Scapy

Experimentieren mit Netzwerkpaketen

Dirk Loss

pyCologne, 2010-01-13

- Wer kennt

Wireshark?

Was ist Scapy?

- Sniffer
- Paketgenerator
  - ähnlich hping2, nemesis, ...

- Python-Library + interaktiv nutzbar


```
>>> _
```

Wozu?

- Experimentieren mit Netzwerkprotokollen auf Paketebene
  - Sicherheitstests
  - TCP/IP lernen in der Praxis

Analyse des Skype Protokolls
[Biondi, Desclaux 2006]

heise online

Sie sind Gast · Einloggen | Registrieren

Home Newsticker 7-Tage-News News-Archiv Leserforum Newsletter News mobil

heise online > News > 2006 > KW 12 > Skype unter die Lupe genommen

21.03.2006 14:44 « Vorige | Nächste »

Skype unter die Lupe genommen

vorlesen / MP3-Download

Aufgrund seiner Einfachheit und Flexibilität hat der kostenlose Voice-over-IP-Dienst Skype recht weite Verbreitung gefunden – nicht zuletzt, weil Skype-Clients auch ohne Umkonfiguration der Firewall von außen erreichbar sind. Sicherheitsspezialisten stellen sich ob solcher Fähigkeiten allerdings die Nackenhaare auf. Da der Hersteller das proprietäre Skype-Protokoll und die genaue Funktionsweise des Clients nicht offengelegt hat, bleibt die Frage, was Skype sonst noch so alles kann und welche Risiken der Einsatz etwa im Unternehmensfeld birgt. Insbesondere wird gerne darüber spekuliert, ob Skype irgendeine Backdoor enthält.

Seit längerem gibt es Versuche, die Arbeitsweise von Skype zu analysieren und zu dokumentieren. Erste Ergebnisse (PDF-Datei) lieferte bereits 2004 die Universität Columbia, die aber hauptsächlich den Netzwerkverkehr untersuchte. Philippe Biondi und Fabrice Desclaux von EADS haben kürzlich auf der Black-Hat-Konferenz nachgelegt und die genauere Arbeitsweise des Clients veröffentlicht.

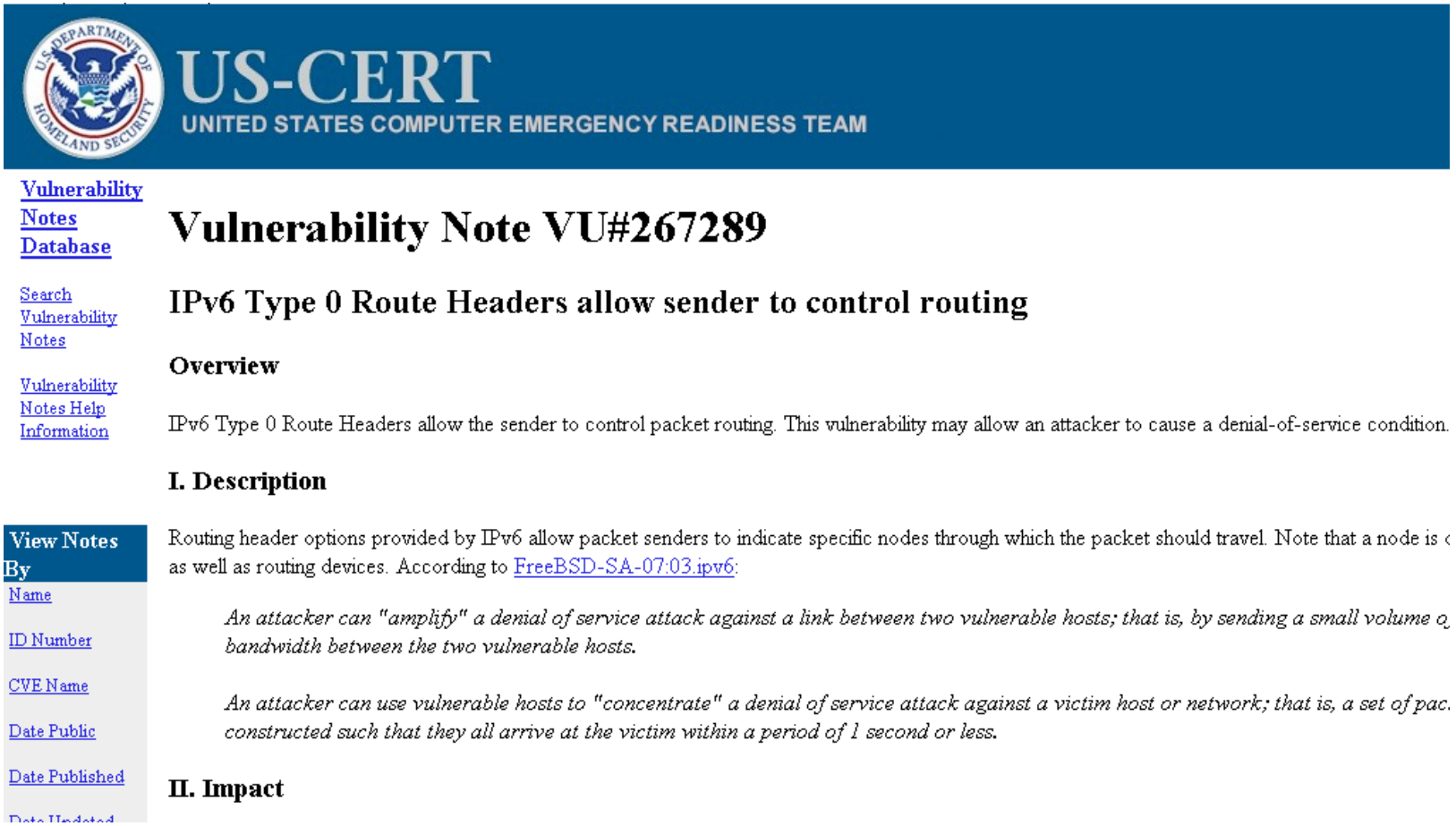
heise Security
Der Update-Check
findet und aktualisiert veraltete Programme mit bekannten Sicherheitslücken in wenigen Minuten.

heise resale Preisradar
Der Wunschmonitor hat 24" Diagonale und Full-HD-Auflösung
Monitore im 16:9-Format dominieren das Angebot - eine Entwicklung, die sich auch beim Kauf- und Nachfrageverhalten der Kunden niederschlägt.

c't 2/2010
Teamwork im Netz
Weitere Themen: Intels Turbo-Doppelke Windows Smartphones, Google App Engine, Security-CD mit Knoppicillin-Nachfolger Desinfec't

http://www.heise.de/newsticker/meldung/Skype-unter-die-Lupe-genommen-111559.html

IPv6 Type 0 Routing Header DoS
[Biondi, Ebalard 2007]



US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TEAM

Vulnerability Notes Database

Search Vulnerability Notes

Vulnerability Notes Help Information

View Notes By
Name
ID Number
CVE Name
Date Public
Date Published

Vulnerability Note VU#267289

IPv6 Type 0 Route Headers allow sender to control routing

Overview

IPv6 Type 0 Route Headers allow the sender to control packet routing. This vulnerability may allow an attacker to cause a denial-of-service condition.

I. Description

Routing header options provided by IPv6 allow packet senders to indicate specific nodes through which the packet should travel. Note that a node is
as well as routing devices. According to FreeBSD-SA-07:03.ipv6:

> *An attacker can "amplify" a denial of service attack against a link between two vulnerable hosts; that is, by sending a small volume o*
> *bandwidth between the two vulnerable hosts.*
>
> *An attacker can use vulnerable hosts to "concentrate" a denial of service attack against a victim host or network; that is, a set of pac*
> *constructed such that they all arrive at the victim within a period of 1 second or less.*

II. Impact

https://www.kb.cert.org/vuls/id/267289

Grundfunktionen

Demo

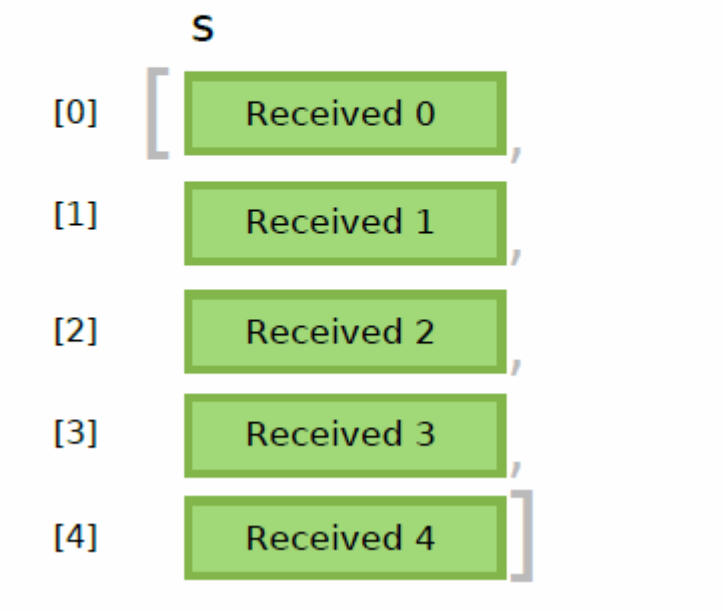
```
>>> s=IP(dst="google.com")/ICMP()
>>> s.show()
###[ IP ]###
  version= 4
  ihl= 0
  tos= 0x0
  len= 0
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= 0x0
  src= 192.168.1.10
  dst= Net('google.com')
  options= ''
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= 0x0
     id= 0x0
     seq= 0x0
>>> sniff(timeout=10)
<Sniffed: TCP:42 UDP:2 ICMP:8 Other:0>
>>> _
```

Bauen

- Objekte instanziieren
  - IP(dst='192.168.1.1')
- Verschachteln
  - Ether()/IP()/UDP()/DNS()
- Scapy wandelt in Strings um
  - Default Werte werden einsetzt, Layer für Layer
  - Checksummen, Längen, etc. werden berechnet

Sniffen und Dekodieren

```
>>> s = sniff(count=5)
```



Sniffen und Dekodieren

- sniff()
- Empfangene Bytes werden als String an Objekt-Konstruktoren übergeben
  - Ether('\x00\x1cJ\x1d\xad\xc9\x00"\xfb%\x14\xf6\x08\x00E\x00\x00\x14\x00\x01\x00\x00@\x00z\xc5\xc0\xa8\xb2+\xd8"\xb5-‘)
- Überschüssige Bytes sind Payload
  - Wird rekursiv ebenfalls dekodiert
  - Automatische Ermittlung des Objekttyps anhand des darüberliegenden Layers

Senden

- Intern grundsätzlich auf Layer 2
  - PF_PACKET oder libdnet
- Bei Senden auf Layer 3 generiert Scapy den Layer 2 selbst
  - Auswahl des Netzwerkinterfaces
    - conf.route
  - ARP Anfrage
    - conf.netcache.arpcache

Funktionen zum Senden

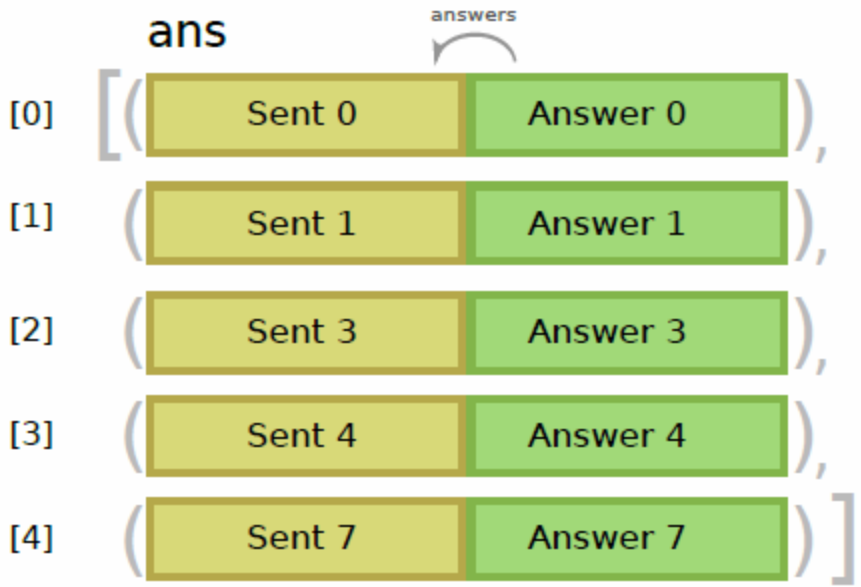
	Layer 2	Layer 3
Senden	sendp()	send()
Senden/Empfangen	srp()	sr()
Senden/ Erste Antwort empfangen	srp1()	sr1()
Loop	srploop()	srloop()
Flooding	srpflood()	srflood()

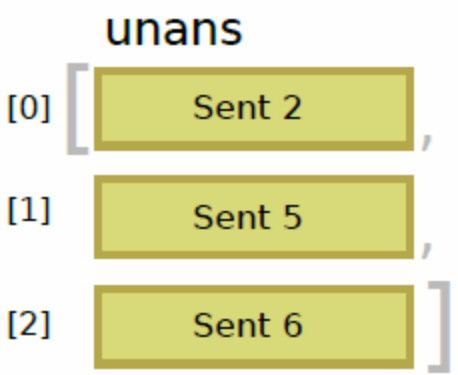
sr1()

```
>>> sr1( IP(dst="192.168.8.1")/ICMP() )
Begin emission:
..Finished to send 1 packets.
.*
Received 4 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=28 id=46681 flags= frag=0L
 ttl=64 proto=ICMP chksum=0x3328 src=192.168.8.1
 dst=192.168.8.14 options='' |<ICMP type=echo-reply code=0
 chksum=0xffff id=0x0 seq=0x0 |<Padding load='\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x91\xf49\xea' |>>>
```

Rückgabewerte von sr()

```
>>> ans, unans = sr(eightpackets)
```






```
>>> ans[2]
(Sent 3, Answer 3)
>>> ans[2][1]
Answer 3
>>> unans[2]
Sent 6
```

High-Level Funktionen

- arping()
- traceroute()
- get_mac_by_ip()
- tshark()
- …

traceroute()

```
>>> ans,unans=traceroute(["www.apple.com","www.cisco.com","www.microsoft.com"])
Received 90 packets, got 90 answers, remaining 0 packets
   17.112.152.32:tcp80 198.133.219.25:tcp80 207.46.19.30:tcp80
1  172.16.15.254    11  172.16.15.254    11   172.16.15.254    11
2  172.16.16.1      11  172.16.16.1      11   172.16.16.1      11
[...]
11 212.187.128.57   11  212.187.128.57   11   212.187.128.46   11
12 4.68.128.106     11  4.68.128.106     11   4.68.128.102     11
13 4.68.97.5        11  64.159.1.130     11   209.247.10.133   11
14 4.68.127.6       11  4.68.123.73      11   209.247.9.50     11
15 12.122.80.22     11  4.0.26.14        11   63.211.220.82    11
16 12.122.10.2      11  128.107.239.53   11   207.46.40.129    11
17 12.122.10.6      11  128.107.224.69   11   207.46.35.150    11
18 12.122.2.245     11  198.133.219.25   SA   207.46.37.26     11
19 12.124.34.38     11  198.133.219.25   SA   64.4.63.70       11
20 17.112.8.11      11  198.133.219.25   SA   64.4.62.130      11
21 17.112.152.32    SA  198.133.219.25   SA   207.46.19.30     SA
[...]
```

arping()

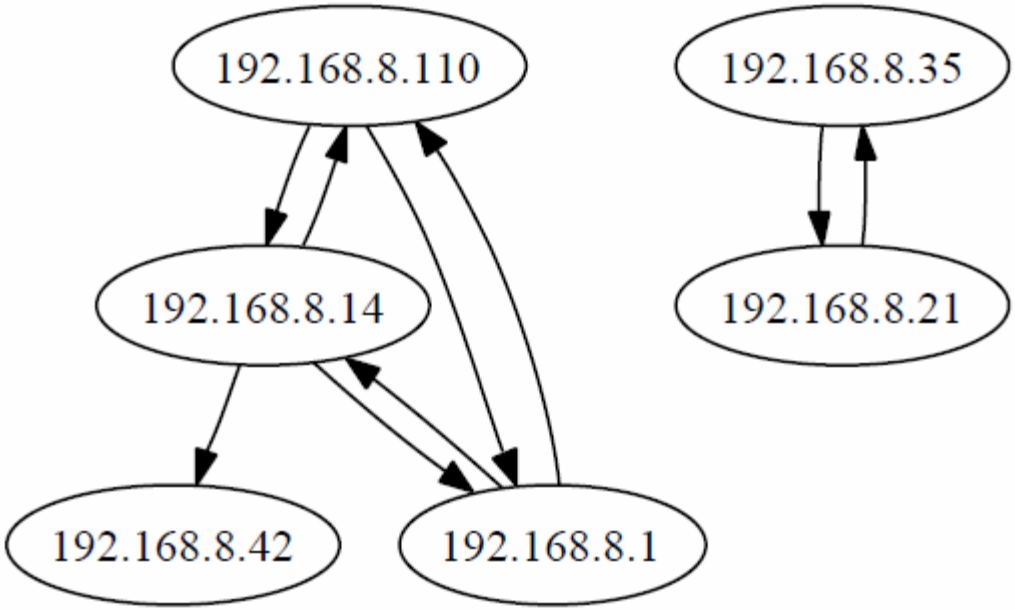
```
>>> arping("172.16.15.0/24")
Begin emission:
*Finished to send 256 packets.
*
Received 2 packets, got 2 answers, remaining 254 packets
00:12:3f:0a:84:5a 172.16.15.64
00:12:79:3d:a3:6a 172.16.15.254
(< ARPing: UDP:0 TCP:0 ICMP:0 Other:2>,
 < Unanswered: UDP:0 TCP:0 ICMP:0 Other:254>)
```

High-Level Funktionen
auf Paketen/Paketlisten

- p.conversations()
- p.plot()
- p.pdfdump()
- p.make_table()
- …

.conversations()

```
>>> a = sniff()
>>> a.conversations()
```



Werkzeuge

- hexdump(), hexedit()
- rdpacp(), wrpcap()
- fragment()
- fuzz()
- …

Fortgeschrittene Features

- IPv6
- SNMP und ASN.1

```
>>> a=SNMP(version=3, PDU=SNMPget(varbindlist=[SNMPvarbind(oid="1.2.3",value=5),
...                                            SNMPvarbind(oid="3.2.1",value="hello")]))
>>> a.show()
###[ SNMP ]###
  version= v3
  community= 'public'
  \PDU\
   |###[ SNMPget ]###
   |  id= 0
   |  error= no_error
   |  error_index= 0
   |  \varbindlist\
   |   |###[ SNMPvarbind ]###
   |   |  oid= '1.2.3'
   |   |  value= 5
   |   |###[ SNMPvarbind ]###
   |   |  oid= '3.2.1'
   |   |  value= 'hello'
```

- Automaten

Internes

- PacketList
- Packet
- Layer
- Field

Aufbau eines IP Pakets

Example : Default Values for IP

```
>>> ls(IP)
version    : BitField             = (4)
ihl        : BitField             = (None)
tos        : XByteField           = (0)
len        : ShortField           = (None)
id         : ShortField           = (1)
flags      : FlagsField           = (0)
frag       : BitField             = (0)
ttl        : ByteField            = (64)
proto      : ByteEnumField        = (0)
chksum     : XShortField          = (None)
src        : Emph                 = (None)
dst        : Emph                 = ('127.0.0.1')
options    : IPoptionsField       = ('')
```

Eigene Layer implementieren

Example

```
class Test(Packet):
    name = "Test protocol"
    fields_desc = [
        ByteField("field1", 1),
        XShortField("field2", 2),
        IntEnumField("field3", 3, { 1:"one", 10:"ten" }),
    ]
```

[PacSec05]

Nachteile

- Geringe Performance
- TCP basierte Protokolle umständlich
  - Kein Zusammensetzen von TCP-Streams
  - TCP/IP Stack weiß nichts von Scapy
- Relativ wenige Layer implementiert
  - zumindest im Vergleich mit Wireshark...

Das Projekt

- Philippe Biondi

- Mercurial Repositories
- Trac für Bugtracking und Wiki
- Mailingliste

Quellcode

- GNU GPL v2
- ~28.000 Zeilen Python Code
  - inkl. Kommentare, etc.
- Seit Scapy v2 aufgeteilt in mehrere Module
  - Problem: Zirkuläre Imports

Scapy und ich...

- Nutzer seit ~2005
- Windows Port, seit 2007
- Dokumentation
- OSPF Extension
- Bug fixes

Fazit

- Flexibles Werkzeug zum Manipulieren von Netzwerkpaketen
  - Für Security Tests oder zum Lernen
  - Ersetzt viele Spezialtools
- Python als Domain Specific Language
  - Elegante Definition neuer Layer
  - Integration in eigene Programme

Vielen Dank für die Aufmerksamkeit!

- Dirk Loss

- mail at dirk-loss de
- http://dirk-loss.de
- http://twitter.com/dloss

Links

- Scapy-Homepage
  - http://www.secdev.org/projects/scapy/
- Wiki, Mercurial-Repository
  - http://trac.secdev.org/scapy
  - http://hg.secdev.org/scapy/
- Mailingliste
  - http://news.gmane.org/gmane.comp.security.scapy.general
  - scapy.ml-subscribe at secdev org

Quellen

- [PacSec05]:

  Philippe Biondi:
  Network packet forgery with Scapy
  http://www.secdev.org/conf/scapy_pacsec05.pdf