

Challenge: [Acoustic Lab](#)

Platform: CyberDefenders

Category: Network Forensics

Difficulty: Medium

Tools Used: Wireshark, Zui

Summary: This lab involved investigating a PCAP and logs from a VoIP honeypot. It was my first experience working with VoIP protocols; however, I still found it fun and relatively easy.

Scenario: This lab takes you into the world of voice communications on the internet. VoIP is becoming the de-facto standard for voice communication. As this technology becomes more common, malicious parties have more opportunities and stronger motives to control these systems to conduct nefarious activities. This challenge was designed to examine and explore some of the attributes of the SIP and RTP protocols.

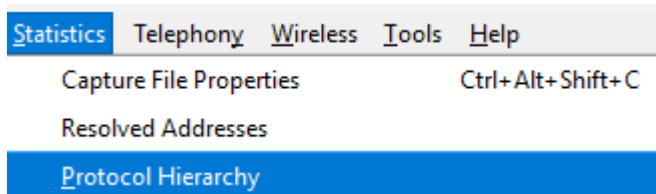
Lab Files:

- "[log.txt](#)" was generated from an unadvertised, passive honeypot located on the internet such that any traffic destined to it must be nefarious. Unknown parties scanned the honeypot with a range of tools, and this activity is represented in the log file.
 - The IP address of the honeypot has been changed to "honey.pot.IP.removed". In terms of geolocation, pick your favorite city.
 - The MD5 hash in the authorization digest is replaced with "MD5_hash_removedXXXXXXXXXXXXXXXXXX"
 - Some octets of external IP addresses have been replaced with an "X"
 - Several trailing digits of phone numbers have been replaced with an "X"
 - Assume the timestamps in the log files are UTC.
- "[Voip-trace.pcap](#)" was created by honeynet members for this forensic challenge to allow participants to employ network analysis skills in the VOIP context.

As a soc analyst, analyze the artifacts and answer the questions.

What is the transport protocol being used?

To identify the transport protocol being used, let's explore the PCAP using Wireshark and navigate to Statistics > Protocol Hierarchy:



This shows us the protocol distribution within the PCAP. As you can see in the image below, the VOIP related protocols are Session Initiation Protocol (SIP) and Real-Time Transport Protocol (RTP):

| Protocol | Percent Packets | Packets | Percent Bytes | Bytes |
|--------------------------------------|-----------------|---------|---------------|---------|
| ▼ Frame | 100.0 | 4447 | 100.0 | 1117758 |
| ▼ Ethernet | 100.0 | 4447 | 5.6 | 62258 |
| ▼ Internet Protocol Version 4 | 100.0 | 4447 | 8.0 | 88940 |
| ▼ User Datagram Protocol | 70.9 | 3154 | 2.3 | 25232 |
| Session Initiation Protocol | 0.4 | 19 | 1.0 | 11173 |
| ▼ Real-Time Transport Protocol | 70.0 | 3113 | 46.2 | 515888 |
| RFC 2833 RTP Event | 2.8 | 125 | 0.0 | 500 |
| Real-time Transport Control Protocol | 0.5 | 21 | 0.3 | 2852 |
| Data | 0.0 | 1 | 0.0 | 4 |

My understand of VOIP as a whole, and its protocols, is limited. From my minimal research, SIP is used to establish, manage, and terminate sessions like voice or video calls, whereas RTP handles the actual transmission of the real-time audio and video data over IP networks. As you can also see, both protocols use UDP, therefore the transport protocol being used is UDP.

Answer: UDP

The attacker used a bunch of scanning tools that belong to the same suite. Provide the name of the suite

I started by investigating the user-agent strings within the PCAP by using Zui and the following query:

- `_path=="http" | count() by user_agent | sort -r count`

| user_agent | count |
|--|-------|
| Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.9) Gecko/20100401 Ubuntu/9.10 (karmic) Firefox/3.5.9 | 92 |
| null | 20 |
| Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.19041.4291 | 4 |
| Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET | 1 |

This unfortunately does not return anything indicative of a scanning tool, the only thing that stands out is the PowerShell user-agent. Let's direct our focus to the log file and see what we can find. If you view the log file using a tool like cat piped to less, or something simple like notepad, we can see a user-agent field:

```
Source: 210.184.X.Y:1083
Datetime: 2010-05-02 01:43:05.606584

Message:

OPTIONS sip:100@honey.pot.IP.removed SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:5061;branch=z9hG4bK-2159139916;rport
Content-Length: 0
From: "sipvicious"<sip:100@1.1.1.1>; tag=X_removed
Accept: application/sdp
User-Agent: friendly-scanner
To: "sipvicious"<sip:100@1.1.1.1>
Contact: sip:100@127.0.0.1:5061
CSeq: 1 OPTIONS
Call-ID: 845752980453913316694142
Max-Forwards: 70
```

Immediately, the string “friendly-scanner” seems suspicious, but let’s explore all user-agents within this log file. Using the following command:

- `cat log.txt | grep "User-Agent:" | sort | uniq -c | sort -r`

We can see that “friendly-scanner” appears 4248 times within the log file:

```
4248 User-Agent: friendly-scanner
18 User-Agent: Zoiper rev.6751
```

Upon a quick Google search, we can see that Friendly-scanner is a tool that scans for SIP servers, if the port is open it attempts to brute-force the SIP server via testing sequential SIP account numbers with common credentials. You can also see that it’s also referred to as SIPVicious:


Looking for a free security toolset to test your SIP infrastructure?

SIPVicious OSS has been around since 2007 and is actively updated to help security teams, QA and developers test SIP-based VoIP systems and applications.

[Releases](#)[Source code](#)[Documentation](#)

Open-source security suite for auditing SIP based VoIP systems

Also known as **friendly-scanner**, it is freely available to help pentesters, security teams and developers quickly test their SIP systems.



Answer: SIPVicious

What is the User-Agent of the victim system?

To find the user-agent of the victim system, we can filter for SIP packets within Wireshark by using the following display filter:

- sip

If you apply the user-agent field as a column, we can see the user agent of the victim systems appears to be “Asterisk PBX 1.6.0.10-FONCORE-r40”:

| Info | User-Agent |
|---|--|
| Request: OPTIONS sip:100@172.25.105.40 | UNfriendly-scanner - for demo purposes |
| Status: 200 OK (OPTIONS) | Asterisk PBX 1.6.0.10-FONCORE-r40 |
| Request: REGISTER sip:172.25.105.40 (1 binding) | X-Lite Beta release 4.0 Beta 2 stamp 56233 |
| Status: 401 Unauthorized | Asterisk PBX 1.6.0.10-FONCORE-r40 |
| Request: REGISTER sip:172.25.105.40 (1 binding) | X-Lite Beta release 4.0 Beta 2 stamp 56233 |
| Status: 200 OK (REGISTER) (1 binding) | Asterisk PBX 1.6.0.10-FONCORE-r40 |
| Request: SUBSCRIBE sip:555@172.25.105.40 | X-Lite Beta release 4.0 Beta 2 stamp 56233 |
| Status: 401 Unauthorized | Asterisk PBX 1.6.0.10-FONCORE-r40 |
| Request: SUBSCRIBE sip:555@172.25.105.40 | X-Lite Beta release 4.0 Beta 2 stamp 56233 |
| Status: 404 Not found (no mailbox) | Asterisk PBX 1.6.0.10-FONCORE-r40 |
| Request: INVITE sip:1000@172.25.105.40 | X-Lite Beta release 4.0 Beta 2 stamp 56233 |
| Status: 401 Unauthorized | Asterisk PBX 1.6.0.10-FONCORE-r40 |
| Request: ACK sip:1000@172.25.105.40 | |
| Request: INVITE sip:1000@172.25.105.40 | X-Lite Beta release 4.0 Beta 2 stamp 56233 |
| Status: 100 Trying | Asterisk PBX 1.6.0.10-FONCORE-r40 |
| Status: 200 OK (INVITE) | Asterisk PBX 1.6.0.10-FONCORE-r40 |
| Request: ACK sip:1000@172.25.105.40 | X-Lite Beta release 4.0 Beta 2 stamp 56233 |
| Request: BYE sip:1000@172.25.105.40 | X-Lite Beta release 4.0 Beta 2 stamp 56233 |
| Status: 200 OK (BYE) | Asterisk PBX 1.6.0.10-FONCORE-r40 |

If you look at the info column, we can see a scanning attempt with the user-agent “UNFriendly-scanner – for demo purposes”. The softphone client (threat actor/tester) messages came from “X-Lite Beta release 4.0 Beta 2 stamp 56233”. The system that consistently replies identifies itself as “Asterisk PBX 1.6.0.10-FONCORE-r40”.

Answer: Asterisk PBX 1.6.0.10-FONCORE-r40

Which tool was only used against the following extensions: 100,101,102,103, and 111?

The SIPVicious suite consists of the following tools:

- svmap
- svwar
- svcrack
- svreport, and
- svcrash

If you look through the [GitHub](#) of SIPVicious, we can see that svcrack.py is a password cracker.

Answer: svcrack.py

Which extension on the honeypot does NOT require authentication?

To identify what extension does not require authentication, we need to look for SIP requests that lack an authorisation header in the logs. A normal SIP request would look something like as follows:

```
Source: 210.184.X.Y:5209
Datetime: 2010-05-02 01:50:16.837204

Message:

REGISTER sip:honey.pot.IP.removed SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:5089;branch=z9hG4bK-3968602124;rport
Content-Length: 0
From: "102" <sip:102@honey.pot.IP.removed>; tag=X_removed
Accept: application/sdp
User-Agent: friendly-scanner
To: "102" <sip:102@honey.pot.IP.removed>
Contact: sip:123@1.1.1.1
CSeq: 2 REGISTER
Call-ID: 1670060393
Max-Forwards: 70
Authorization: Digest username="102",realm="localhost",nonce="3645318188",uri="sip:honey.pot.IP.removed",response="MD5_hash_removedXXXXXXXXXXXXXXXXXX",algorithm=MD5
```

If you look through the logs, we can see that extension 100 does not contain an authorisation field:

```
Source: 210.184.X.Y:1083
Datetime: 2010-05-02 01:43:05.606584

Message:

OPTIONS sip:100@honey.pot.IP.removed SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:5061;branch=z9hG4bK-2159139916;rport
Content-Length: 0
From: "sipvicious"<sip:100@1.1.1.1>; tag=X_removed
Accept: application/sdp
User-Agent: friendly-scanner
To: "sipvicious"<sip:100@1.1.1.1>
Contact: sip:100@127.0.0.1:5061
CSeq: 1 OPTIONS
Call-ID: 845752980453913316694142
Max-Forwards: 70
```

Answer: 100

How many extensions were scanned in total?

To determine the answer, all we need to do is count the distinct To: SIP extensions only in messages where the user agent is the scanner. We can do so by using the following command:

- `awk '/User-Agent: friendly-scanner/{inblock=1}/To:/{if(inblock){ if (match($0,/sip:([^\@>]+)/,m)) {print m[1]} }/^$/{inblock=0}} log.txt | sort -u | wc -l`

In all honesty, ChatGPT generated this command, but it produced the correct output.

```
awk '/User-Agent: friendly-scanner/{inblock=1}/To:/{if(inblock){ if (match($0,/sip:([^\@>]+)/,m)) {print m[1]} }/^$/{inblock=0}} log.txt | sort -u | wc -l
2652
```

Answer: 2652

There is a trace for a real SIP client. What is the corresponding user-agent? (two words, once space in between)

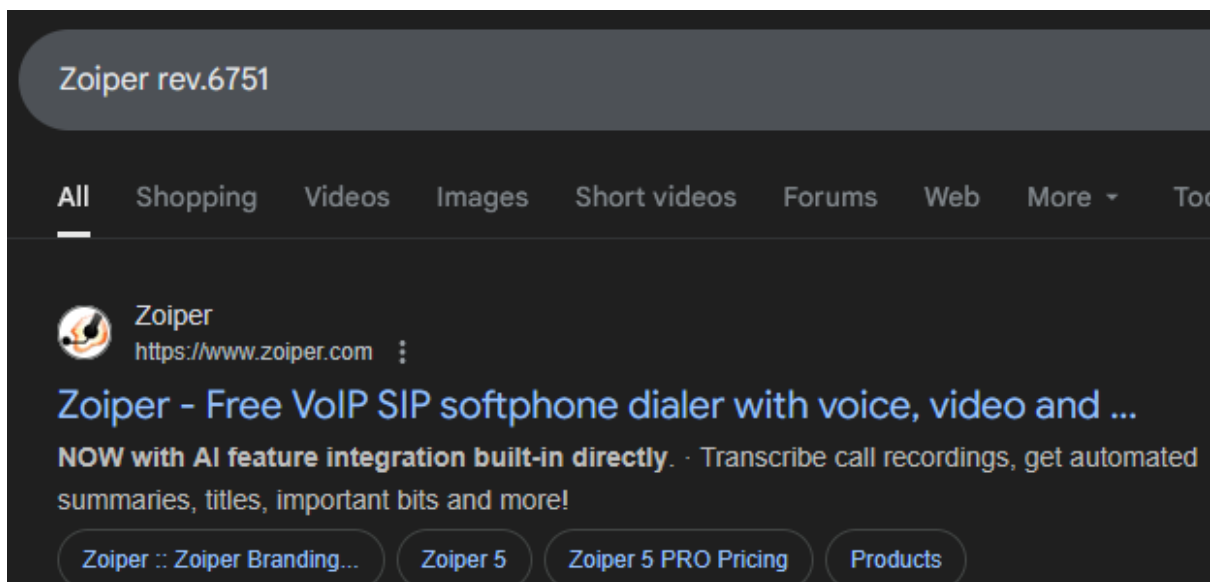
Using the following command:

- `grep "User-Agent:" log.txt | sort | uniq -c | sort -r`

We can see that there is only two user-agents within the log file, one being the scanner, and the other being a real SIP client:

```
4248 User-Agent: friendly-scanner
18 User-Agent: Zoiper rev.6751
```

If you search for this user-agent, we can see references to free VoIP softphone dialers:



Answer: Zoiper rev.6751

Multiple real-world phone numbers were dialed. What was the most recent 11-digit number dialed from extension 101?

To extract 11-digit numbers dialed from extension 101, we can use the grep command provided in the hint:

- `grep -Ei 'From: "Unknown"<sip:101' log.txt -B8 | grep INVITE`

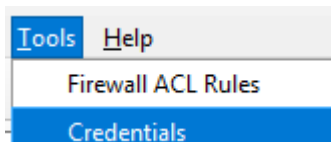
```
INVITE sip:900114382089XXXX@honey.pot.IP.removed;transport=UDP SIP/2.0
INVITE sip:00112322228XXXX@honey.pot.IP.removed;transport=UDP SIP/2.0
INVITE sip:00112524021XXXX@honey.pot.IP.removed;transport=UDP SIP/2.0
```

The most recent number is 00112524021.

Answer: 00112524021

What are the default credentials used in the attempted basic authentication? (format is username:password)

Wireshark has a semi useful feature that attempts to extract credentials found within the PCAP. It's located at Tools > Credentials:



Here we can see a bunch of basic auth attempts with the same username “maint”:

| Packet N ^ | Protocol | Username | Additional Info |
|---------------------|-----------------|-----------------------|-----------------|
| 60 | HTTP basic auth | maint | |
| 71 | HTTP basic auth | maint | |
| 115 | HTTP basic auth | maint | |
| 119 | HTTP basic auth | maint | |
| 124 | HTTP basic auth | maint | |
| 129 | HTTP basic auth | maint | |
| 134 | HTTP basic auth | maint | |
| 164 | HTTP basic auth | maint | |
| 171 | HTTP basic auth | maint | |
| 176 | HTTP basic auth | maint | |
| 181 | HTTP basic auth | maint | |
| 189 | HTTP basic auth | maint | |
| 214 | HTTP basic auth | maint | |
| 222 | HTTP basic auth | maint | |
| 227 | HTTP basic auth | maint | |

If you view any one of the packets, we can see the password within the authorisation field in the packet details pane:

```
GET /maint HTTP/1.1\r\n
Host: 172.25.105.40\r\n
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.9) Gecko/20100401 Ubuntu/9.10 (karmic)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
Referer: http://172.25.105.40/user/\r\n
Cookie: lng=en; PHPSESSID=gl8n9gi8r091ke1vb0hpf97na1\r\n
Authorization: Basic bWFpbmQ6cGFzc3dvcmQ=\r\n
Credentials: maint:password
```

Answer: maint:password

Which codec does the RTP stream use? (3 words, 2 spaces in between)

Using the following display filter in Wireshark enables us to examine RTP traffic:

- rtp

If you view the packet details pane, we can see the payload type:

Payload type: ITU-T G.711 PCMU (0)

Answer: ITU-T G.711 PCMU

How long is the sampling time (in milliseconds)?

Through using my best friend ChatGPT, we can determine that the sampling time in milliseconds is 0.125.

Answer: 0.125

What was the password for the account with username 555?

Using the following filter, we can look for HTTP packets that contain the string “username”:

- http matches "username"

In packet number 1287, if you follow its HTTP stream, we can see what appears to be some sort of config file. Near the bottom we can find the password for user “555”:

```
type=friend
username=555
secret=1234
host=dynamic
extension=from-trunk
context=from-trunk
```

Answer: 1234

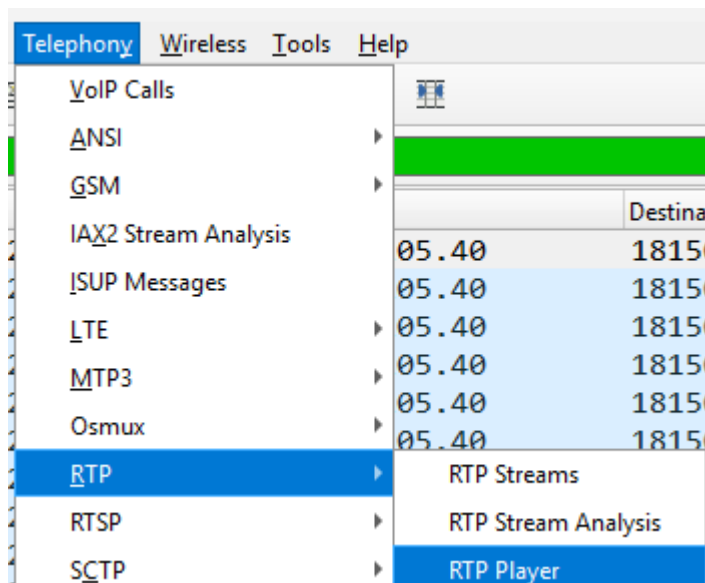
Which RTP packet header field can be used to reorder out of sync RTP packets in the correct sequence?

The Timestamp header field can be used to reorder out of sync RTP packets.

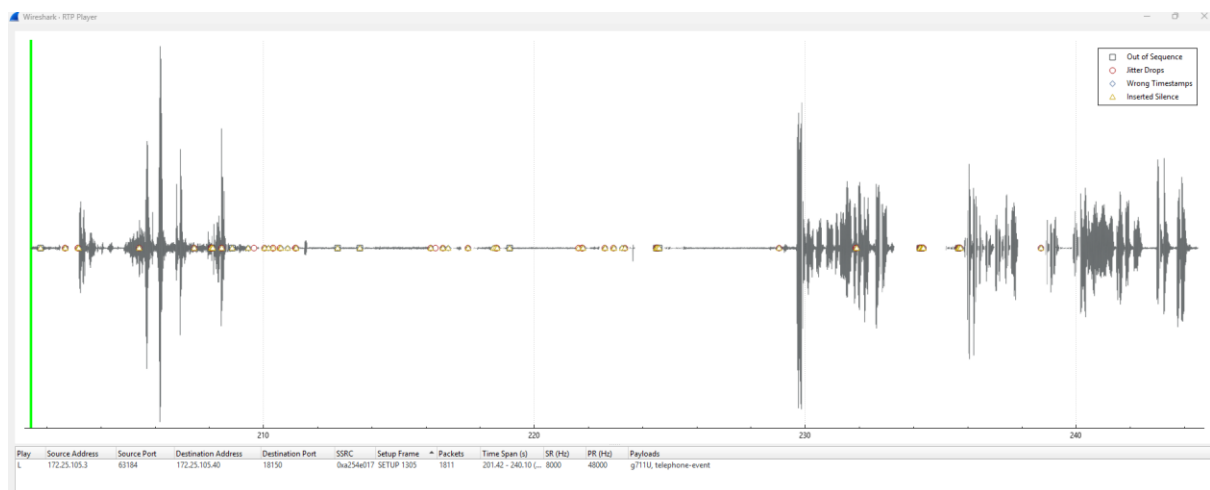
Answer: timestamp

The trace includes a secret hidden message. Can you hear it?

Wireshark has tool that enables us to listen to RTP traffic. To listen, navigate to Telephony > RTP > RTP Player:



Here we can find one conversation between 172.25.105.3 and 172.25.105.40:



After clicking the play button, the secret message is said near the end of the call.

Answer: mexico