9@192.168.1.104
CSeq: 54512 REGISTER
Expires: 1800
Max-Forwards: 70
User-Agent: X-Lite release 1105x
Content-Length: 0

Received from 192.168.1.104:
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 192.168.1.120:5060;rport=5060;branch=z9hG4bK9AE42E044816479
49E19C9C281BD7CDC
From: 506 <sip:506@192.168.1.104>;tag=120975822
To: 506 <sip:506@192.168.1.104>;tag=b27e1a1d33761e85846fc98f5f3a7e58.bdc9
Call-ID: 9F7F6FB9AFFA47278BE3CB571B3744D9@192.168.1.104
CSeq: 54512 REGISTER
WWW-Authenticate: Digest realm="domain2", nonce="440bcbe24670d5d0448fd78ec4b
672a3c29de346"
Server: Sip EXpress router (0.9.6 (i386/linux))
Content-Length: 0
Warning: 392 192.168.1.104:5060 "Noisy feedback tells:  pid=29785
 req_src_ip=192.168.1.120 req_src_port=5060 in_uri=sip:192.168.1.104
out_uri=sip:192.168.1.104 via_cnt==1"
```

As you can see, we received a 401 SIP response as we expected. However, what happens if an invalid username is sent instead? Well, it depends on each specific SIP deployment's configuration. This time, with our own target deployment, let's try to send an invalid username in a REGISTER request to our SIP EXpress Router (192.168.1.104). *thisisthecanary*, the nonsensical username we chose, is almost surely not a valid extension or username in any organization.

```
Sent to 192.168.1.104
REGISTER sip:thisisthecanary@192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.120:2174;branch=el7mCh5QhC6WNg
From: test <sip:thisisthecanary@192.168.1.104>;tag=vkffYiKFjn
To: test <sip:thisisthecanary@192.168.1.104>
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 1 REGISTER
Contact: <sip:test@192.168.1.120:2174>
```

```
Max_forwards: 70
User Agent: SIPSCAN 1.0
Content-Type: application/sdp
Subject: SIPSCAN Probe
Expires: 7200
Content-Length: 0

Received from 192.168.1.104:
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 192.168.1.120:2174;branch=el7mCh5QhC6WNg
From: test <sip:thisisthecanary@192.168.1.104>;tag=vkffYiKFjn
To: test <sip:thisisthecanary@192.168.1.104>;tag=b27e1a1d33761e85846fc98f5f3
a7e58.b11e
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 1 REGISTER
WWW-Authenticate: Digest realm="domain2",
nonce="440bc944e0e0dc62d7185d035576505481d9dd34"
Server: Sip EXpress router (0.9.6 (i386/linux))
Content-Length: 0
Warning: 392 192.168.1.104:5060 "Noisy feedback tells:  pid=29782
req_src_ip=192.168.1.120 req_src_port=2174
in_uri=sip:thisisthecanary@192.168.1.104
out_uri=sip:thisisthecanary@192.168.1.104 via_cnt==1"
```

As you can see, we received exactly the same response as we would with a legitimate username, a 401 SIP response. However, let's see what happens if we send the exact same requests to our Asterisk server at 192.168.1.103. First, here's an actual SIP trace of our Snom 320 phone turning on and registering:

```
Sent to 192.168.1.103
REGISTER sip:192.168.1.103 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.21:2051;branch=z9hG4bK-v7brim4vvk49;rport
From: "Snom 320" <sip:201@192.168.1.103>;tag=e35li4iydd
To: "Snom 320" <sip:201@192.168.1.103>
Call-ID: 3c2670092710-g9u6jfnehewi@snom320
CSeq: 1 REGISTER
Max-Forwards: 70
Contact: <sip:201@192.168.1.21:2051;line=ylcbbss9>;q=1.0;+sip.instance=
"<urn:uuid:bf9b2fe3-b95c-4cfd-96ad-5b52bf1d0c2a>"
;audio;mobility="fixed";duplex="full";description="snom320";actor=
"principal";events="dialog";methods=
"INVITE,ACK,CANCEL,BYE,REFER,OPTIONS,NOTIFY,SUBSCRIBE,PRACK,MESSAGE,INFO"
User-Agent: snom320/4.1
Supported: gruu
Allow-Events: dialog
```

```
X-Real-IP: 192.168.1.21
WWW-Contact: <http://192.168.1.21:80>
WWW-Contact: <https://192.168.1.21:443>
Expires: 3600
Content-Length: 0

Received from: 192.168.1.103
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 192.168.1.21:2051;branch=z9hG4bK-v7brim4vvk49
From: "Snom 320" <sip:201@192.168.1.103>;tag=e35li4iydd
To: "Snom 320" <sip:201@192.168.1.103>;tag=as356abebc
Call-ID: 3c2670092bf2-g9u6jfnehewi@snom320
CSeq: 1 REGISTER
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
Contact: <sip:201@192.168.1.103>
WWW-Authenticate: Digest realm="asterisk", nonce="38e8e429"
Content-Length: 0
```

Okay, a 401 response is what we expected. Now let's send the same invalid username *thisisthecanary* to see what the Asterisk server responds with

```
REGISTER sip:thisisthecanary@192.168.1.103 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.120:2219;branch=el7mCh5QhC6WNg
From: test <sip:thisisthecanary@192.168.1.103>;tag=vkffYiKFjn
To: test <sip:thisisthecanary@192.168.1.103>
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 1 REGISTER
Contact: <sip:test@192.168.1.120:2219>
Max_forwards: 70
User Agent: SIPSCAN 1.0
Content-Type: application/sdp
Subject: SIPSCAN Probe
Expires: 7200
Content-Length: 0

Received from 192.168.1.103:
SIP/2.0 403 Forbidden
Via: SIP/2.0/UDP 192.168.1.120:2219;branch=el7mCh5QhC6WNg
From: test <sip:thisisthecanary@192.168.1.103>;tag=vkffYiKFjn
To: test <sip:thisisthecanary@192.168.1.103>;tag=as44107711
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 1 REGISTER
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
```

```
Contact: <sip:thisisthecanary@192.168.1.103>
Content-Length: 0
```

Aha! Notice that with the invalid username, we received a 403 SIP response (Forbidden) instead of the 401 response we would get with a REGISTER request using a valid username.

By using this differentiated response to our advantage, we have the means to enumerate all valid extensions on our Asterisk server (192.168.1.103), and by sending REGISTER requests to as many extensions or usernames as possible, we should be able to eliminate invalid extensions with those we try that receive a nonstandard SIP response (for example, 403 Forbidden) in return. Perhaps we can find another enumeration technique to use on the SIP EXpress Router since both the valid and invalid username attempts result in the same response.

INVITE Username Enumeration

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

INVITE scanning is the noisiest and least stealthy method for SIP username enumeration because it involves actually ringing the target's phones. Even after normal business hours, missed calls are usually logged on the phones and on the target SIP proxy, so there's a fair amount of traceback evidence left behind. However, if you don't mind the audit trail, it can often be another useful directory discovery method. The background of a typical call initiation is covered in the "SIP 101" section at the beginning of the chapter.

First, let's see what happens when we try to call a valid user who has already registered with our SIP EXpress Router server (192.168.1.104). We used SiVuS to generate these messages:

```
Sent to 192.168.1.104:
INVITE sip:506@192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.120:2590;rport;branch=z9hG4bK44FE55FBBCC449A9A4BE
B71869664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:506@192.168.1.104>
Contact: <sip:test@192.168.1.120:2590>
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 329171 INVITE
Max-Forwards: 70
Content-Type: application/sdp
User-Agent: X-Lite release 1105x
```

```
Content-Length: 305

v=0
o=test 1339154 8901572 IN IP4 192.168.1.120
s=X-Lite
c=IN IP4 192.168.1.120
t=0 0
m=audio 8000 RTP/AVP 0 8 3 98 97 101
a=rtpmap:0 pcmu/8000
a=rtpmap:8 pcma/8000
a=rtpmap:3 gsm/8000
a=rtpmap:98 iLBC/8000
a=rtpmap:97 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv

Received from 192.168.1.104:
SIP/2.0 100 trying -- your call is important to us
Via: SIP/2.0/UDP
192.168.1.120:2590;rport=2590;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:506@192.168.1.104>
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 329171 INVITE
Server: Sip EXpress router (0.9.6 (i386/linux))
Content-Length: 0
Warning: 392 192.168.1.104:5060 "Noisy feedback tells:  pid=29788
req_src_ip=192.168.1.120 req_src_port=2590 in_uri=sip:506@192.168.1.104
out_uri=sip:506@192.168.1.56:5060 via_cnt==1"

Received from 192.168.1.104:
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP
192.168.1.120:2590;rport=2590;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:506@192.168.1.104>;tag=3557948964
Contact: <sip:506@192.168.1.56:5060>
Record-Route: <sip:192.168.1.104;ftag=325602560;lr=on>
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 329171 INVITE
Server: X-Lite release 1105x
Content-Length: 0
```

Because we never sent a CANCEL request to tear down the call, the target phone is left ringing. In later chapters, we'll look at certain denial of service attacks, such as *INVITE floods,* whereby all free lines on the target's phone are exhausted with bogus incoming calls. Sending a follow-up CANCEL request ensures that not every single phone in the office is ringing when people start coming into work in the morning, giving away that someone tried to SIP-scan the network. Now let's see what happens on our SER server when we try to call our nonexistent user, *thisisthecanary*:

```
Sent to 192.168.1.104:
INVITE sip:thisisthecanary@192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP
192.168.1.120:2549;rport;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:thisisthecanary@192.168.1.104>
Contact: <sip:test@192.168.1.120:2549>
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 946396 INVITE
Max-Forwards: 70
Content-Type: application/sdp
User-Agent: X-Lite release 1105x
Content-Length: 305

v=0
o=test 6585767 8309317 IN IP4 192.168.1.120
s=X-Lite
c=IN IP4 192.168.1.120
t=0 0
m=audio 8000 RTP/AVP 0 8 3 98 97 101
a=rtpmap:0 pcmu/8000
a=rtpmap:8 pcma/8000
a=rtpmap:3 gsm/8000
a=rtpmap:98 iLBC/8000
a=rtpmap:97 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv

Received from 192.168.1.104:
SIP/2.0 404 Not Found
Via: SIP/2.0/UDP
192.168.1.120:2549;rport=2549;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:thisisthecanary@192.168.1.104>;tag=b27e1a1d33761e85846fc98f5f3a7e58
.611c
```

```
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 946396 INVITE
Server: Sip EXpress router (0.9.6 (i386/linux))
Content-Length: 0
Warning: 392 192.168.1.104:5060 "Noisy feedback tells:  pid=29775
req_src_ip=192.168.1.120 req_src_port=2549
in_uri=sip:thisisthecanary@192.168.1.104
out_uri=sip:thisisthecanary@192.168.1.104 via_cnt==1"
```

Notice that we get a 404 Not Found response, thereby granting us a useful method for enumerating valid users on the SIP EXpress Router service at 192.168.1.104. Now let's see what happens with our Asterisk server at 192.168.1.103 when we try both a valid and invalid INVITE request:

Sent to 192.168.1.103:

```
INVITE sip:205@192.168.1.103 SIP/2.0
Via: SIP/2.0/UDP
192.168.1.120:2617;rport;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:205@192.168.1.103>
Contact: <sip:test@192.168.1.120:2617>
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 111530 INVITE
Max-Forwards: 70
Content-Type: application/sdp
User-Agent: X-Lite release 1105x
Content-Length: 305

v=0
o=test 1669075 4839162 IN IP4 192.168.1.120
s=X-Lite
c=IN IP4 192.168.1.120
t=0 0
m=audio 8000 RTP/AVP 0 8 3 98 97 101
a=rtpmap:0 pcmu/8000
a=rtpmap:8 pcma/8000
a=rtpmap:3 gsm/8000
a=rtpmap:98 iLBC/8000
a=rtpmap:97 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv
```

Received from 192.168.1.103:

```
SIP/2.0 100 Trying
```

```
Via: SIP/2.0/UDP
192.168.1.120:2617;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:205@192.168.1.103>
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 111530 INVITE
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
Contact: <sip:205@192.168.1.103>
Content-Length: 0

Received from 192.168.1.103:
SIP/2.0 503 Service Unavailable
Via: SIP/2.0/UDP 192.168.1.120:2617;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869
664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:205@192.168.1.103>;tag=as51cce52a
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 111530 INVITE
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
Contact: <sip:205@192.168.1.103>
Content-Length: 0
```

Now we try to send an INVITE to our invalid user, *thisisthecanary,* to see if the responses differ:

```
Sent to 192.168.1.103:
INVITE sip:thisisthecanary@192.168.1.103 SIP/2.0
Via: SIP/2.0/UDP
192.168.1.120:2594;rport;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:thisisthecanary@192.168.1.103>
Contact: <sip:test@192.168.1.120:2594>
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 853716 INVITE
Max-Forwards: 70
Content-Type: application/sdp
User-Agent: X-Lite release 1105x
Content-Length: 305

v=0
o=test 7098201 3974048 IN IP4 192.168.1.120
s=X-Lite
c=IN IP4 192.168.1.120
```

```
t=0 0
m=audio 8000 RTP/AVP 0 8 3 98 97 101
a=rtpmap:0 pcmu/8000
a=rtpmap:8 pcma/8000
a=rtpmap:3 gsm/8000
a=rtpmap:98 iLBC/8000
a=rtpmap:97 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv

Received from 192.168.1.103:
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 192.168.1.120:2594;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869
664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:thisisthecanary@192.168.1.103>
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 853716 INVITE
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
Contact: <sip:thisisthecanary@192.168.1.103>
Content-Length: 0

Received from 192.168.1.103:
SIP/2.0 503 Service Unavailable
Via: SIP/2.0/UDP 192.168.1.120:2594;branch=z9hG4bK44FE55FBBCC449A9A4BEB71869
664AEC
From: test <sip:test@192.168.1.120>;tag=325602560
To: <sip:thisisthecanary@192.168.1.103>;tag=as7d316dda
Call-ID: 1D49F1E5-25D8-4B90-B16D-0DB57899DDB2@192.168.1.120
CSeq: 853716 INVITE
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
Contact: <sip:thisisthecanary@192.168.1.103>
Content-Length: 0
```

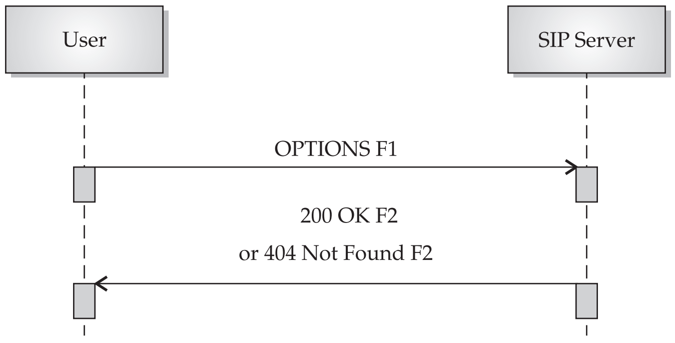
Unfortunately, both SIP response codes were the same, which means, in this case, we can't use INVITE scanning for our Asterisk deployment to differentiate between invalid and valid extensions. However, at least this technique worked for the SIP EXpress Router deployment!

Of course, you can also send INVITE requests directly to phones if you already know their IP addresses. This way you can bypass the proxy altogether if you're concerned about logging.

OPTIONS Username Enumeration

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

The OPTIONS method is the most stealthy and effective method for enumerating SIP users. The OPTIONS method is supported (as commanded by RFC 3261) by all SIP services and user agents and is used for advertising supported message capabilities and, in some cases, legitimate users. The simple flow of an OPTIONS request and response looks like this:



So let’s take the same methodical approach we’ve taken in the previous two sections and send two different types of OPTIONS requests to our SER server (192.168.1.104). First, we try the valid username 506:

```
Sent to 192.168.1.104:
OPTIONS sip:506@192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.120:2700;branch=el7mCh5QhC6WNg
From: test <sip:test@192.168.1.120>;tag=vkffYiKFjn
To: test <sip:506@192.168.1.104>
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 42 OPTIONS
Contact: <sip:test@192.168.1.120:2700>
Max_forwards: 70
User Agent: SIPSCAN 1.0
Subject: SIPSCAN 1.0 Probe
Content-Length: 0

Received from 192.168.1.104:
SIP/2.0 200 Ok
```

```
Via: SIP/2.0/UDP 192.168.1.120:2700;branch=el7mCh5QhC6WNg
From: test <sip:test@192.168.1.120>;tag=vkffYiKFjn
To: test <sip:506@192.168.1.104>;tag=1280386989
Contact: <sip:506@192.168.1.56:5060>
Call-ID: AXy1SAVzvwd9@192.168.1.120
Allow: INVITE,ACK,BYE,CANCEL,OPTIONS,NOTIFY
CSeq: 42 OPTIONS
Server: X-Lite release 1105x
Content-Length: 0
```

Notice that not only did we determine that this is a valid user, but in looking at the Server field, we can also deduce what type of phone is associated with that extension—in this case, an XTEN X-Lite softphone. This information might come in handy later. Now let's see what happens when we send an OPTIONS request with the invalid username *thisisthecanary*:

```
Sent to 192.168.1.104:
OPTIONS sip:thisisthecanary@192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.120:2659;branch=el7mCh5QhC6WNg
From: test <sip:test@192.168.1.120>;tag=vkffYiKFjn
To: test <sip:thisisthecanary@192.168.1.104>
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 1 OPTIONS
Contact: <sip:test@192.168.1.120:2659>
Max_forwards: 70
User Agent: SIPSCAN 1.0
Subject: SIPSCAN 1.0 Probe
Content-Length: 0

Received from 192.168.1.104:
SIP/2.0 404 Not Found
Via: SIP/2.0/UDP 192.168.1.120:2659;branch=el7mCh5QhC6WNg
From: test <sip:test@192.168.1.120>;tag=vkffYiKFjn
To: test <sip:thisisthecanary@192.168.1.104>;tag=b27e1a1d33761e85846fc98f5f3
a7e58.6e4b
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 1 OPTIONS
Server: Sip EXpress router (0.9.6 (i386/linux))
Content-Length: 0
Warning: 392 192.168.1.104:5060 "Noisy feedback tells:  pid=29782
req_src_ip=192.168.1.120 req_src_port=2659
in_uri=sip:thisisthecanary@192.168.1.104
out_uri=sip:thisisthecanary@192.168.1.104 via_cnt==1"
```

As you can see, the OPTIONS response (404 Not Found) conveniently lets us know that this user doesn't exist, or is not currently logged in. Now let's try the same technique against our Asterisk server (192.168.1.103). We'll try user 201 in our valid probe:

```
Sent to 192.168.1.103:
OPTIONS sip:201@192.168.1.103 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.120:2723;branch=el7mCh5QhC6WNg
From: test <sip:test@192.168.1.120>;tag=vkffYiKFjn
To: test <sip:201@192.168.1.103>
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 16 OPTIONS
Contact: <sip:test@192.168.1.120:2723>
Max_forwards: 70
User Agent: SIPSCAN 1.0
Subject: SIPSCAN 1.0 Probe
Content-Length: 0

Received from 192.168.1.103:
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.120:2723;branch=el7mCh5QhC6WNg
From: test <sip:test@192.168.1.120>;tag=vkffYiKFjn
To: test <sip:201@192.168.1.103>;tag=as3ad8a754
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 16 OPTIONS
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
Contact: <sip:192.168.1.103>
Accept: application/sdp
Content-Length: 0
```

So far so good. Let's now hope that we get a different response when we try an invalid user. Again, we use *thisisthecanary* to test the standard error response from Asterisk:

```
Sent to 192.168.1.103:
OPTIONS sip:thisisthecanary@192.168.1.103 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.120:2708;branch=el7mCh5QhC6WNg
From: test <sip:test@192.168.1.120>;tag=vkffYiKFjn
To: test <sip:thisisthecanary@192.168.1.103>
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 1 OPTIONS
Contact: <sip:test@192.168.1.120:2708>
Max_forwards: 70
User Agent: SIPSCAN 1.0
Subject: SIPSCAN 1.0 Probe
Content-Length: 0
```

```
Received from 192.168.1.103:
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.120:2708;branch=el7mCh5QhC6WNg
From: test <sip:test@192.168.1.120>;tag=vkffYiKFjn
To: test <sip:thisisthecanary@192.168.1.103>;tag=as1faacd0f
Call-ID: AXy1SAVzvwd9@192.168.1.120
CSeq: 1 OPTIONS
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
Contact: <sip:192.168.1.103>
Accept: application/sdp
Content-Length: 0
```

Unfortunately, we get the exact same response with an invalid username as well (200 OK). It looks like we won't be able to use OPTIONS scanning for our Asterisk target. At least the REGISTER scanning detailed previously in "REGISTER Username Enumeration" worked for us!

Automated OPTIONS Scanning with sipsak

Popularity:	4
Simplicity:	7
Impact:	4
Risk Rating:	**5**

Boy, would it be nice to find a tool to automate as many of the previously discussed scanning techniques as possible! For OPTIONS scanning, there's a command-line tool called sipsak that can help. sipsak (http://sipsak.org) is developed by the authors of SIP EXpress Router, and it is helpful in stress testing and diagnosing SIP service issues. While it was not designed specifically for SIP username enumeration, it can be easily tailored for such a task. Unfortunately, sipsak only supports OPTIONS requests by default, and sending other SIP methods requires some tweaking.

Let's reconstruct the OPTIONS scanning example against our SIP EXpress Router proxy server with sipsak. First, we try to probe the server with the valid extension 506:

```
[root@attacker]# sipsak -vv -s sip:506@192.168.1.104
New message with Via-Line:
OPTIONS sip:506@192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP asterisk1.local:32874;rport
From: sip:sipsak@asterisk1.local:32874;tag=702e3179
To: sip:506@192.168.1.104
Call-ID: 1882075513@asterisk1.local
CSeq: 1 OPTIONS
Contact: sip:sipsak@asterisk1.local:32874
Content-Length: 0
```

```
Max-Forwards: 70
User-Agent: sipsak 0.8.11
Accept: text/plain


** request **
OPTIONS sip:506@192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP asterisk1.local:32874;rport
From: sip:sipsak@asterisk1.local:32874;tag=702e3179
To: sip:506@192.168.1.104
Call-ID: 1882075513@asterisk1.local
CSeq: 1 OPTIONS
Contact: sip:sipsak@asterisk1.local:32874
Content-Length: 0
Max-Forwards: 70
User-Agent: sipsak 0.8.11
Accept: text/plain



message received:
SIP/2.0 200 Ok
Via: SIP/2.0/UDP asterisk1.local:32874;received=192.168.1.103;rport=32874
From: sip:sipsak@asterisk1.local:32874;tag=702e3179
To: <sip:506@192.168.1.104>;tag=644497335
Contact: <sip:506@192.168.1.56:5060>
Call-ID: 1882075513@asterisk1.local
Allow: INVITE,ACK,BYE,CANCEL,OPTIONS,NOTIFY
CSeq: 1 OPTIONS
Server: X-Lite release 1105x
Content-Length: 0



** reply received after 5.479 ms **
   SIP/2.0 200 Ok
   final received
```

Now we try to probe the server using our invalid extension, *thisisthecanary*:

```
[root@asterisk1 sipsak-0.9.6]# sipsak -vv -s sip:thisisthecana-
ry@192.168.1.104
New message with Via-Line:
OPTIONS sip:thisisthecanary@192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP asterisk1.local:32876;rport
From: sip:sipsak@asterisk1.local:32876;tag=1aeccc21
```

```
To: sip:thisisthecanary@192.168.1.104
Call-ID: 451726369@asterisk1.local
CSeq: 1 OPTIONS
Contact: sip:sipsak@asterisk1.local:32876
Content-Length: 0
Max-Forwards: 70
User-Agent: sipsak 0.8.11
Accept: text/plain


** request **
OPTIONS sip:thisisthecanary@192.168.1.104 SIP/2.0
Via: SIP/2.0/UDP asterisk1.local:32876;rport
From: sip:sipsak@asterisk1.local:32876;tag=1aeccc21
To: sip:thisisthecanary@192.168.1.104
Call-ID: 451726369@asterisk1.local
CSeq: 1 OPTIONS
Contact: sip:sipsak@asterisk1.local:32876
Content-Length: 0
Max-Forwards: 70
User-Agent: sipsak 0.8.11
Accept: text/plain



message received:
SIP/2.0 404 Not Found
Via: SIP/2.0/UDP asterisk1.local:32876;rport=32876;received=192.168.1.103
From: sip:sipsak@asterisk1.local:32876;tag=1aeccc21
To: sip:thisisthecanary@192.168.1.104;tag=b27e1a1d33761e85846fc98f5f3a7e58.9
543
Call-ID: 451726369@asterisk1.local
CSeq: 1 OPTIONS
Server: Sip EXpress router (0.9.6 (i386/linux))
Content-Length: 0
Warning: 392 192.168.1.104:5060 "Noisy feedback tells:  pid=29782
req_src_ip=192.168.1.103 req_src_port=32876
in_uri=sip:thisisacanary@192.168.1.104
out_uri=sip:thisisacanary@192.168.1.104 via_cnt==1"



** reply received after 0.562 ms **
   SIP/2.0 404 Not Found
   final received
```

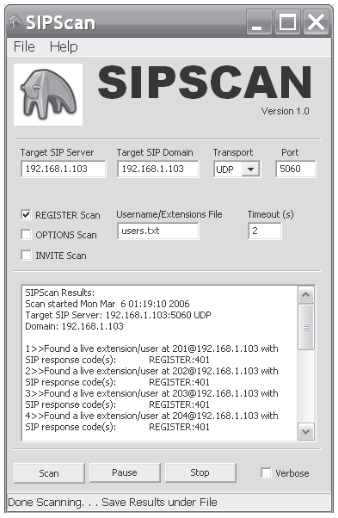
If we wanted to, we could easily script sipsak to iterate through a list of usernames and then parse the output afterward to find the live extensions. For instance, here's a (very) simple example:

```
#!/usr/local/bin/perl

@usernames = (500,501,505,503,504,505,506,507,508,509,510,511,512,513,514,515);

foreach $key (@usernames) {
      system("sipsak -vv -s sip:$key\@192.168.1.104 >\> sipsak_output.txt")
}
```

Automated REGISTER, INVITE, and OPTIONS Scanning with SIPSCAN Against SIP Servers

Popularity:	*4*
Simplicity:	*9*
Impact:	*4*
Risk Rating:	***5***

Because sipsak required some heavy lifting to get it to scan with INVITE and REGISTER requests as well, we decided to write our own graphical SIP username/extension enumerator called SIPSCAN (available from http://www.hackingvoip.com/ and shown in Figure 3-6).



Target SIP Server: 192.168.1.103
Target SIP Domain: 192.168.1.103
Transport: UDP
Port: 5060

[✓] REGISTER Scan
[ ] OPTIONS Scan
[ ] INVITE Scan
Username/Extensions File: users.txt
Timeout (s): 2

```
SIPScan Results:
Scan started Mon Mar  6 01:19:10 2006
Target SIP Server: 192.168.1.103:5060 UDP
Domain: 192.168.1.103

1>>Found a live extension/user at 201@192.168.1.103 with
SIP response code(s):          REGISTER:401
2>>Found a live extension/user at 202@192.168.1.103 with
SIP response code(s):          REGISTER:401
3>>Found a live extension/user at 203@192.168.1.103 with
SIP response code(s):          REGISTER:401
4>>Found a live extension/user at 204@192.168.1.103 with
SIP response code(s):          REGISTER:401
```

[ ] Verbose

Done Scanning. . . Save Results under File

Figure 3-6 SIPSCAN using REGISTER requests against the Asterisk deployment at 192.168.1.103

SIPSCAN combines the aspects of all three of the previous scanning methods and only returns the live SIP extensions/users that it finds. By default, SIPSCAN comes with a list of usernames (`users.txt`) to brute force. You should, of course, tailor your own list to suit the needs of the target environment you are scanning. For this example, our `users.txt` file includes the following:

```
thisisthecanary
test
echo
admin
dave
101
102
103
104
104
105
106
107
108
110
201
202
203
204
204
205
206
207
208
210
401
402
403
404
404
405
406
407
408
410
501
502
503
```

```
504
504
505
506
507
508
510
```

You'll notice our now infamous *thisisthecanary* username at the top of the list. You *must* keep this username as the first entry because SIPSCAN uses it to baseline an invalid SIP response for each of the scanning techniques selected. As we've seen previously, without a "known bad" username, we won't know whether we can accurately differentiate valid extensions from invalid ones. Let's try a REGISTER scanning attempt using SIPSCAN against our Asterisk server:

```
SIPSCAN Results:
Scan started Mon Mar  6 01:19:10 2006
Target SIP Server: 192.168.1.103:5060 UDP
Domain: 192.168.1.103

1>\>Found a live extension/user at 201@192.168.1.103
with SIP response code(s):    REGISTER:401
2>\>Found a live extension/user at 202@192.168.1.103
with SIP response code(s):    REGISTER:401
3>\>Found a live extension/user at 203@192.168.1.103
with SIP response code(s):    REGISTER:401
4>\>Found a live extension/user at 204@192.168.1.103
with SIP response code(s):    REGISTER:401
5>\>Found a live extension/user at 204@192.168.1.103
with SIP response code(s):    REGISTER:401
6>\>Found a live extension/user at 205@192.168.1.103
with SIP response code(s):    REGISTER:401
7>\>Found a live extension/user at 207@192.168.1.103
with SIP response code(s):    REGISTER:401
```

As we could have predicted from the previous manual examples against our Asterisk server, it looks as if the RESISTER scan was successful in ferreting out the valid extensions. Let's see what happens when we try selecting just the OPTIONS scan:

```
SIPSCAN Results:
Scan started Mon Mar  6 01:28:16 2006
Target SIP Server: 192.168.1.103:5060 UDP
Domain: 192.168.1.103
```

As expected, we got no results because all of the users we tried to enumerate returned a 200 OK SIP response, not really allowing us to differentiate between the valid ones and nonexistent ones. If we select more than one scanning technique (REGISTER and OPTION), SIPSCAN will combine the results from all methods and show you the merged output:

```
SIPSCAN Results:
Scan started Mon Mar  6 01:35:10 2006
Target SIP Server: 192.168.1.103:5060 UDP
Domain: 192.168.1.103
1>\>Found a live extension/user at 201@192.168.1.103
with SIP response code(s):    REGISTER:401  OPTIONS: 200
2>\>Found a live extension/user at 202@192.168.1.103
with SIP response code(s):    REGISTER:401  OPTIONS: 200
3>\>Found a live extension/user at 203@192.168.1.103
with SIP response code(s):    REGISTER:401  OPTIONS: 200
4>\>Found a live extension/user at 204@192.168.1.103
with SIP response code(s):    REGISTER:401  OPTIONS: 200
5>\>Found a live extension/user at 204@192.168.1.103
with SIP response code(s):    REGISTER:401  OPTIONS: 200
6>\>Found a live extension/user at 205@192.168.1.103
with SIP response code(s):    REGISTER:401  OPTIONS: 200
7>\>Found a live extension/user at 207@192.168.1.103
with SIP response code(s):    REGISTER:401  OPTIONS: 200
```

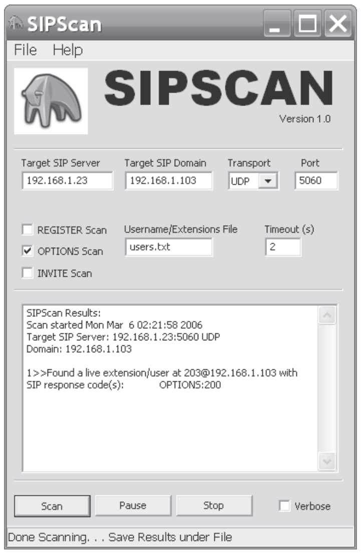
Obviously selecting all three scans provides the best possible details.

Automated OPTIONS Scanning Using SIPSCAN Against SIP Phones

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

Instead of focusing our efforts at SIP servers, SIPSCAN can also be used against some SIP phones as well. Through OPTIONS scanning, we can determine the exact extension(s) that the phone uses to log in to the SIP proxy or registrar. For instance, if we point SIPSCAN at our Cisco 7912 phone at 192.168.1.23 in our target deployment with the options selected as shown in Figure 3-7, we get the following results:

```
SIPSCAN Results:
Scan started Mon Mar  6 02:21:58 2006
Target SIP Server: 192.168.1.23:5060 UDP
Domain: 192.168.1.103

1>\>Found a live extension/user at 203@192.168.1.103
with SIP response code(s):     OPTIONS:200
```



Target SIP Server: 192.168.1.23
Target SIP Domain: 192.168.1.103
Transport: UDP
Port: 5060

[ ] REGISTER Scan
[✓] OPTIONS Scan
[ ] INVITE Scan

Username/Extensions File: users.txt
Timeout (s): 2

```
SIPScan Results:
Scan started Mon Mar  6 02:21:58 2006
Target SIP Server: 192.168.1.23:5060 UDP
Domain: 192.168.1.103

1>>Found a live extension/user at 203@192.168.1.103 with
SIP response code(s):          OPTIONS:200
```

[ ] Verbose

Done Scanning. . . Save Results under File

Figure 3-7 Using SIPSCAN against our Cisco IP Phone 7912 to find its extension

Looking at our network diagram in Figure 2-1 in Chapter 2, we can confirm that 203 is, in fact, the extension assigned to that phone.

Let's try it on another phone just for fun. How about our Polycom IP301 SoundPoint at 192.168.1.27?

```
SIPSCAN Results:
Scan started Mon Mar  6 02:24:58 2006
Target SIP Server: 192.168.1.27:5060 UDP
Domain: 192.168.1.103

1>\>Found a live extension/user at 207@192.168.1.103
with SIP response code(s):     OPTIONS:200
```

That one worked too; we got a 200 OK response telling us that the 207 extension is valid.

TIP Similar to the varying levels of success we had scanning different servers, OPTIONS scanning doesn't always work against all SIP phones because it depends on how the specific phone vendor implemented the SIP stack.

Knowing the exact extension assigned to a phone gives an attacker vital information needed to perform some of the more advanced attacks described in later chapters. Knowing the CEO's phone extension might make it easier for a hacker to brute force voicemail credentials, spoof SIP credentials and calls, and kick his phone off the network.

SIP User/Extension Enumeration Countermeasures

Preventing automated enumeration of SIP extensions and usernames is difficult. Much like preventing normal port scanning, it is hard to shield services such as SIP that by its very nature needs to be exposed to a certain extent. Enabling authentication of users and usage (INVITE, REGISTER, and so on) on your SIP proxy server will prevent some types of anonymous directory scanning. However, as you saw from the previous examples, that won't always help.

A recommendation you will hear from us over and over again is to segment portions of the VoIP network on a VLAN separate from the traditional data network. This will help mitigate a variety of threats, including enumeration. This is not always possible with some SIP architectures that include softphones residing on the user's PC.

There are also some VoIP intrusion prevention devices that can detect a rapid succession of INVITE, OPTIONS, or REGISTER probes against a SIP proxy target and block the source address from further scanning. These devices are available from multiple vendors, including SecureLogix (www.securelogix.com), Sipera (www.sipera.com), and BorderWare (www.borderware.com), and can be deployed at various locations in the network, such as "in front" of the SIP proxy/registrar and/or on a connection to the public network for SIP trunks.

ENUMERATION OF OTHER VOIP SUPPORT SERVICES

Obviously, VoIP platforms rely on a plethora of common network services such as DNS, Microsoft Active Directory, LDAP, RADIUS, and so on. Enumerating most of these common services from a typical security auditing perspective is already covered in great detail in the main *Hacking Exposed, Fifth Edition* book, which we highly recommend reading. Rather than reiterate a lot of general security enumeration techniques already covered elsewhere, we've tried to limit the scope of this section to enumerating those main support services that most VoIP devices rely on.

Enumerating TFTP Servers

Popularity:	5
Simplicity:	9
Impact:	9
Risk Rating:	**7**

The majority of the phones that we set up in our test environment rely upon a Trivial File Transfer Protocol (TFTP) server for downloading their configuration settings. TFTP is dangerously insecure in that it requires no authentication to upload or fetch a file. This means that in the majority of enterprise VoIP installations, a TFTP server is typically exposed to the network so phones can download their initial settings each time they power up.

When booting up each time, many phones first try to download a configuration file. Sometimes this configuration file is a derivative of the phone's MAC address. For instance, our Avaya 4620 phone tries to download the files `46xxsettings.txt` and `46xxupgrade.scr` each time it is powered on. Our Cisco 7912 IP phone tries to download the files `SIPDefault.cnf` and `SEP001562EA69E8.cnf` (001562EA69E8 is its MAC address) each time from the same TFTP server. One of the easiest ways for a hacker to compromise a VoIP network is to focus first on the TFTP servers.

The first step to enumerating the files on a TFTP server is locating the server within the network. As you saw in the Googling exercises in Chapter 1, this might be as easy as reading the TFTP server IP address from the web-based configuration readout. As a refresher, let's scan our target deployment again simply looking for listening services on UDP port 69 (tftp):

```
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2006-03-07 01:56 CST
Interesting ports on 192.168.1.21:
PORT   STATE  SERVICE
69/udp closed tftp
MAC Address: 00:04:13:24:23:8D (Snom Technology AG)

Interesting ports on 192.168.1.22:
PORT   STATE         SERVICE
69/udp open|filtered tftp
MAC Address: 00:0F:34:11:80:45 (Cisco Systems)

Interesting ports on 192.168.1.23:
PORT   STATE  SERVICE
69/udp closed tftp
MAC Address: 00:15:62:86:BA:3E (Unknown)

Interesting ports on 192.168.1.24:
PORT   STATE  SERVICE
69/udp closed tftp
MAC Address: 00:0E:08:DA:DA:17 (Sipura Technology)

Interesting ports on 192.168.1.25:
PORT   STATE  SERVICE
69/udp closed tftp
MAC Address: 00:0B:82:06:4D:37 (Grandstream Networks)

Interesting ports on 192.168.1.27:
```

```
PORT   STATE         SERVICE
69/udp open|filtered tftp
MAC Address: 00:04:F2:03:15:46 (Circa Communications)

Interesting ports on 192.168.1.51:
PORT   STATE  SERVICE
69/udp closed tftp
MAC Address: 00:04:13:23:34:95 (Snom Technology AG)

Interesting ports on 192.168.1.53:
PORT   STATE  SERVICE
69/udp closed tftp
MAC Address: 00:04:0D:50:40:B0 (Avaya)

Interesting ports on 192.168.1.54:
PORT   STATE  SERVICE
69/udp closed tftp
MAC Address: 00:0E:08:DA:24:AE (Sipura Technology)

Interesting ports on 192.168.1.55:
PORT   STATE         SERVICE
69/udp open|filtered tftp
MAC Address: 00:E0:11:03:03:97 (Uniden SAN Diego R&D Center)

Interesting ports on 192.168.1.57:
PORT   STATE         SERVICE
69/udp open|filtered tftp
MAC Address: 00:01:E1:02:C8:DB (Kinpo Electronics)

Interesting ports on 192.168.1.103:
PORT   STATE         SERVICE
69/udp open|filtered tftp
MAC Address: 00:09:7A:44:15:DB (Louis Design Labs.)

Interesting ports on domain2 (192.168.1.104):
PORT   STATE  SERVICE
69/udp closed tftp
```

As you can see, we found a TFTP server on 192.168.1.103 (also our Asterisk server). Most automated banner-grabbing utilities will identify the TFTP service running on this server. The reason we wanted to run Nmap again is to remind you that the MAC addresses we'll shortly need for enumerating some of the configuration filenames are all available from these scanning results.

As opposed to normal FTP, TFTP provides no mechanism for a directory listing (in other words, no "ls"). This means that unless you already know the names of the files you wish to download, you're out of luck figuring out what else is sitting in the same directory. This is where the MAC addresses will come in handy because we know the general format for Cisco and other phone configuration files is often based on the MAC address. Through brute-force trial-and-error, you can enumerate and download many of the configuration files on a TFTP server.

.COM We have provided an up-to-date list of configuration filenames on our website (http://www.hackingvoip.com) for use with manual or automated TFTP enumeration. Obviously, you will need to modify some of the names with the appropriate MAC addresses gleaned from your own scanning.

Here's an example of enumerating our TFTP server manually by tweaking the list of brute-forcing names we've provided. We use a tool called TFTPbrute.pl, which was written by our colleagues who authored the book, *Hacking Exposed Cisco Networks* (McGraw-Hill, 2006). The tool is available for download from http://www.hackingexposedcisco.com/tools.

```
[root@attacker]# perl tftpbrute.pl 192.168.1.103 brutefile.txt 100
tftpbrute.pl, , V 0.1
TFTP file word database: brutefile.txt
TFTP server 192.168.1.103
Max processes 100
 Processes are: 1
 Processes are: 2
 Processes are: 3
 Processes are: 4
 Processes are: 5
 Processes are: 6
 Processes are: 7
 Processes are: 8
 Processes are: 9
 Processes are: 10
 Processes are: 11
 Processes are: 12
*** Found  TFTP server remote filename : sip.cfg
*** Found  TFTP server remote filename : 46xxsettings.txt
 Processes are: 13
 Processes are: 14
*** Found  TFTP server remote filename : sip_4602D02A.txt
*** Found  TFTP server remote filename : XMLDefault.cnf.xml
*** Found  TFTP server remote filename : SipDefault.cnf
*** Found  TFTP server remote filename : SEP001562EA69E8.cnf
```

Now that we know the name of the configuration file to the target 7960 Cisco IP Phone, we can download it and look for any useful information. For example:

```
[root@attacker]# tftp 192.168.1.103

tftp> get SEP001562EA69E8.cnf

[root@attacker]# cat SEP001562EA69E8.cnf

# SIP Configuration Generic File (start)

# Line 1 Settings
line1_name: "502"                        ; Line 1 Extension\User ID
line1_displayname: "502"                 ; Line 1 Display Name
line1_authname: "502"                    ; Line 1 Registration Authentication
line1_password: "1234"                   ; Line 1 Registration Password

# Line 2 Settings
line2_name: ""                           ; Line 2 Extension\User ID
line2_displayname: ""                    ; Line 2 Display Name
line2_authname: "UNPROVISIONED"          ; Line 2 Registration Authentication
line2_password: "UNPROVISIONED"          ; Line 2 Registration Password

# Line 3 Settings
line3_name: ""                           ; Line 3 Extension\User ID
line3_displayname: ""                    ; Line 3 Display Name
line3_authname: "UNPROVISIONED"          ; Line 3 Registration Authentication
line3_password: "UNPROVISIONED"          ; Line 3 Registration Password

# Line 4 Settings
line4_name: ""                           ; Line 4 Extension\User ID
line4_displayname: ""                    ; Line 4 Display Name
line4_authname: "UNPROVISIONED"          ; Line 4 Registration Authentication
line4_password: "UNPROVISIONED"          ; Line 4 Registration Password

# Line 5 Settings
line5_name: ""                           ; Line 5 Extension\User ID
line5_displayname: ""                    ; Line 5 Display Name
line5_authname: "UNPROVISIONED"          ; Line 5 Registration Authentication
line5_password: "UNPROVISIONED"          ; Line 5 Registration Password

# Line 6 Settings
line6_name: ""                           ; Line 6 Extension\User ID
line6_displayname: ""                    ; Line 6 Display Name
line6_authname: "UNPROVISIONED"          ; Line 6 Registration Authentication
line6_password: "UNPROVISIONED"          ; Line 6 Registration Password
```

```
# NAT/Firewall Traversal
nat_address: ""
voip_control_port: "5060"
start_media_port: "16384"
end_media_port:  "32766"


# Phone Label (Text desired to be displayed in upper right corner)
phone_label: "cisco 7960"            ; Has no effect on SIP messaging

# Time Zone phone will reside in
time_zone: EST

# Phone prompt/password for telnet/console session
phone_prompt: "Cisco7960"                         ; Telnet/Console Prompt
phone_password: "abc"                             ; Telnet/Console Password

# SIP Configuration Generic File (stop)
```

Yikes! Not only does an attacker now know the SIP username and password for this user/phone, but also the administrative password for the telnet service, which also happens to be enabled on this phone.

NOTE Security researcher Ofir Arkin was one of the first to document many of these types of attacks against a Cisco environment in his paper "The Trivial Cisco IP Phones Compromise" (http://www.sys-security.com/archive/papers/The_Trivial_Cisco_IP_Phones_Compromise.pdf).

TFTP Enumeration Countermeasures

While an easy recommendation would be to avoid using TFTP in your VoIP environment, the reality is that many VoIP phones require it and give you no other choice for upgrading or configuration changes. Some of the newer models are beginning to migrate to web configuration instead; however, TFTP will be a necessary evil for the foreseeable future.

Two tips to mitigate the threat of TFTP enumeration include the following:

- Restrict access to TFTP servers by using firewall rules that only allow certain IP address ranges to contact the TFTP server. This prevents arbitrary scanning; however, UDP source addresses can be spoofed.
- Segment the IP phones, TFTP servers, SIP servers, and general VoIP support infrastructure on a separate switched VLAN.

SNMP Enumeration

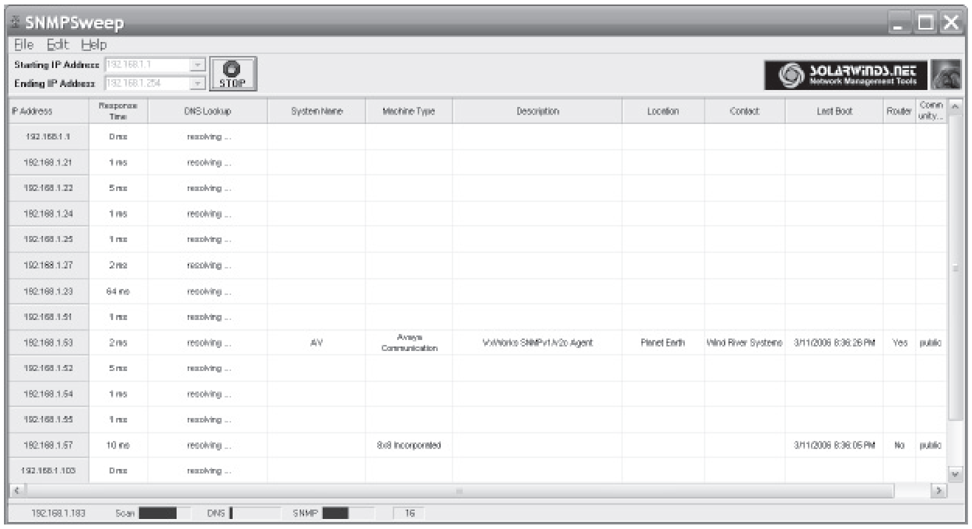
Popularity:	*7*
Simplicity:	*7*
Impact:	*10*
Risk Rating:	***8***

Simple Network Management Protocol (SNMP) version 1 is another inherently insecure protocol used by many VoIP devices, as you learned in Chapter 2. Let's use Nmap again to see if we can find any devices that support it. Because SNMP typically listens on UDP port 162, we'll start off with

```
[root@domain2 ~]# nmap -sU 192.168.1.1-254 -p 162
```

Or you can use a graphical SNMP probing tool such as SolarWinds SNMPSweep, as shown in Figure 3-8.

Based on the information shown in the figure, we now use the "public" community string to enumerate most of the configuration settings on those phones. The tool


SNMPSweep
File Edit Help
Starting IP Address 192.168.1.1
Ending IP Address 192.168.1.254
STOP
SOLARWINDS.NET Network Management Tools

IP Address	Response Time	DNS Lookup	System Name	Machine Type	Description	Location	Contact	Last Boot	Router	Community...
192.168.1.1	0 ms	resolving ...								
192.168.1.21	1 ms	resolving ...								
192.168.1.22	5 ms	resolving ...								
192.168.1.24	1 ms	resolving ...								
192.168.1.25	1 ms	resolving ...								
192.168.1.27	2 ms	resolving ...								
192.168.1.23	64 ms	resolving ...								
192.168.1.51	1 ms	resolving ...								
192.168.1.53	2 ms	resolving ...	AV	Avaya Communication	VxWorks SNMPv1/v2c Agent	Planet Earth	Wind River Systems	3/11/2006 8:36:26 PM	Yes	public
192.168.1.52	5 ms	resolving ...								
192.168.1.54	1 ms	resolving ...								
192.168.1.55	1 ms	resolving ...								
192.168.1.57	10 ms	resolving ...		8x8 Incorporated				3/11/2006 8:36:05 PM	No	public
192.168.1.103	0 ms	resolving ...								

192.168.1.183 Scan DNS SNMP 16

Figure 3-8 SNMPSweep shows that the Avaya IP phone and Zultys Zip2 phone both responded to SNMP probes with the "public" community string.

snmpwalk (http://net-snmp.sourceforge.net/docs/man/snmpwalk.html) is useful for such a task:

```
[root@domain2 ~]# snmpwalk -c public -v 1 192.168.1.53
SNMPv2-MIB::sysDescr.0 = STRING: VxWorks SNMPv1/v2c Agent
SNMPv2-MIB::sysObjectID.0 = OID: SNMPv2-SMI::enterprises.6889.1.69.1.5
SNMPv2-MIB::sysUpTime.0 = Timeticks: (207512) 0:34:35.12
SNMPv2-MIB::sysContact.0 = STRING: Wind River Systems
SNMPv2-MIB::sysName.0 = STRING: AV
SNMPv2-MIB::sysLocation.0 = STRING: Planet Earth
SNMPv2-MIB::sysServices.0 = INTEGER: 79
IF-MIB::ifNumber.0 = INTEGER: 2
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifDescr.1 = STRING: Avaya0
IF-MIB::ifDescr.2 = STRING: lo0
IF-MIB::ifType.1 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.2 = INTEGER: softwareLoopback(24)
IF-MIB::ifMtu.1 = INTEGER: 1500
IF-MIB::ifMtu.2 = INTEGER: 32768
IF-MIB::ifSpeed.1 = Gauge32: 10000000
IF-MIB::ifSpeed.2 = Gauge32: 0
IF-MIB::ifPhysAddress.1 = STRING: 0:4:d:50:40:b0
IF-MIB::ifPhysAddress.2 = STRING:
IF-MIB::ifAdminStatus.1 = INTEGER: up(1)
IF-MIB::ifAdminStatus.2 = INTEGER: up(1)
IF-MIB::ifOperStatus.1 = INTEGER: up(1)
IF-MIB::ifOperStatus.2 = INTEGER: up(1)
IF-MIB::ifLastChange.1 = Timeticks: (0) 0:00:00.00
IF-MIB::ifLastChange.2 = Timeticks: (0) 0:00:00.00
IF-MIB::ifInOctets.1 = Counter32: 0
IF-MIB::ifInOctets.2 = Counter32: 0
IF-MIB::ifInUcastPkts.1 = Counter32: 736
IF-MIB::ifInUcastPkts.2 = Counter32: 106
IF-MIB::ifInNUcastPkts.1 = Counter32: 99
IF-MIB::ifInNUcastPkts.2 = Counter32: 0
IF-MIB::ifInDiscards.1 = Counter32: 0
IF-MIB::ifInDiscards.2 = Counter32: 0
IF-MIB::ifInErrors.1 = Counter32: 0
IF-MIB::ifInErrors.2 = Counter32: 0
IF-MIB::ifInUnknownProtos.1 = Counter32: 0
IF-MIB::ifInUnknownProtos.2 = Counter32: 0
IF-MIB::ifOutOctets.1 = Counter32: 0
IF-MIB::ifOutOctets.2 = Counter32: 0
```

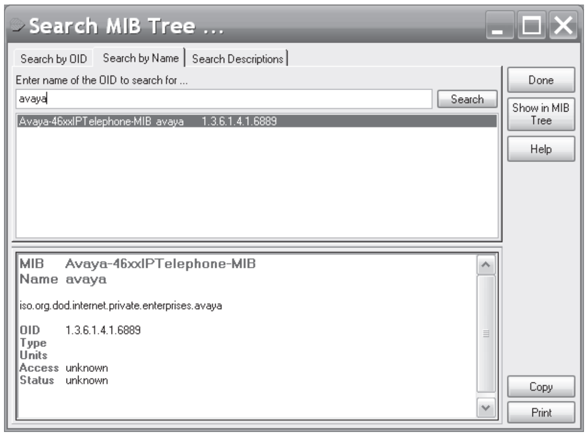
```
IF-MIB::ifOutUcastPkts.1 = Counter32: 742
IF-MIB::ifOutUcastPkts.2 = Counter32: 106
IF-MIB::ifOutNUcastPkts.1 = Counter32: 4
IF-MIB::ifOutNUcastPkts.2 = Counter32: 0
IF-MIB::ifOutDiscards.1 = Counter32: 0
IF-MIB::ifOutDiscards.2 = Counter32: 0
IF-MIB::ifOutErrors.1 = Counter32: 0
IF-MIB::ifOutErrors.2 = Counter32: 0
IF-MIB::ifOutQLen.1 = Gauge32: 0
IF-MIB::ifOutQLen.2 = Gauge32: 0
IF-MIB::ifSpecific.1 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.2 = OID: SNMPv2-SMI::zeroDotZero
IP-MIB::ipForwarding.0 = INTEGER: forwarding(1)
IP-MIB::ipDefaultTTL.0 = INTEGER: 64
IP-MIB::ipInReceives.0 = Counter32: 864
IP-MIB::ipInHdrErrors.0 = Counter32: 0
IP-MIB::ipInAddrErrors.0 = Counter32: 0
IP-MIB::ipForwDatagrams.0 = Counter32: 0
IP-MIB::ipInUnknownProtos.0 = Counter32: 1
IP-MIB::ipInDiscards.0 = Counter32: 0
IP-MIB::ipInDelivers.0 = Counter32: 869
IP-MIB::ipOutRequests.0 = Counter32: 857
IP-MIB::ipOutDiscards.0 = Counter32: 0
IP-MIB::ipOutNoRoutes.0 = Counter32: 0
IP-MIB::ipReasmTimeout.0 = INTEGER: 60
IP-MIB::ipReasmReqds.0 = Counter32: 0
IP-MIB::ipReasmOKs.0 = Counter32: 0
IP-MIB::ipReasmFails.0 = Counter32: 0
IP-MIB::ipFragOKs.0 = Counter32: 0
IP-MIB::ipFragFails.0 = Counter32: 0
IP-MIB::ipFragCreates.0 = Counter32: 0
IP-MIB::ipAdEntAddr.127.0.0.1 = IpAddress: 127.0.0.1
IP-MIB::ipAdEntAddr.192.168.1.53 = IpAddress: 192.168.1.53
IP-MIB::ipAdEntIfIndex.127.0.0.1 = INTEGER: 2
IP-MIB::ipAdEntIfIndex.192.168.1.53 = INTEGER: 1
IP-MIB::ipAdEntNetMask.127.0.0.1 = IpAddress: 255.0.0.0
IP-MIB::ipAdEntNetMask.192.168.1.53 = IpAddress: 255.255.255.0
IP-MIB::ipAdEntBcastAddr.127.0.0.1 = INTEGER: 1
IP-MIB::ipAdEntBcastAddr.192.168.1.53 = INTEGER: 1
IP-MIB::ipAdEntReasmMaxSize.127.0.0.1 = INTEGER: 65535
IP-MIB::ipAdEntReasmMaxSize.192.168.1.53 = INTEGER: 65535
RFC1213-MIB::ipRouteDest.0.0.0.0 = IpAddress: 0.0.0.0
RFC1213-MIB::ipRouteDest.24.93.41.125 = IpAddress: 24.93.41.125
RFC1213-MIB::ipRouteDest.127.0.0.1 = IpAddress: 127.0.0.1
```

```
RFC1213-MIB::ipRouteDest.192.168.1.0 = IpAddress: 192.168.1.0
RFC1213-MIB::ipRouteIfIndex.0.0.0.0 = INTEGER: 1
RFC1213-MIB::ipRouteIfIndex.24.93.41.125 = INTEGER: 1
RFC1213-MIB::ipRouteIfIndex.127.0.0.1 = INTEGER: 2
RFC1213-MIB::ipRouteIfIndex.192.168.1.0 = INTEGER: 1
RFC1213-MIB::ipRouteMetric1.0.0.0.0 = INTEGER: 1
RFC1213-MIB::ipRouteMetric1.24.93.41.125 = INTEGER: 1
RFC1213-MIB::ipRouteMetric1.127.0.0.1 = INTEGER: 0
RFC1213-MIB::ipRouteMetric1.192.168.1.0 = INTEGER: 0
RFC1213-MIB::ipRouteMetric2.0.0.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric2.24.93.41.125 = INTEGER: -1
RFC1213-MIB::ipRouteMetric2.127.0.0.1 = INTEGER: -1
RFC1213-MIB::ipRouteMetric2.192.168.1.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric3.0.0.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric3.24.93.41.125 = INTEGER: -1
RFC1213-MIB::ipRouteMetric3.127.0.0.1 = INTEGER: -1
RFC1213-MIB::ipRouteMetric3.192.168.1.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric4.0.0.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric4.24.93.41.125 = INTEGER: -1
RFC1213-MIB::ipRouteMetric4.127.0.0.1 = INTEGER: -1
RFC1213-MIB::ipRouteMetric4.192.168.1.0 = INTEGER: -1
RFC1213-MIB::ipRouteNextHop.0.0.0.0 = IpAddress: 192.168.1.1
RFC1213-MIB::ipRouteNextHop.24.93.41.125 = IpAddress: 192.168.1.1
RFC1213-MIB::ipRouteNextHop.127.0.0.1 = IpAddress: 127.0.0.1
RFC1213-MIB::ipRouteNextHop.192.168.1.0 = IpAddress: 192.168.1.53
RFC1213-MIB::ipRouteType.0.0.0.0 = INTEGER: indirect(4)
RFC1213-MIB::ipRouteType.24.93.41.125 = INTEGER: indirect(4)
RFC1213-MIB::ipRouteType.127.0.0.1 = INTEGER: direct(3)
RFC1213-MIB::ipRouteType.192.168.1.0 = INTEGER: direct(3)
RFC1213-MIB::ipRouteProto.0.0.0.0 = INTEGER: other(1)
RFC1213-MIB::ipRouteProto.24.93.41.125 = INTEGER: local(2)
RFC1213-MIB::ipRouteProto.127.0.0.1 = INTEGER: local(2)
RFC1213-MIB::ipRouteProto.192.168.1.0 = INTEGER: local(2)
RFC1213-MIB::ipRouteAge.0.0.0.0 = INTEGER: 2067
RFC1213-MIB::ipRouteAge.24.93.41.125 = INTEGER: 2025
RFC1213-MIB::ipRouteAge.127.0.0.1 = INTEGER: 2079
RFC1213-MIB::ipRouteAge.192.168.1.0 = INTEGER: 2068
RFC1213-MIB::ipRouteMask.0.0.0.0 = IpAddress: 0.0.0.0
RFC1213-MIB::ipRouteMask.24.93.41.125 = IpAddress: 255.255.255.255
RFC1213-MIB::ipRouteMask.127.0.0.1 = IpAddress: 255.255.255.255
RFC1213-MIB::ipRouteMask.192.168.1.0 = IpAddress: 255.255.255.0
RFC1213-MIB::ipRouteMetric5.0.0.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric5.24.93.41.125 = INTEGER: -1
RFC1213-MIB::ipRouteMetric5.127.0.0.1 = INTEGER: -1
```

```
RFC1213-MIB::ipRouteMetric5.192.168.1.0 = INTEGER: -1
RFC1213-MIB::ipRouteInfo.0.0.0.0 = OID: SNMPv2-SMI::zeroDotZero
RFC1213-MIB::ipRouteInfo.24.93.41.125 = OID: SNMPv2-SMI::zeroDotZero
RFC1213-MIB::ipRouteInfo.127.0.0.1 = OID: SNMPv2-SMI::zeroDotZero
RFC1213-MIB::ipRouteInfo.192.168.1.0 = OID: SNMPv2-SMI::zeroDotZero
IP-MIB::ipNetToMediaIfIndex.1.192.168.1.104 = INTEGER: 1
IP-MIB::ipNetToMediaIfIndex.2.192.168.1.53 = INTEGER: 2
IP-MIB::ipNetToMediaPhysAddress.1.192.168.1.104 = STRING: 0:9:7a:44:17:d9
IP-MIB::ipNetToMediaPhysAddress.2.192.168.1.53 = STRING: 0:4:d:50:40:b0
IP-MIB::ipNetToMediaNetAddress.1.192.168.1.104 = IpAddress: 192.168.1.104
IP-MIB::ipNetToMediaNetAddress.2.192.168.1.53 = IpAddress: 192.168.1.53
IP-MIB::ipNetToMediaType.1.192.168.1.104 = INTEGER: dynamic(3)
IP-MIB::ipNetToMediaType.2.192.168.1.53 = INTEGER: static(4)
IP-MIB::ipRoutingDiscards.0 = Counter32: 0
IP-MIB::icmpInMsgs.0 = Counter32: 4
IP-MIB::icmpInErrors.0 = Counter32: 0
IP-MIB::icmpInDestUnreachs.0 = Counter32: 1
IP-MIB::icmpInTimeExcds.0 = Counter32: 0
IP-MIB::icmpInParmProbs.0 = Counter32: 0
IP-MIB::icmpInSrcQuenchs.0 = Counter32: 0
IP-MIB::icmpInRedirects.0 = Counter32: 0
IP-MIB::icmpInEchos.0 = Counter32: 3
IP-MIB::icmpInEchoReps.0 = Counter32: 0
IP-MIB::icmpInTimestamps.0 = Counter32: 0
IP-MIB::icmpInTimestampReps.0 = Counter32: 0
IP-MIB::icmpInAddrMasks.0 = Counter32: 0
IP-MIB::icmpInAddrMaskReps.0 = Counter32: 0
IP-MIB::icmpOutMsgs.0 = Counter32: 27
IP-MIB::icmpOutErrors.0 = Counter32: 24
IP-MIB::icmpOutDestUnreachs.0 = Counter32: 24
IP-MIB::icmpOutTimeExcds.0 = Counter32: 0
IP-MIB::icmpOutParmProbs.0 = Counter32: 0
IP-MIB::icmpOutSrcQuenchs.0 = Counter32: 0
IP-MIB::icmpOutRedirects.0 = Counter32: 0
IP-MIB::icmpOutEchos.0 = Counter32: 0
IP-MIB::icmpOutEchoReps.0 = Counter32: 3
IP-MIB::icmpOutTimestamps.0 = Counter32: 0
IP-MIB::icmpOutTimestampReps.0 = Counter32: 0
IP-MIB::icmpOutAddrMasks.0 = Counter32: 0
IP-MIB::icmpOutAddrMaskReps.0 = Counter32: 0
TCP-MIB::tcpRtoAlgorithm.0 = INTEGER: vanj(4)
TCP-MIB::tcpRtoMin.0 = INTEGER: 1000 milliseconds
TCP-MIB::tcpRtoMax.0 = INTEGER: 64000 milliseconds
TCP-MIB::tcpMaxConn.0 = INTEGER: -1
```

```
TCP-MIB::tcpActiveOpens.0 = Counter32: 6
TCP-MIB::tcpPassiveOpens.0 = Counter32: 4
TCP-MIB::tcpAttemptFails.0 = Counter32: 1
TCP-MIB::tcpEstabResets.0 = Counter32: 0
TCP-MIB::tcpCurrEstab.0 = Gauge32: 0
TCP-MIB::tcpInSegs.0 = Counter32: 96
TCP-MIB::tcpOutSegs.0 = Counter32: 99
TCP-MIB::tcpRetransSegs.0 = Counter32: 0
TCP-MIB::tcpInErrs.0 = Counter32: 0
TCP-MIB::tcpOutRsts.0 = Counter32: 0
UDP-MIB::udpInDatagrams.0 = Counter32: 890
UDP-MIB::udpNoPorts.0 = Counter32: 26
UDP-MIB::udpInErrors.0 = Counter32: 0
UDP-MIB::udpOutDatagrams.0 = Counter32: 855
UDP-MIB::udpLocalAddress.0.0.0.0.68 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.0.0.0.0.161 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.0.0.0.0.1031 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.0.0.0.0.1033 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.0.0.0.0.5060 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.0.0.0.0.10000 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.127.0.0.1.1032 = IpAddress: 127.0.0.1
UDP-MIB::udpLocalPort.0.0.0.0.68 = INTEGER: 68
UDP-MIB::udpLocalPort.0.0.0.0.161 = INTEGER: 161
UDP-MIB::udpLocalPort.0.0.0.0.1031 = INTEGER: 1031
UDP-MIB::udpLocalPort.0.0.0.0.1033 = INTEGER: 1033
UDP-MIB::udpLocalPort.0.0.0.0.5060 = INTEGER: 5060
UDP-MIB::udpLocalPort.0.0.0.0.10000 = INTEGER: 10000
UDP-MIB::udpLocalPort.127.0.0.1.1032 = INTEGER: 1032
SNMPv2-MIB::snmpInPkts.0 = Counter32: 799
SNMPv2-MIB::snmpOutPkts.0 = Counter32: 788
SNMPv2-MIB::snmpInBadVersions.0 = Counter32: 6
SNMPv2-MIB::snmpInBadCommunityNames.0 = Counter32: 6
SNMPv2-MIB::snmpInBadCommunityUses.0 = Counter32: 0
SNMPv2-MIB::snmpInASNParseErrs.0 = Counter32: 0
SNMPv2-MIB::snmpInTooBigs.0 = Counter32: 0
SNMPv2-MIB::snmpInNoSuchNames.0 = Counter32: 0
SNMPv2-MIB::snmpInBadValues.0 = Counter32: 0
SNMPv2-MIB::snmpInReadOnlys.0 = Counter32: 0
SNMPv2-MIB::snmpInGenErrs.0 = Counter32: 0
SNMPv2-MIB::snmpInTotalReqVars.0 = Counter32: 1067
SNMPv2-MIB::snmpInTotalSetVars.0 = Counter32: 0
SNMPv2-MIB::snmpInGetRequests.0 = Counter32: 22
SNMPv2-MIB::snmpInGetNexts.0 = Counter32: 749
SNMPv2-MIB::snmpInSetRequests.0 = Counter32: 0
```

```
SNMPv2-MIB::snmpInGetResponses.0 = Counter32: 0
SNMPv2-MIB::snmpInTraps.0 = Counter32: 0
SNMPv2-MIB::snmpOutTooBigs.0 = Counter32: 0
SNMPv2-MIB::snmpOutNoSuchNames.0 = Counter32: 0
SNMPv2-MIB::snmpOutBadValues.0 = Counter32: 0
SNMPv2-MIB::snmpOutGenErrs.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetNexts.0 = Counter32: 0
SNMPv2-MIB::snmpOutSetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetResponses.0 = Counter32: 811
SNMPv2-MIB::snmpOutTraps.0 = Counter32: 1
SNMPv2-MIB::snmpEnableAuthenTraps.0 = INTEGER: disabled(2)
SNMPv2-MIB::snmpSilentDrops.0 = Counter32: 0
SNMPv2-MIB::snmpProxyDrops.0 = Counter32: 0
IF-MIB::ifName.1 = STRING:
IF-MIB::ifName.2 = STRING:
IF-MIB::ifInMulticastPkts.1 = Counter32: 0
IF-MIB::ifInMulticastPkts.2 = Counter32: 0
IF-MIB::ifInBroadcastPkts.1 = Counter32: 0
IF-MIB::ifInBroadcastPkts.2 = Counter32: 0
IF-MIB::ifOutMulticastPkts.1 = Counter32: 0
IF-MIB::ifOutMulticastPkts.2 = Counter32: 0
IF-MIB::ifOutBroadcastPkts.1 = Counter32: 0
IF-MIB::ifOutBroadcastPkts.2 = Counter32: 0
IF-MIB::ifLinkUpDownTrapEnable.1 = INTEGER: disabled(2)
IF-MIB::ifLinkUpDownTrapEnable.2 = INTEGER: disabled(2)
IF-MIB::ifHighSpeed.1 = Gauge32: 0
IF-MIB::ifHighSpeed.2 = Gauge32: 0
IF-MIB::ifPromiscuousMode.1 = INTEGER: false(2)
IF-MIB::ifPromiscuousMode.2 = INTEGER: false(2)
IF-MIB::ifConnectorPresent.1 = INTEGER: false(2)
IF-MIB::ifConnectorPresent.2 = INTEGER: false(2)
IF-MIB::ifAlias.1 = STRING:
IF-MIB::ifAlias.2 = STRING:
IF-MIB::ifCounterDiscontinuityTime.1 = Timeticks: (0) 0:00:00.00
IF-MIB::ifCounterDiscontinuityTime.2 = Timeticks: (0) 0:00:00.00
IF-MIB::ifStackStatus.0.1 = INTEGER: active(1)
IF-MIB::ifStackStatus.0.2 = INTEGER: active(1)
IF-MIB::ifStackStatus.1.0 = INTEGER: active(1)
IF-MIB::ifStackStatus.2.0 = INTEGER: active(1)
IF-MIB::ifRcvAddressStatus.1."...P@." = INTEGER: active(1)
IF-MIB::ifRcvAddressType.1."...P@." = INTEGER: nonVolatile(3)
IF-MIB::ifTableLastChange.0 = Timeticks: (0) 0:00:00.00
IF-MIB::ifStackLastChange.0 = Timeticks: (0) 0:00:00.00
```



Enter name of the OID to search for ...
avaya

Avaya-46xxIPTelephone-MIB avaya 1.3.6.1.4.1.6889

```
MIB     Avaya-46xxIPTelephone-MIB
Name    avaya

iso.org.dod.internet.private.enterprises.avaya

OID     1.3.6.1.4.1.6889
Type
Units
Access  unknown
Status  unknown
```

Figure 3-9 SolarWind's MIB browser finding the Avaya OID

The SNMP MIB exposes interesting configuration information about the Avaya phone, including its vendor type (Avaya), underlying operating system (VxWorks), MAC address, and ports of possible UDP-related services that might be of interest for further enumeration (68, 161, 1031, 1032, 1033, 5060). Now that we know this is an Avaya phone, we can easily find specific SNMP MIB information for this manufacturer from Google to query for further information on this device. You can also use Solarwinds' graphical tool, SNMP MIB browser, which has a built-in MIB database and is shown in Figure 3-9. We eventually determine that 1.3.6.1.4.1.6889 is the appropriate SNMP OID, which we can use for another, more detailed SNMP query:

```
[root@domain2 ~]# snmpwalk -c public -v 1 192.168.1.53 1.3.6.1.4.1.6889
SNMPv2-SMI::enterprises.6889.2.69.1.1.1.0 = STRING: "Obsolete"
SNMPv2-SMI::enterprises.6889.2.69.1.1.2.0 = STRING: "4620D01B"
SNMPv2-SMI::enterprises.6889.2.69.1.1.3.0 = STRING: "AvayaCallserver"
SNMPv2-SMI::enterprises.6889.2.69.1.1.4.0 = IpAddress: 192.168.1.104
SNMPv2-SMI::enterprises.6889.2.69.1.1.5.0 = INTEGER: 1719
SNMPv2-SMI::enterprises.6889.2.69.1.1.6.0 = STRING: "051612501065"
SNMPv2-SMI::enterprises.6889.2.69.1.1.7.0 = STRING: "700316698"
SNMPv2-SMI::enterprises.6889.2.69.1.1.8.0 = STRING: "051611403489"
SNMPv2-SMI::enterprises.6889.2.69.1.1.9.0 = STRING: "00:04:0D:50:40:B0"
SNMPv2-SMI::enterprises.6889.2.69.1.1.10.0 = STRING: "100"
SNMPv2-SMI::enterprises.6889.2.69.1.1.11.0 = IpAddress: 192.168.1.53
SNMPv2-SMI::enterprises.6889.2.69.1.1.12.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.13.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.14.0 = INTEGER: 0
```

```
SNMPv2-SMI::enterprises.6889.2.69.1.1.15.0 = STRING: "192.168.1.1"
SNMPv2-SMI::enterprises.6889.2.69.1.1.16.0 = IpAddress: 192.168.1.1
SNMPv2-SMI::enterprises.6889.2.69.1.1.17.0 = IpAddress: 255.255.255.0
SNMPv2-SMI::enterprises.6889.2.69.1.1.18.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.1.19.0 = STRING: "192.168.1.104"
SNMPv2-SMI::enterprises.6889.2.69.1.1.20.0 = IpAddress: 192.168.1.104
SNMPv2-SMI::enterprises.6889.2.69.1.1.21.0 = STRING: "b20d01b2_3.bin"
SNMPv2-SMI::enterprises.6889.2.69.1.1.22.0 = STRING: "s20d01b2_2.bin"
SNMPv2-SMI::enterprises.6889.2.69.1.1.23.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.1.24.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.1.25.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.1.26.0 = INTEGER: 46
SNMPv2-SMI::enterprises.6889.2.69.1.1.27.0 = INTEGER: 34
SNMPv2-SMI::enterprises.6889.2.69.1.1.28.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.29.0 = INTEGER: 6
SNMPv2-SMI::enterprises.6889.2.69.1.1.30.0 = INTEGER: 6
SNMPv2-SMI::enterprises.6889.2.69.1.1.31.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.32.0 = STRING: "46xxupgrade.scr"
SNMPv2-SMI::enterprises.6889.2.69.1.1.33.0 = STRING: "24.93.41.125"
SNMPv2-SMI::enterprises.6889.2.69.1.1.34.0 = STRING: "Obsolete"
SNMPv2-SMI::enterprises.6889.2.69.1.1.35.0 = STRING: "domain2"
SNMPv2-SMI::enterprises.6889.2.69.1.1.36.0 = IpAddress: 0.0.0.0
SNMPv2-SMI::enterprises.6889.2.69.1.1.37.0 = INTEGER: 1
SNMPv2-SMI::enterprises.6889.2.69.1.1.38.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.39.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.1.40.0 = STRING: "<ZSPV_x.x>"
SNMPv2-SMI::enterprises.6889.2.69.1.1.41.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.1.42.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.1.43.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.44.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.45.0 = INTEGER: 1
SNMPv2-SMI::enterprises.6889.2.69.1.1.46.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.47.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.1.48.0 = STRING: "700259674"
SNMPv2-SMI::enterprises.6889.2.69.1.1.49.0 = STRING: " "
SNMPv2-SMI::enterprises.6889.2.69.1.1.50.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.51.0 = STRING: "192.168.1.104"
SNMPv2-SMI::enterprises.6889.2.69.1.1.52.0 = STRING: "Obsolete"
SNMPv2-SMI::enterprises.6889.2.69.1.1.53.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.1.54.0 = STRING: "0.0.0.0"
SNMPv2-SMI::enterprises.6889.2.69.1.1.55.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.56.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.57.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.58.0 = INTEGER: 0
```

```
SNMPv2-SMI::enterprises.6889.2.69.1.1.59.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.1.60.0 = STRING: "Obsolete"
SNMPv2-SMI::enterprises.6889.2.69.1.1.61.0 = STRING: "Obsolete"
SNMPv2-SMI::enterprises.6889.2.69.1.1.62.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.1.63.0 = STRING: "192.168.1.104"
SNMPv2-SMI::enterprises.6889.2.69.1.1.64.0 = STRING: "0.0.0.0"
SNMPv2-SMI::enterprises.6889.2.69.1.1.65.0 = INTEGER: 50002
SNMPv2-SMI::enterprises.6889.2.69.1.1.66.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.1.67.0 = STRING: "1"
SNMPv2-SMI::enterprises.6889.2.69.1.1.68.0 = STRING: "1SunApr2L"
SNMPv2-SMI::enterprises.6889.2.69.1.1.69.0 = STRING: "LSunOct2L"
SNMPv2-SMI::enterprises.6889.2.69.1.1.70.0 = STRING: "0:00"
SNMPv2-SMI::enterprises.6889.2.69.1.1.71.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.1.72.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.1.0 = STRING: "0.0.0.0"
SNMPv2-SMI::enterprises.6889.2.69.1.2.2.0 = INTEGER: 1719
SNMPv2-SMI::enterprises.6889.2.69.1.2.3.0 = STRING: "192.168.1.53"
SNMPv2-SMI::enterprises.6889.2.69.1.2.4.0 = STRING: "192.168.1.1"
SNMPv2-SMI::enterprises.6889.2.69.1.2.5.0 = STRING: "255.255.255.0"
SNMPv2-SMI::enterprises.6889.2.69.1.2.6.0 = STRING: "Obsolete"
SNMPv2-SMI::enterprises.6889.2.69.1.2.7.0 = INTEGER: 176
SNMPv2-SMI::enterprises.6889.2.69.1.2.8.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.2.9.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.2.10.0 = INTEGER: 46
SNMPv2-SMI::enterprises.6889.2.69.1.2.11.0 = INTEGER: 34
SNMPv2-SMI::enterprises.6889.2.69.1.2.12.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.13.0 = INTEGER: 6
SNMPv2-SMI::enterprises.6889.2.69.1.2.14.0 = INTEGER: 6
SNMPv2-SMI::enterprises.6889.2.69.1.2.15.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.16.0 = INTEGER: 1
SNMPv2-SMI::enterprises.6889.2.69.1.2.17.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.18.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.19.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.20.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.21.0 = INTEGER: 1
SNMPv2-SMI::enterprises.6889.2.69.1.2.22.0 = INTEGER: 60
SNMPv2-SMI::enterprises.6889.2.69.1.2.23.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.2.24.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.25.0 = STRING: "Obsolete"
SNMPv2-SMI::enterprises.6889.2.69.1.2.26.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.27.0 = STRING: "192.168.1.104"
SNMPv2-SMI::enterprises.6889.2.69.1.2.28.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.29.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.2.30.0 = INTEGER: 0
```

```
SNMPv2-SMI::enterprises.6889.2.69.1.3.1.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.3.2.0 = STRING: "s20d01b2_2.bin"
SNMPv2-SMI::enterprises.6889.2.69.1.3.3.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.3.4.1.1.1 = STRING:
"Jan 01 00:00:00:tConfig:Unexpected Msg from <Int Lvl>: mt=19, st=72"
SNMPv2-SMI::enterprises.6889.2.69.1.3.4.1.1.2 = STRING:
"Jan 01 00:00:00:tHttpDownLoop:httpdownload: Connection Timeout"
SNMPv2-SMI::enterprises.6889.2.69.1.3.4.1.1.3 = STRING:
"Jan 01 00:00:00:tBoot:http_getScript: Download failed
"
SNMPv2-SMI::enterprises.6889.2.69.1.3.4.1.1.4 = STRING:
"Jan 01 00:00:00:tBoot:http_getScript: Download failed
"
SNMPv2-SMI::enterprises.6889.2.69.1.3.4.1.1.5 = STRING: "Jan 01 00:00:00:
tBoot:msgQSend failed (mt=2, st=0) errno=3d0001, QID=0x80e43480"
SNMPv2-SMI::enterprises.6889.2.69.1.3.4.1.1.6 = STRING:
"Jan 01 00:00:02:tPhone:Unexpected Msg from tUiDirector: mt=26, st=107"
SNMPv2-SMI::enterprises.6889.2.69.1.4.1.0 = INTEGER: 5004
SNMPv2-SMI::enterprises.6889.2.69.1.4.2.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.3.0 = STRING: "0.0.0.0"
SNMPv2-SMI::enterprises.6889.2.69.1.4.4.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.5.0 = STRING: "EM_AudioCapability_
g711Ulaw64k_chosen"
SNMPv2-SMI::enterprises.6889.2.69.1.4.6.0 = STRING: "EM_AudioCapability_
g711Ulaw64k_chosen"
SNMPv2-SMI::enterprises.6889.2.69.1.4.7.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.8.0 = INTEGER: 20
SNMPv2-SMI::enterprises.6889.2.69.1.4.9.0 = STRING: "503"
SNMPv2-SMI::enterprises.6889.2.69.1.4.10.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.11.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.12.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.13.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.14.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.15.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.16.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.17.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.18.0 = INTEGER: 1
SNMPv2-SMI::enterprises.6889.2.69.1.4.19.0 = INTEGER: 5
SNMPv2-SMI::enterprises.6889.2.69.1.4.20.0 = INTEGER: 11
SNMPv2-SMI::enterprises.6889.2.69.1.4.21.0 = INTEGER: 1
SNMPv2-SMI::enterprises.6889.2.69.1.4.22.0 = INTEGER: 10
SNMPv2-SMI::enterprises.6889.2.69.1.4.23.0 = INTEGER: 9
SNMPv2-SMI::enterprises.6889.2.69.1.4.24.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.4.25.0 = INTEGER: 1
```

```
SNMPv2-SMI::enterprises.6889.2.69.1.4.26.0 = INTEGER: 1
SNMPv2-SMI::enterprises.6889.2.69.1.4.27.0 = INTEGER: 1
SNMPv2-SMI::enterprises.6889.2.69.1.5.1.0 = STRING: "Obsolete"
SNMPv2-SMI::enterprises.6889.2.69.1.5.2.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.3.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.4.0 = STRING: "cn"
SNMPv2-SMI::enterprises.6889.2.69.1.5.5.0 = STRING: "telephoneNumber"
SNMPv2-SMI::enterprises.6889.2.69.1.5.6.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.5.7.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.8.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.9.0 = STRING: "Latin 1"
SNMPv2-SMI::enterprises.6889.2.69.1.5.10.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.5.11.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.12.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.13.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.14.0 = INTEGER: 3
SNMPv2-SMI::enterprises.6889.2.69.1.5.15.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.5.16.0 = STRING: "  "
SNMPv2-SMI::enterprises.6889.2.69.1.5.17.0 = STRING: "  "
SNMPv2-SMI::enterprises.6889.2.69.1.5.18.0 = STRING: "  "
SNMPv2-SMI::enterprises.6889.2.69.1.5.19.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.5.20.0 = STRING: "  "
SNMPv2-SMI::enterprises.6889.2.69.1.5.21.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.5.22.0 = STRING: "  "
SNMPv2-SMI::enterprises.6889.2.69.1.5.23.0 = STRING: "  "
SNMPv2-SMI::enterprises.6889.2.69.1.5.24.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.5.25.0 = STRING: "Obsolete"
SNMPv2-SMI::enterprises.6889.2.69.1.5.26.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.27.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.28.0 = INTEGER: 8000
SNMPv2-SMI::enterprises.6889.2.69.1.5.29.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.30.0 = INTEGER: 1
SNMPv2-SMI::enterprises.6889.2.69.1.5.31.0 = INTEGER: 49721
SNMPv2-SMI::enterprises.6889.2.69.1.5.32.0 = INTEGER: -1
SNMPv2-SMI::enterprises.6889.2.69.1.5.33.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.5.34.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.6.1.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.6.2.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.7.1.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.7.2.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.7.3.0 = INTEGER: 0
SNMPv2-SMI::enterprises.6889.2.69.1.7.4.0 = ""
SNMPv2-SMI::enterprises.6889.2.69.1.7.5.0 = IpAddress: 0.0.0.0
SNMPv2-SMI::enterprises.6889.2.69.1.7.6.0 = ""
```

```
SNMPv2-SMI::enterprises.6889.2.69.1.7.7.0 = IpAddress: 0.0.0.0
SNMPv2-SMI::enterprises.6889.2.69.1.7.8.0 = INTEGER: 3600
SNMPv2-SMI::enterprises.6889.2.69.1.7.9.0 = STRING: "192.168.1.104"
SNMPv2-SMI::enterprises.6889.2.69.1.7.10.0 = IpAddress: 192.168.1.104
SNMPv2-SMI::enterprises.6889.2.69.1.7.11.0 = STRING: "192.168.1.104"
SNMPv2-SMI::enterprises.6889.2.69.1.7.12.0 = IpAddress: 0.0.0.0
SNMPv2-SMI::enterprises.6889.2.69.1.7.13.0 = INTEGER: 2
```

This vendor-specific SNMP query gave us even more information about the phone, including its SIP username (503), DNS server, configuration HTTP server IP address (192.168.1.104), its SIP domain (domain2), and other juicy configuration details. SolarWinds also has a graphical equivalent to snmpwalk called SNMP MIB Browser, which includes the database used earlier to find the Avaya OID.

SNMP Enumeration Countermeasures

If possible, disable SNMP support on your phones. Change the default public and private SNMP community strings on all other network devices running SNMP v1 and v2. Upgrade any devices to SNMP v3, which supports strong authentication rather than simple text strings (public/private community strings).

Enumerating VxWorks VoIP Devices

Popularity:	2
Simplicity:	3
Impact:	10
Risk Rating:	**5**

Many IP phones are developed on embedded real-time operating systems, such as VxWorks (http://www.vxworks.com). Before the phone actually ships, some vendors forget to turn off the remote debugging feature of VxWorks, which allows for administrative debugging access to the device. The VxWorks remote debugger typically listens on UDP or TCP port 17185 and allows connections from a remote debugging client.

Let's try scanning our test deployment with Nmap to see if any of our phones respond on those ports:

```
[root@domain2 ~]# nmap -sT 192.168.1.1-254 -p 17185

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2006-03-11 22:19 CST
Interesting ports on 192.168.1.21:
PORT      STATE  SERVICE
17185/tcp closed unknown
MAC Address: 00:04:13:24:23:8D (Snom Technology AG)
```

```
Interesting ports on 192.168.1.22:
PORT      STATE    SERVICE
17185/tcp filtered unknown
MAC Address: 00:0F:34:11:80:45 (Cisco Systems)

Interesting ports on 192.168.1.23:
PORT      STATE  SERVICE
17185/tcp closed unknown
MAC Address: 00:15:62:86:BA:3E (Unknown)

Interesting ports on 192.168.1.24:
PORT      STATE  SERVICE
17185/tcp closed unknown
MAC Address: 00:0E:08:DA:DA:17 (Sipura Technology)

Interesting ports on 192.168.1.25:
PORT      STATE    SERVICE
17185/tcp filtered unknown
MAC Address: 00:0B:82:06:4D:37 (Grandstream Networks)

Interesting ports on 192.168.1.27:
PORT      STATE  SERVICE
17185/tcp closed unknown
MAC Address: 00:04:F2:03:15:46 (Circa Communications)

Interesting ports on 192.168.1.51:
PORT      STATE  SERVICE
17185/tcp closed unknown
MAC Address: 00:04:13:23:34:95 (Snom Technology AG)

Interesting ports on 192.168.1.53:
PORT      STATE  SERVICE
17185/tcp closed unknown
MAC Address: 00:04:0D:50:40:B0 (Avaya)

Interesting ports on 192.168.1.54:
PORT      STATE  SERVICE
17185/tcp closed unknown
MAC Address: 00:0E:08:DA:24:AE (Sipura Technology)

Interesting ports on 192.168.1.57:
PORT      STATE  SERVICE
17185/tcp closed unknown
MAC Address: 00:01:E1:02:C8:DB (Kinpo Electronics)
```

```
Interesting ports on 192.168.1.103:
PORT      STATE  SERVICE
17185/tcp closed unknown
MAC Address: 00:09:7A:44:15:DB (Louis Design Labs.)

Interesting ports on domain2 (192.168.1.104):
PORT      STATE  SERVICE
17185/tcp closed unknown

Nmap finished: 109 IP addresses (12 hosts up) scanned in 19.239 seconds
[root@domain2 ~]# nmap -sU 192.168.1.1-254 -p 17185

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2006-03-11 22:21 CST
Interesting ports on 192.168.1.21:
PORT      STATE  SERVICE
17185/udp closed wdbrpc
MAC Address: 00:04:13:24:23:8D (Snom Technology AG)

Interesting ports on 192.168.1.22:
PORT      STATE         SERVICE
17185/udp open|filtered wdbrpc
MAC Address: 00:0F:34:11:80:45 (Cisco Systems)

Interesting ports on 192.168.1.23:
PORT      STATE  SERVICE
17185/udp closed wdbrpc
MAC Address: 00:15:62:86:BA:3E (Unknown)

Interesting ports on 192.168.1.24:
PORT      STATE  SERVICE
17185/udp closed wdbrpc
MAC Address: 00:0E:08:DA:DA:17 (Sipura Technology)

Interesting ports on 192.168.1.25:
PORT      STATE  SERVICE
17185/udp closed wdbrpc
MAC Address: 00:0B:82:06:4D:37 (Grandstream Networks)

Interesting ports on 192.168.1.27:
PORT      STATE         SERVICE
17185/udp open|filtered wdbrpc
MAC Address: 00:04:F2:03:15:46 (Circa Communications)
```

```
Interesting ports on 192.168.1.51:
PORT      STATE  SERVICE
17185/udp closed wdbrpc
MAC Address: 00:04:13:23:34:95 (Snom Technology AG)

Interesting ports on 192.168.1.53:
PORT      STATE  SERVICE
17185/udp closed wdbrpc
MAC Address: 00:04:0D:50:40:B0 (Avaya)

Interesting ports on 192.168.1.54:
PORT      STATE  SERVICE
17185/udp closed wdbrpc
MAC Address: 00:0E:08:DA:24:AE (Sipura Technology)

Interesting ports on 192.168.1.57:
PORT      STATE         SERVICE
17185/udp open|filtered wdbrpc
MAC Address: 00:01:E1:02:C8:DB (Kinpo Electronics)

Interesting ports on 192.168.1.103:
PORT      STATE  SERVICE
17185/udp closed wdbrpc
MAC Address: 00:09:7A:44:15:DB (Louis Design Labs.)

Interesting ports on domain2 (192.168.1.104):
PORT      STATE  SERVICE
17185/udp closed wdbrpc

Nmap finished: 109 IP addresses (12 hosts up) scanned in 9.043 seconds
```

It looks like the Cisco 7940 (192.168.1.22), Polycom (192.168.1.27), and Zultys (192.168.1.57) might have that port enabled. All an attacker has to do is connect with the native VxWorks debugger to gain full administrative access to that device.

There's really no recourse that an end user can take in this case—unless the vendor closes this gaping hole themselves with a patch or update.

NOTE The VoIP service enumeration examples covered in this chapter span the wealth of other support services that may exist in a VoIP network. Rather than reinvent the wheel, we recommend picking up a copy of *Hacking Exposed, Fifth Edition* by Stuart McClure, Joel Scambray, and George Kurtz (McGraw-Hill, 2005), which covers other enumeration examples not specific to VoIP that may be useful in your enumeration efforts.

SUMMARY

Information gathering is one of the most powerful tools at a hacker's disposal. As you have seen, there is no lack of sensitive information to be had through enumeration. Fortunately, it's also a tool you can use to harden your network against many of the simple techniques outlined in this chapter. Here are some general tenets to follow when configuring the phones, servers, and networking equipment on your network:

- Restrict access to as many administrative services as possible through firewall rules and switch VLAN segmentation.
- Change default administrative passwords, community strings, and usernames (if applicable) to mitigate brute-force attacks.
- Turn off as many services as possible to avoid extraneous information leakage.
- Perform regular security sweeps using automated and manual scans.
- Deploy VoIP-aware firewalls and intrusion prevention systems to detect many of the reconnaissance attacks outlined in this chapter.

REFERENCES

- Aharoni, Matti. "SNMP Enumeration and Hacking." *SecurityProNews*. September 9, 2003. http://www.securitypronews.com/securitypronews-24-20030909SNMPEnumerationandHacking.html
- Arkin, Ofir. "The Trivial Cisco IP Phones Compromise." September 2002. http://www.sys-security.com/archive/papers/The_Trivial_Cisco_IP_Phones_Compromise.pdf
- GDB and VxWorks. http://developer.apple.com/documentation/DeveloperTools/gdb/gdb/gdb_19.html#SEC164
- Merdinger, Shawn. "VoIP WiFi Phone Handset Security Analysis." Shmoocon 2006. http://www.shmoocon.org/2006/presentations/shmoocon_preso_voip_wifi_phone_merdinger.pdf
- RTP News. http://www.cs.columbia.edu/~hgs/rtp/
- Sinnreich, Henry, Alan B. Johnston, Robert J. Sparks, and Vinton G. Cerf. *SIP Beyond VoIP: The Next Step in IP Communications*. Melville: VON Publishing LLC, 2005.
- SIP Resource Center. http://www.cs.columbia.edu/sip/
- *SiVuS User Guide*. http://www.vopsecurity.org/SiVuS-User-Doc.pdf