Analysis and Signature of Skype VoIP Session Traffic

Fraunhofer FOKUS Technical Report NGNI-SKYPE-06b

Sven Ehlert, Sandrine Petgang
Fraunhofer FOKUS, Berlin, Germany
{ehlert, petgang}@fokus.fraunhofer.de

July 25th, 2006

Abstract

Skype is a peer-to-peer VoIP application that has gained substantial popularity since its launch in 2003. However, none of Skype's algorithms or its protocol specification are available for public inspection, which impedes evaluation from a security perspective.

In this report we present an analysis of Skype operation from the network point of view. From the analysis we develop traffic signatures that allow a third party monitoring entity to detect the usage of the Skype application. These signatures concentrate on Skype signalling traffic and contain different characteristics, including port usage, network packet sizes and payload content. The application of theses signatures in a detection tool shows their effectiveness to properly detect Skype versions 1.4 and 2.0, and 2.5 traffic.

1 Introduction

Voice over IP (VoIP) has become rapidly popular in recent times. Several international providers are already offering free or low cost VoIP services, like Google [1], Yahoo [2], Skype [3], or Gizmo Project [4]. As many business and governmental organisations are considering or have deployed VoIP communication to extend or even replace POTS communication, security consideration are becoming more and more crucial. For example, it is mandatory that the VoIP service does only expose information the user is willing to reveal and that confidential information cannot be intercepted by third parties.

While many providers use the IETF Session Initiation Protocol (SIP) [5] (Yahoo, Gizmo Project; Google Talk is currently considering SIP), the popular VoIP service Skype uses a proprietary signalling and media protocol for voice calling, instant messaging, audio conferencing and file transfers. Additionally, all traffic is end-to-end encrypted. Thus, it is currently not known what exact information is transported with the application. It also impedes protocol analysis for potential security holes.

Much of Skype's success results from their concept of delivering user friendly operation, e.g. the client can operate without manual user configuration. However, this user friendliness is largely due to Skype's ability to detect the current network configuration (e.g. restrictions resulting from Network Address Translators (NAT) or firewalls) and deliver mechanisms to circumvent many applied network security restrictions.

Monitoring Skype operation can thus show potential security breaches in the network and can help to assess current security policies. Traffic monitoring for security reasons is widely used in Intrusion Detection Systems [6],[7].

In this report we examine Skype from a network point of view. We analyse network traffic with the goal to detect patterns that are intrinsic to the Skype protocol. This information can then be fed into a third party network monitoring tool, allowing a security operator to monitor and detect Skype traffic.

Detecting Skype traffic has not yet been widely covered in the research community, which is mainly on account of Skype's closed operation model. Suh et al. [8] monitor Skype traffic using relay nodes. Instead of examining payload content they use heuristics and statistical analysis to detect Skype traffic. Guha et al. [9] measure combined Skype signalling and media traffic and present a Skype traffic model and compare it to other peer-to-peer applications.

Baset and Schulzrinne [10] present a broad analysis of Skype operation mechanisms, including basic Skype message flows. Their analysis is based on older versions of the Skype protocol up to version 1.4. The protocol has significantly changed with version 2.0. They also focus on general Skype operation without presenting a detection signature. Fabrice [11] recently presented first results in his attempt to reverse engineer Skype. He outlines possibilities to debug the Skype binary application and shows initial results in decrypting Skype's protocol operation. Other topics of Skype research include Skype performance measurements [12], [13].

This report is organized as follows: In Sect. 2 we present an overview of Skype operation as far as it is currently known. We give an overview of our analysis method in Sect. 3, including a description of the utilized testbed. The analysis consists of two parts: In Sect. 4 we describe general Skype port usage analysis. As port analysis alone can not give sufficient detail to accurately describe Skype network traffic, we present in Sect. 5 an analysis of session message flows including payload patterns. Based on the gained data we develop in Sect. 6 a signature of Skype session traffic. We conclude our work in Sect. 7 and give an outlook of open monitoring issues. A list of used abbreviations is appended.

2 Skype Operation Overview

Baset [10] presents a broad overview of Skype operation and Skype network entities. We follow in this report his naming and abbreviation scheme.

Unlike client-server based SIP, Skype uses an overlay peer-to-peer (P2P) network, similar to its file sharing predecessor KaZaa [14]. There are three main components in the Skype network.

- The Skype *login server* (LS) is one of the the few central components of the network. Every user is authenticated through the login server to gain access to the network.

- A *Skype client* (SC) is a participating user in the network. SC provide all user functionality to access the network, that is login, initiate and receive calls, instant messages and file transfer.

- *Super Node* (SN). Any SC can also become a super node, which provide additional functionality to other SN and SC. A super node performs routing tasks such as forwarding requests to appropriate destinations and answer to queries from other SC or SN. SN also can forward login requests in case the login server is not directly reachable from a SC. Additionally, SN provide media proxying capabilities for other SC that have only restricted internet access, be it through Network Address Translation (NAT) or restricted firewalls. Every SC will automatically become a SN if it meets certain criteria, e.g. high speed and unrestricted internet access. It is generally difficult to prohibit a SN from becoming a SN. To log in to the network, a SC needs to contact at least one SN. Several SN are hard coded into the SC's executable. These *bootstrap SN* are contacted upon first launch of the client to gather an updated and more extensive list of currently available SN. Baset lists seven different bootstrap SN [10], which we can confirm in our experiments.

Except for some dedicated operations like authentication, user list storage or Skype-to-PSTN connectivity, there are no further central servers in the Skype network. All other operations, e.g. user searches or message forwarding are performed in a decentralized way by the SN. Figure 1 depicts the Skype network structure.

Both signalling and media traffic is encrypted for any kind of Skype connection. There is only one message containing plain text content after installation. Here the SC contacts the Skype web page to check for an updated version of the SC.

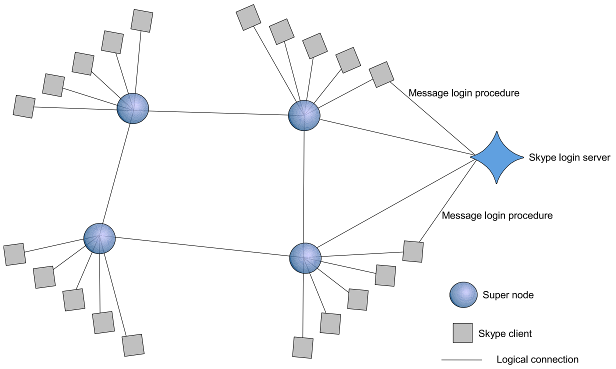

Figure 1: **Skype P2P network**. Skype clients are interconnected through special clients (Super nodes) that also provide additional routing and message forwarding features to the Skype network. Users need to contact a login server either directly or through a super node to gain access to the network.

3 Analysis Method and Testbed

To detect characteristics of Skype's network behaviour, we have captured and analysed multiple Skype session traces.

All test have been conducted on Windows XP computers with Skype versions 1.4 and 2.0 [1]. Hence, given results apply thus to all versions from version 1.4 on. If network traffic differs with later versions, additional remarks are given.

We have installed the Ethereal packet sniffer to record incoming and outgoing traffic during a Skype session, both at the caller and callee's side. Note, that in this report we solely analyse signalling traffic.

The exact operation of the Skype protocol is currently unknown. To determine accurate Skype network characteristics, we have modified network parameters at both nodes to gain a broader data set.

Firewall We applied three different settings: No firewall protection at all, UDP / TCP incoming port restrictions and both incoming and outgoing port restrictions. See Sec. 4 for relevant Skype port restrictions.

Network Address Translation We assigned public IP addresses without NAT and private IP addresses behind a full-cone NAT [15]. Caller and Callee have been both in the same and different networks.

We have varied these combinations between caller and callee to capture traces in different application of these parameters. The following operations have been conducted in every different network configuration:

- Creating a Skype account through the client interface.
- Logging in to the Skype network.
- Logging in to the Skype network with known contacts of other Skype users are stored already in Skype's "Buddy list". Two scenarios have been created, where users ("Buddies") are off-line or on-line during login.
- Login with different user names, and also with wrong credentials.
- Different call scenarios, e.g. call initiated from a user already on the contact list, calls to users having the caller blocked, conference calls, calls with different presence information set (Available, Do Not Disturb), callee does not respond, call into PSTN, etc.

The tests have been reiterated several times. For the reiteration, two different methods have been applied:

- Simply logging off from the Skype network and capturing a new trace after a following login.
- Complete deinstallation of the Skype client including removal of System registry entries (using Windows Restoration Points) followed by a fresh install of the client. This was conducted to distinguish if Skype uses some communication procedures only executed after a fresh login.

For the whole project more than 700 different traces have been captured and used for the analysis.

[1] Version 2.5 came out at the end of our study. We have evaluated our developed signature with version 2.5

The traces have been analysed with the goal to detect Skype intrinsic network features. As most of Skype's traffic is encrypted, our analysis focuses on available identifying features:

- protocol and port usage,
- packet size (denoted as s). It is always given as the message payload size, i.e. the content after the UDP / TCP header,
- packet content, when clear characteristics are discernible.

4 Skype Port Analysis

Skype uses for communication both UDP and TCP connections. As a fallback mechanism it has the ability to initiate connections to TCP ports 443 (HTTPS) and 80 (HTTP), in this order. To detect the used port range we have imposed several network port restrictions to force the SC to use different port combinations to login to the Skype network.

For UDP traffic, it generally uses an arbitrarily set port. The user can change this port number from SC's configuration dialogue. If set, the SC is reachable by this port for UDP (and incoming TCP) traffic. If the user does not specify a port, the SC selects arbitrarily one port which it uses during operation. The only fixed UDP port number is 33033, which is used to contact statically configured bootstrap SN from the SC. We could not devise a pattern for UDP port usage if this port is not selected by the user.

While Skype uses both UDP and TCP connections, UDP is not mandatory for its operation. In our experiments the SC was able to login even with only TCP traffic allowed.

For TCP the SC tries to contact other hosts at ports higher than 1024. If firewall restrictions for this port range are applied, it tries again to this host on port 443. If this also fails it will initiate a connection over port 80. Concurrently it tries this scheme at other hosts.

The SC clients selects its source TCP port in the following way: initially it starts at a port between 1026 and 1040, incrementing successively if no answer arrives. Approximately 2 hours later it stops sending packets for at least 30 minutes. After restarting Skype, it continues incrementing with an average gap of 5 from the previously stopped number (i.e. if Skype stopped last time at 3945, it will begin with port number 3950); Whenever we waited more than one hour before restarting the client it started again with a port between 1026 and 1040. After reaching source port 5000, it starts over from port 1026. Hence, the client depends on an open TCP source port in the range from 1026 until 5000. If the client is not able to establish a connection this way, it will not be able to log in to the network.

Note, that this behaviour was only visible with Skype version 1.4. Version 2.0 gives up after approximately 8 minutes of unsuccessful login attempts and reports an error.

4.1 Proxy Internet Configuration

The Windows version of SC allows the configuration of an internet proxy to gain network access. In one setup we have placed the SC behind a web proxy (Squid on a Linux system), and allowed internet access only through the use of this web proxy.

We have observed that the SC at first tries to directly contact other peers on an port over 1024, port 443, and port 80, in this order. It only uses the proxy as a fall-back if it can not establish a direct connection to other hosts. Thus, a proxy can be used for Skype traffic monitoring only in a network setup where internet access is available exclusively via this proxy.

5 Skype Message Flow Analysis

We have investigated Skype message flows that occur whenever a user logs in to the network.

5.1 General Query / Response Patterns

We discovered a key feature for UDP messages initiating from the client to different destinations, i.e.

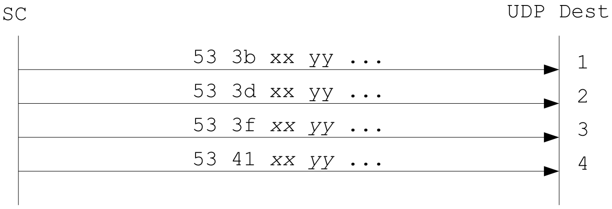

Figure 2: **Key signalling messages**. Initiating UDP signalling messages seem to contain in their first two byte an incrementing session identifier field.

packets that are not sent as a reply to another message. The first two bytes of those UDP requests contain a number that is increased by two on every new request. (Figure 2). This number is of 16 bit size, arranged in network byte order. The first request is sent using an arbitrarily selected number. We have not been able to determine the algorithm upon which the first session number is generated. In our experiments we did not encounter any sequence number close to 0xff 0xff[2] to observe a possible wrap-around. The increase occurs with every new initiating connection, independent of the destination. We conjure that this might be a global session identifier for Skype.

While we could not deduce a pattern for the first two bytes in these responses, we have witnessed a pattern for the third byte for messages with $s = 26$, 51 or 53 byte. These bytes are either set to 0x02 or have the lower nibble[3] set to 0xd.

5.2 Skype Network Traffic Analysis

We have observed three distinct tasks a SC performs after application startup, which we call *UDP Probe*, *TCP SN Handshake* and *TCP Authentication*. Note that these names do not necessarily completely reflect the actual performed Skype operation, as we have not been able to fully decipher Skype operation.

UDP Probe We assume the SC client checks at this stage network restrictions and contacts potential SN. We denote messages exchanged at this stage U1 – U4.

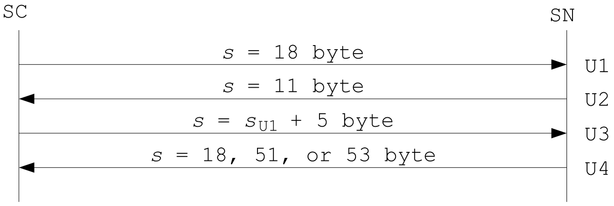

Figure 3: **UDP Probe signalling**. The SC exchanges four messages with a SN to detect possible candidates that allow it to connect to the Skype network (Skype v1.4).

TCP SN Handshake The SC connects to a SN to establish a permanent connection to last for the whole session. We denote messages exchanged at this stage T1 – T4.

TCP Authentication The SC contacts a Login Server (LS) for authentication. We denote messages exchanged at this stage L1 – L4.

5.2.1 UDP Probe

After startup, a SC initiates a double-two-way handshake, involving two queries to a SN (U1 and U3) and two responses (U2 and U4). The message order and the corresponding message sizes can be seen from Figure 3. As described in Sect. 4, the destination port is 33033 when contacting a bootstrap SN, and defined by user preferences for other SN.

Within Skype versions 1.4 and 2.0 there are different message sizes for U1 and U3. With Skype v1.4 s_{U1} is fixed at 18 byte, while v2.0, s_{U1} varies. In both versions the following holds true: $s_{\mathrm{U3}} = s_{\mathrm{U1}} + 5$ byte. Also in both versions $s_{\mathrm{U2}} = 11$ byte and s_{U4} is either 18, 51, or 53 byte. We have analysed these messages regarding their payload and have discovered different types of content exchange, which we call *session identifiers*, *function parameters*, and *IP address exchange*.

Session identifiers. U1 is a initiating message as described in Sec. 5.1, hence the first two bytes

[2] To distinguish hexadecimal number from decimal ones, all hexadecimal numbers are prefixed with *0x*.

[3] A nibble is a an aggregation of four bits. There a two nibbles in a byte, a lower nibble (representing the lower four bits) and a higher nibble (representing the higher four bits).

in this message contain a session identifier. We have discovered equal session identifiers in U2 and U3. However, this match is not kept with U4.

Function parameter The third byte of a message seems to be a message type encoding. We have observed that this value is 0x02 for U1 and U4. The value changes for both U2 and U3, however, the lower nibble stays always the same. It is 0x7 for U2 and 0x3 for U3. In the case of U3, in our experiments we have witnessed that the 4th byte of the message has always the value 0x01.

IP Address exchange The remainder of these messages contain four four-byte-values exchanged between the SC and the SN. We have discovered that two of these 4-tuples contain IP addresses of the originating and destination network. We observed these exchanges within U2 and U3. If $\mathrm{M}_{\text{x-y}}$ denotes the payload bytes x to y in message M then $\mathrm{U2}_{4\text{-}7}$ contains the SC's IP address, while $\mathrm{U3}_{9\text{-}12}$ contains the SN's IP address.

We assume that this message exchange is part of Skype's NAT detection algorithm, which operates similar to STUN (Simple Traversal of UDP over NAT) [16]. Note, that these address fields never contain private addresses. Instead, the host's publicly visible (e.g. the address of the NAT device) is transferred.

Two other 4-tuples are transported twice over the network and occur in the following way: $\mathrm{U1}_{8\text{-}11} = \mathrm{U3}_{13\text{-}16}$, and $\mathrm{U2}_{8\text{-}11} = \mathrm{U3}_{5\text{-}8}$. The significance of these two 4-tuples is unknown.

The full UDP probe message exchange is depicted in Figure 4.

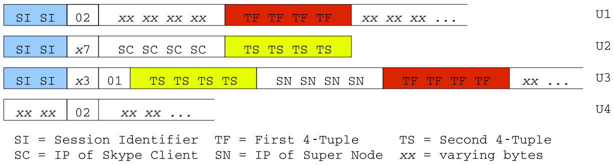

Figure 4: **UPD Probe payload content**. A SC and a SN exchange four 4-tuples. Among them seem to be IP addresses to detect Network Address Translators.

If $s_{\mathrm{U4}} = 18$ byte, the SC successively attempts to establish a TCP connection to the same SN; in case of an answer with $s = 51$ or 53 byte, no TCP connection is initiated and the SC contacts another host with a simpler handshake. Note that the originator of a 51 or 53 byte reply is not contacted any further during the whole session. Especially bootstrap SN always reply in this way. It seems that only an 18-byte answer is an affirmative answer, i.e. the SN is ready to accept further communication. An answer of 51 or 53 byte seems to indicate a rejecting or redirecting answer.

The successive, simpler handshake consists only of a message like U1 with an answer resembling U4; U2 and U3 don't occur. Again, $s_{\mathrm{U4}} = 51$ or 53 byte seem to indicate a negative reply. This simpler pattern is continued until a TCP connection is established. Depending on the number of messages exchanged (four or two) we refer to a *full UDP probe* or a *partial UDP probe*. We denote the handshake *positive* in case of $s_{\mathrm{U4}} = 18$ byte or *negative* in case of $s_{\mathrm{U4}} = 51$ or 53 byte.

The SC continues to initiate partial UDP probes, even after it has been logged into the Skype network. This is necessary in case the SC changes its contacted SN (see Sect. 5.2.2).

5.2.2 TCP SN Handshake

We have observed that the SC needs at least one open TCP connection to a SN to successfully login to the network. The general TCP connection setup works as follows:

1. A TCP connection is initiated to a potential SN after a positive UDP probe. The TCP connection is established to the same port the previous UDP message was received from.

2. If the TCP connection is closed or times out, the SC initiates other TCP connections to different hosts that previously have been positively UDP probed.

3. If a TCP connection is successful, SC and SN exchange messages. After this handshake the TCP connection either is kept alive or closed.

4. Generally, if the TCP connection is kept alive after step 3, it is closed only when the user terminates the application. However, we have observed several instances of the termination of the TCP connection before ap plication close and the establishment of a new TCP connection to a different host. We assume that this happens if the previous SN can not operate as a SN any more or becomes unavailable. In this case the new TCP connection is established in the same way the previous connection was established.

This TCP message flow consists of several messages exchanges between SC and SN. Within the first six messages exchanged, the SC and SN agree upon keeping this connection or dropping it. As this traffic is completely encrypted, we have not been able to determine a significant payload signature. Hence, we concentrate our observation on TCP packet sizes. Due to the fact that Skype "pushes" – i.e. sends TCP messages with the PSH bit set – we've been able to to determine a pattern based on TCP payload size (Figure 5).

For Skype v1.4, after the SC has established the TCP connection to a potential SN, it sends a message with $s = 14$ byte (T1). This is followed by a message with varying sizes. Following, the SC sends a third message with s between 22 and 29 byte, replied with a message with varying sizes (T3). This message is finally acknowledged by the SC with a message with $s = 4$, 15, 16, or 17 byte (T4).

The most significant case is whenever the SN sends T3 with $s = 339$ byte. In all theses cases the TCP connection is terminated and the SC tries to establish a different connection to another SN. If, however the described reply has any other size different from 339 byte and bigger than 57 byte, the TCP connection will be kept alive. This node will become the SN for the SC for the whole duration the user is connected to the network unless the SN itself becomes unavailable.

This procedure is similar with Skype v2.0, except that s_{T1} is not fixed at 14 bytes, but varies. An additonal messages is sent to the SN before T2. T2, T3, and T4 stay the same.

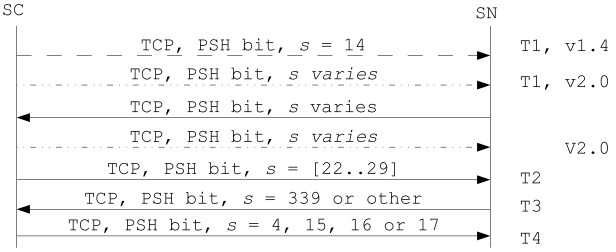

Figure 5: **TCP SN signalling**. The SC seems to negotiate with the SN if the SC is allowed to access the Skype network through this SN.

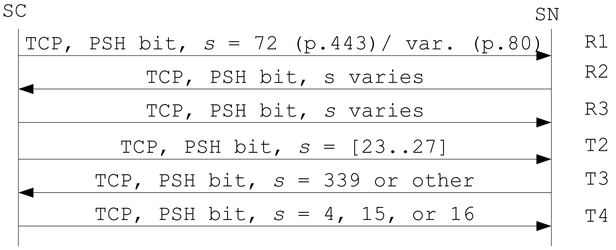
SC
SN
TCP, PSH bit, s = 72 (p.443)/ var. (p.80)
R1
TCP, PSH bit, s varies
R2
TCP, PSH bit, s varies
R3
TCP, PSH bit, s = [23..27]
T2
TCP, PSH bit, s = 339 or other
T3
TCP, PSH bit, s = 4, 15, or 16
T4

Figure 6: **Restricted network signalling** . A modified TCP SN handshake is preceded by three additional messages when communicating over ports 443 and 80.

TCP Restrictions Applied. When a SC client cannot establish a regular TCP connection to a SN after a previous UDP probe, it will try to establish a TCP connection over ports 443 and 80. Note, that although Skype does communicate over those well-known ports, this does not command the usage of the respective protocols. As a matter of fact, Skype seems to use a modification of the Transport Layer Security (TLS) protocol [17] for port 443, but it does not utilize HTTP over port 80.pattern

When operatin in restricted mode, three messages R1 – R3 are exchanged, followed by a slightly modified SN TCP handshake consisting only of messages T2 – T4 (See Figure 6). R1 – R3 vary depending on port usage and Skype version.

Port 443 Operation. We observed that in both versions $s_{\mathrm{R1}} = 72$ byte. Additionally, R1's payload begins with the 56 byte sequence (hexadecimal notation)

```
80 46 01 03 01 00 2d 00
00 00 10 00 00 05 00 00
04 00 00 0a 00 00 09 00
00 64 00 00 62 00 00 08
00 00 03 00 00 06 01 00
80 07 00 c0 03 00 80 06
00 40 02 00 80 04 00 80.
```

We have observed that this message is a fully compliant TLS 1.0 `ClientHello` message wrapped in a SSLv2 record layer [17]. All observed R1 messages differ only in the last 16 byte (R1$_{57\text{-}72}$), which in TLS describe a variable attribute challenge.

The size of the reply R2 varies, however in about 40% we noticed 93 byte. Additionally, the first 79 bytes of R2 are always (hexadecimal notation)

```
16 03 01 00 4a 02 00 00
46 03 01 40 1b e4 86 02
ad e0 29 e1 77 74 e5 44
b9 c9 9c b4 31 31 5e 02
dd 77 9d 15 4a 96 09 ba
5d a8 70 20 1c a0 e4 f6
4c 63 51 ae 2f 8e 4e e1
e6 76 6a 0a 88 d5 d8 c5
5c ae 98 c5 e4 81 f2 2a
69 bf 90 58 00 05 00.
```

This pattern is partly modelled after a TLS `ServerHello` message. Especially interesting are here fields which in TLS context must not be static, while they are in Skype's usage. These are `gmt_unix_time` (R2$_{12\text{-}15}$) which is fixed at Jan, 31, 2004 18:23:18:000000000, and `random_bytes` (R2$_{16\text{-}43}$).

The remainder of R2 starting from byte 80 does not comply to the TLS standard.

The SC responds with R3, with $s_{\mathrm{R3}} = 14$ byte for Skype v1.4, while it varies for v2.0.

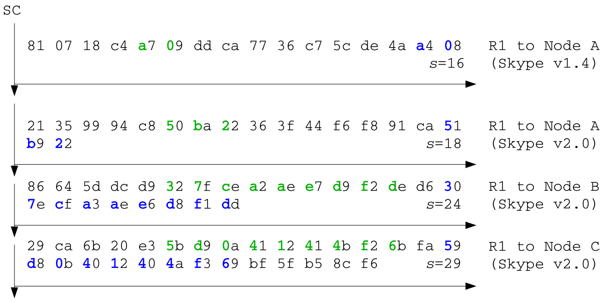
SC
81 07 18 c4 a7 09 dd ca 77 36 c7 5c de 4a a4 08
s=16
R1 to Node A
(Skype v1.4)
21 35 99 94 c8 50 ba 22 36 3f 44 f6 f8 91 ca 51
b9 22
s=18
R1 to Node A
(Skype v2.0)
86 64 5d dc d9 32 7f ce a2 ae e7 d9 f2 de d6 30
7e cf a3 ae e6 d8 f1 dd
s=24
R1 to Node B
(Skype v2.0)
29 ca 6b 20 e3 5b d9 0a 41 12 41 4b f2 6b fa 59
d8 0b 40 12 40 4a f3 69 bf 5f b5 8c f6
s=29
R1 to Node C
(Skype v2.0)

Figure 7: **Port 80 signature (v1.4 and v2.0)**. When communicating over port 80, several nibble sequences (indicated in bold) occur twice in all R1 messages. Note the shift by one position of the first occurrence between Skype v1.4 (topmost) and v2.0 (three remaining).

Port 80 Operation. With Skype v1.4, R1 has always a size of 16 byte. R2 has varying sizes and s_{R3} = 14 byte. We observed varying sizes for R1 – R3 in Skype v2.0 traffic. Additionally, no static payload is transmitted as in the case with port 443. However, we have been able to determine a pattern in the payload content of R1.

In Skype v1.4 we have observed that the higher nibbles of R1$_{5\text{-}6}$ are the same as the higher nibbles R1$_{15\text{-}16}$. In Skype v2.0, the higher nibbles of R1$_{6\text{-}14}$ (note the start at the 6th byte in comparison to v1.4) are the same as the higher nibbles of R1$_{16\text{-}24}$. This has special implications if $s_{\mathrm{R1}} < 24$ byte, as the comparison range will be shorter. See Figure 7.

5.2.3 TCP Authentication

After a TCP connection is established to a SN, the SC client needs to authenticate itself. Baset [10] detected two login servers (LS) 195.215.8.141 and 212.72.49.141, which we confirmed in our experiment.

Generally, four TCP messages (L1 – L4) are exchanged between the SC and a LS, with the first two messages with a size of 5 bytes and varying sizes for successive messages (see Figure 8).

Further messages are exchanged when the user supplies a wrong password. After user authentication the connection to the LS is closed.

All message have some significant payload bytes:

- L1$_{1\text{-}5}$: 0x16 0x03 0x01 0x00 0x00. Note, that this is already the whole content of L1.

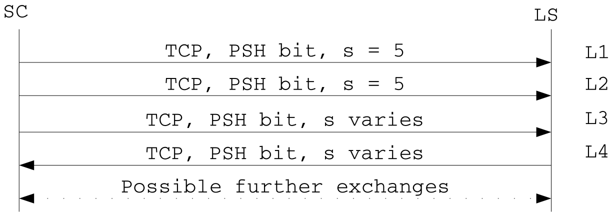

Figure 8: **TCP authentication signalling**. The SC tries first to contact the LS directly. The number of messages exchanges depends on the success or failure of the login attempt.

- L2$_{1\text{-}5}$: 0x17 0x03 0x01 0x00 0x00. Note, that this is already the whole content of L2.
- L3$_{1\text{-}15}$: 0x16 0x03 0x01 0x00 0xcd 0x41 0x03 0x00 0x09 0x80 0x40 0x04 0x08 0xc0 0x01. Additionally, the byte sequence 0x00 0x0c 0x01 0x17 0x03 0x01 0x00 occurs at an undetermined position within L3.
- L4$_{1\text{-}4}$: 0x17 0x03 0x01 0x00.

Again, the bytes sequence 0x16 0x03 0x01 0x00 0x00 also occurs in the beginning of TLS `ServerHello` messages. However, in the same context 0x17 0x03 0x01 0x00 0x00 is not a valid TLS message fragment [10].

We have encountered this traffic pattern also when communicating to two other groups of Skype servers. One group consists of hosts 195.215.8.140 and 212.72.49.155. As these servers are only connected when a users tries to call another user in the PSTN network, we assume them to be Skype-to-PSTN gateways (SkypeOut) [18]. The other group are hosts 195.215.8.142 and 212.72.49.142. We have not been able to determine their functionality, however they are connected whenever a user adds, blocks or deletes an address on his contact list.

Restricting Access to LS. Whenever access to a LS or any of the other described servers is not possible, the SC relays this traffic through another SN [10]. In this case we could only detect messages L3 and L4 exchanged with a randomly selected SN. We did not observe messages L1 and L2 exchanged.

Restricted TCP Access. In the case that TCP access over port 1024 was not allowed, the SC initiated the same handshake over ports 443 and 80. In all three cases the handshake does not differ.

Automatic Login. Upon statup of the Skype application, the user is given the opportunity to save his credentials for further logins to the network. In this case the Skype application stores this information and retrieves it the next time the user starts up the application. The users does not have to enter his credentials again for further login attempts. This behaviour is the default in the Skype application. We assume that this information is stored encrypted in the file `C:\Documents and Settings\<username\Application Data\Skype\user\config.xml`, which contains a section `<Account>` with subsection `<credentials>` that only occurs whenever the *Automatic Login* option is enabled.

With this option enabled, no connection to any login server is performed at application startup, i.e. there is no data present in the traffic that matches messages L1 – L4.

Enabling Automatic Login only affects login traffic, neither UDP probes nor TCP SN traffic is affected.

6 Skype Detection Signature

With our analysis of the protocol at hand we can develop a signature to detect Skype login traffic. A signature should meet three criteria:

1. It should be as compact as possible. A complex signature complicates the monitoring unit which results in less performance with multiple concurrent traffic patterns to monitor.
2. The signature should consider all possible cases how traffic could be injected in the network; Otherwise Skype sessions might evade detection.
3. The number of detected false positives, i.e. the detection of a Skype session if in reality no Skype session was established, should optimally be zero.

From our analysis we have learnt that all three goals are difficult to achieve with Skype traffic. For exmple, a first idea could be to scan for packets with byte sequence 0x16 0x03 0x01 0x00 0xcd 0x41 0x03 0x00 0x09 0x80 0x40 0x04 0x08 0xc0 0x01, as they always occur in packets exchanged with the LS. With this method however, one can not detect subsequent logins to the LS as desribed in Sect. 5.2.3. Furthermore, it it impossible to determine the end of the Skype session.

We therefore propose a different signature which is based on UDP probe detection and TCP SN handshake detection. Additionally, we monitor ports 443 and 80, in case the SC is deployed behind a restricted network. The signature consists of multiple steps, with the actual detection desribed in steps 7 to 9.

1. Network traffic scans for UDP packets with a payload size of 18 bytes and the third byte of the payload set to 0x02 (U1). Note, that for Skype v2.0 detection only payload scanning is feasible, as the size of the packets might differ. Save both IP address / port tuple of the originator (CON_{SC}) and the destination (CON_{SN}). Also, save the first two bytes of this message ($\text{U1}_{\text{1-2}}$).

2. Scan for a reply $\text{CON}_{\text{SC}} \leftarrow \text{CON}_{\text{SN}}$ (U2) with $s = 11$ byte and $\text{U2}_{\text{1-2}} = \text{U1}_{\text{1-2}}$ and the lower nibble of $\text{U2}_3 = 7$. Alternatively, scan for the message described in step 4 (In case of a partial UDP probe).

3. Scan for another request $\text{CON}_{\text{SC}} \rightarrow \text{CON}_{\text{SN}}$ (U3), with the following three conditions: 1) $s_{\text{U3}} = s_{\text{U1}} + 5$; 2) $\text{U3}_{\text{1-2}} = \text{U1}_{\text{1-2}}$; and 3) lower nibble of $\text{U3}_3 = 3$.

4. Scan for a reply $\text{CON}_{\text{SC}} \leftarrow \text{CON}_{\text{SN}}$ (U4) with $s_{\text{U4}} = 18$ and $\text{U4}_3 = 0\text{x}02$. We now have detected Skype usage in the network.

5. Scan for subsequent occurrences of steps 1-4. This is necessary to detect a possible SN change.

6. Scan for a TCP connection $\text{CON}_{\text{SC}} \rightarrow \text{CON}_{\text{SN}}$. Note, that any of the tuples CON_{SC} / CON_{SC} as described in steps 1-4 need to be taken into account. This might become the TCP SN handshake.

7. Within this TCP connection scan from a packet $\text{CON}_{\text{SC}} \rightarrow \text{CON}_{\text{SN}}$ with s between 22 and 29 byte (T2).

8. Scan for a reply from $\text{CON}_{\text{SC}} \leftarrow \text{CON}_{\text{SN}}$ (T3). Check if $s_{\text{T3}} = 339$ byte, in this case the contacted node will not become the SN. Continue again from step 6.

9. If $s_{\text{T3}} \neq 339$ and $s_{\text{T3}} > 57$, scan for a final reply from $\text{CON}_{\text{SC}} \rightarrow \text{CON}_{\text{SN}}$, with $s = 4$, 15, 16, or 17 byte. We now have detected a successful connection from the SC to a SN.

10. Continue scanning from step 6 onward to detect a possible SN change during the session. After a SN change, keep the parameters of the last active SN, discarding previous SN parameters.

11. Scan for TCP connection close of the last active SN connection. This signals the termination of Skype usage by the client.

To take detection over ports 443 and 80 into account, the following steps need to executed:

For port 443 traffic, scan for outgoing packets with $s = 72$ byte that begin with the hexadecimal pattern

```
80 46 01 03 01 00 2d 00
00 00 10 00 00 05 00 00
04 00 00 0a 00 00 09 00
00 64 00 00 62 00 00 08
00 00 03 00 00 06 01 00
80 07 00 c0 03 00 80 06
00 40 02 00 80 04 00 80.
```

As in step 1, save both IP address / port tuple of the originator (CON_{SC}) and the destination (CON_{SN}).

This is a already very strong indication of Skype traffic. However, for more evidence scan for a reply $\text{CON}_{\text{SC}} \leftarrow \text{CON}_{\text{SN}}$ beginning with

```
16 03 01 00 4a 02 00 00
46 03 01 40 1b e4 86 02
ad e0 29 e1 77 74 e5 44
b9 c9 9c b4 31 31 5e 02
dd 77 9d 15 4a 96 09 ba
5d a8 70 20 1c a0 e4 f6
```

```
4c 63 51 ae 2f 8e 4e e1
e6 76 6a 0a 88 d5 d8 c5
5c ae 98 c5 e4 81 f2 2a
69 bf 90 58 00 05 00.
```

We have detect initial SN communication. Continue with step 7.

For port 80 communication examine outgoing packets (R1) where the higher nibble of $R1_{5\text{-}6}$ = higher nibble of $R1_{15\text{-}16}$ (for Skype v1.4) or the higher nibble of $R_{16\text{-}24}$ = higher nibble of $R1_{16\text{-}24}$ (for Skype v2.0). We have detect initial SN communication. Continue with step 7.

6.1 Signature Analysis

This signature is complete in a way that it can successfully detect Skype session start and stop under all known Skype network operation modes. For this, this signature does not need to take TCP Authentication traffic into account. It is sufficient to monitor traffic involving SN connections, as authentication traffic occurs after a connection to a SN has been established. Only in the unlikely case that the user has no automatic login activated and provides wrong credentials, this signature detects Skype usage while in fact only the application is running.

Note, that the actual detection is performed in steps 6 to 9. Other steps are applied to imporve detection accuracy.

6.2 Application

We have incorporated the signature as a module into the OpenIMP measurement platform [19]. We have thouroughfully tested the monitoring capabilities with all our previously recorded tracefiles and also installed it as a life monitor for an internal network. In all cases, the monitor was able to successfully detect Skype network traffic. Additionally, we have run different other internet services like VoIP (SIP, IAX2), Web Access (HTTP / HTTPS), and Messaging Services (MSN) in the same network. The monitoring tool did never falsy classify them as Skype traffic.

We also installed Skype v2.5 in our test network. The monitoring tool was able to successfully detect Skype session. However, there seems to be a variation in port 80 restricted traffic (R1). The monitoring tool was still able to detect port 80 restricted Skype traffic by checking the TCP SN handshake (signature steps 7 to 9).

7 Conclusions and Future Work

In this report we have presented an analysis of different message flows of the Skype VoIP application. Skype implements several features impeding Skype usage detection from a network level, which are unavailable protocol specification, complete encryption of all messages, clandestine port number usage and several different operation modes.

We have been able through packet inspection to determine several characteristics of the Skype protocol that can be used for packet monitoring. Skype detection is possible through deep packet inspection in combination with port usage correlations. Our developed signature is able to detect Skype v1.4 and v2.0 usage from the moment the user logs into the Skype network until it logs out from the service.

Still, detecting Skype traffic is not a lightweight task. Because of Skype's P2P character, the security operater has to install monitoring systems at all network egress points to detect all user operation. Additionally, exact detection of Skype usage in high traffic scenarios requires powerful monitoring hardware, as monitoring with our signature needs to operate statefully, and multiple packets have to be analysed down to the payload level. Successful detection requires continuous monitoring, i.e. it is not possible to take network snapshots and present a result solely on this snapshot analysis. We have seen that patterns differ between the examined version 1.4, 2.0, and 2.5. It is likely that this will continue with newer Skype versions.

In future work we would like to overcome some of these limitations. Our main interest lies in the detection of actual voice calls. i.e. the detection of active usage of the application. We see two possibilities here. First, it might be possible to detect signatures also for voice signalling traffic. However, we judge the probability to detect such signature rather low. Much of the traffic patterns we describe in this report are detectable due to the fact that we have concentrated our efforts on Skype traffic occurring immediately after application launch, where full encryption of traffic still has to be established. This is not the case when the user is already logged in to the network. We are therefore following the recent progress in decrypting Skype operation with interest [11]. A second possibility to detect voice calls lies in the analysis of the voice traffic itself. Detecting unique patterns in this traffic would also allow eliminate the requirement of the monitoring agent to constantly analyse all incoming traffic.

Acknowledgements. We would like to thank Thomas Günther for additional input and initial testing tools.

Used Abbreviations

CON Connection tuple (address / port)

L*x* TCP login server authentication message

LS Login Server

PSTN Public switched telephone network

R*x* Restricted access message

s size of packet payload

SC Skype Client

SN Super node

TLS Transport Layer Security

T*x* TCP super node handshake message

U*x* UDP probe message

References

[1] Google talk (beta). http://www.google.com/talk.

[2] Yahoo messenger with voice. http://messenger.yahoo.com.

[3] Skype - the whole world can talk for free. http://www.skype.com.

[4] Gizmo - a free phone for your computer. http://www.gizmoproject.com.

[5] G. Camarillo A. Johnston J. Peterson R. Spark M. Handley E. Schooler J. Rosenberg, H. Schulzrinne. Session initiation protocol. *RFC 3261*, 2002.

[6] V. Paxon. Bro: A system for detecting network intruders in real-time. *Computer Networks, 31(23-24)*, 1999.

[7] A. Valdes and K. Skinner. Adaptive, model-based monitoring for cyber attack detection. *RAID 2000*, 2000.

[8] J. Kurose D. Towsley K. Suh, D.R. Figueiredo. Characterizing and detecting relayed traffic: A case study using skype. *UMass Computer Science Technical Report 2005-50*, 2005.

[9] N. Daswani S. Guha and R. Jain. An experimental study of the skype peer-to-peer voip system. *5th International Workshop on Peer-to-Peer Systems (IPTPS '06)*, 2006.

[10] H. Schulzrinne S. A. Baset. An analysis of the skype peer-to-peer internet telephony protocol. *IEEE Infocom*, 2006.

[11] D. Fabrice. Skype uncovered, 2005. http://www.ossir.org/windows/supports/liste-windows-2005.shtml.

[12] G. Lisha and L. Junzhou. Performance analysis of a p2p-based voip software. *Advanced International Conference on Telecommunications and-International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, 2006.

[13] M. Fiedler K. Tutschku T. Hossfeld, A. Binzenhoefer. Measurement and analysis of skype voip traffic in 3g umts systems. *University of*

Wuerzburg - Institute of Computer Sciencet - Report No. 377, 2005.

[14] K. W. Ross J. Liang, R. Kumar. The kazaa overlay: A measurement study. *Computer Networks 49, 6*, 2005.

[15] M. Holdrege P. Srisuresh. Ip network address translator (nat) terminology and considerations. *RFC 2663*, 1999.

[16] C. Huitema R. Mahy J. Rosenberg, J. Weinberger. Stun - simple traversal of user datagram protocol (udp) through network address translators (nats). *RFC 3489*, 2003.

[17] C. Allen T. Dierks. The tls protocol version 1.0. *RFC 2246*, 1999.

[18] Skypeout skype-to-pstn service, 2004. http://www.skype.com/products/skypeout/.

[19] Open imp – internet measurement project. http://www.ip-measurement.org/openimp/.