Issues with the Interworking of Application Layer Protocols and the MIDCOM Architecture

Andreas Müller and Sebastian Kiesel

Abstract— **In the current Internet, there are many devices performing specific functions in the network going beyond simply routing and forwarding IP packets to their destination. Common examples for such devices—which are frequently referred to as *middleboxes*—are firewalls and network address translators. Middleboxes usually operate exclusively on the network and transport layer in order to fulfill their task, mostly taking adequate action on the basis of the addresses and port numbers of traversing IP packets. However, there are some services ordinary middleboxes do not get along with. As a result of that, the corresponding data packets mostly do not reach their actual destination and hence, a reasonable communication is normally no longer possible.**

One approach for dealing with this problem is the so-called IETF MIDCOM (MIDdlebox COMmunications) architecture. In this paper—after elucidating the problem in more detail and providing an overview of other existing solution approaches—, we especially focus on MIDCOM and we address some potential problems concerning the interworking between this approach and some specific services, including possible solutions for coping with them. This way, we want to point out the issues that should be taken into account when designing new application layer protocols or when extending existing ones in order to make them fully interoperable with the MIDCOM architecture.

Keywords— **IETF MIDCOM, middlebox, firewall, NAT, SIP, protocol design**

I. Introduction

In the current Internet, there are many devices, such as firewalls or network address translators (NAT), performing specific functions in the network going beyond simply routing and forwarding IP packets to their destination. These devices—which are frequently referred to as *middleboxes*—have all in common, that they usually evaluate exclusively protcol-related information from the network and the transport layer in order to fulfill their tasks. Firewalls, for example, represent a building block for protecting local networks from potential attackers. They are widely-spread used for the enforcement of access policies on the data traffic routed accross a borderline between two domains of different security levels and requirements. The currently most prevalent form are pure packet filters, usually deciding on the basis of IP addresses and port numbers whether a packet should be forwarded or not. Alternatively, a firewall could also be realized as an application layer gateway (ALG), but as ALGs are usually more complex to implement and need more time for processing traversing data flows, they are not so often used as the other alternative. For that reason, we will focus in this paper on scenarios, where the bulk data (and especially real-time multimedia streams) is inspected by pure packet filters only. Signaling messages may be handled by application layer entities, as they are less frequent and less sensitive to delays.

Andreas Müller and Sebastian Kiesel are with the Institute of Communication Networks and Computer Engineering (IKR), University of Stuttgart, Pfaffenwaldring 47, D-70569 Stuttgart, Germany. E-mail: {muellera, kiesel}@ikr.uni-stuttgart.de.

The network administrator, who is in charge of the firewall, usually wants to allow or disallow the use of a predefined set of services. Packet filters, in contrast, do not have any notion of services. Their decisions whether to allow (forward) or disallow (discard) a packet are rather based on the IP addresses and TCP or UDP port numbers contained in the individual packets. For many Internet services, like e-mail or web browsing, generally dedicated, well-known port numbers are used. Consequently, this way, it is possible to map services to port numbers and vice versa, so that from the list of (un)wanted services a static access control list (ACL) can be generated, containing all port numbers of the packets that should be forwarded and serving as the basis for any firewall decision.

Basically, this would be all well and good, if there were not some services behaving differently. The services considered here deviate from the well-known port number concept and their major characteristics can be put down to two single points:

1. They are session-oriented, i. e., a communication relationship usually covers several different flows.
2. The connection parameters—such as IP addresses and port numbers—of at least some of these flows are dynamically negotiated using some special control flow.

Common examples for such services are IP-telephony, the File Transfer Protocol (FTP) as well as many peer-to-peer services. An IP-telephony session, for example, normally is made up of one signaling flow and one or more data flows for the transport of the actual media (voice) data. One very common protocol for signaling is the Session Initiation Protocol (SIP), which normally uses the well-known port number '5060'. However, the IP addresses and port numbers defining the end points of the corresponding media streams are dynamically negotiated during the SIP session establishment phase. As a consequence, an ordinary packet filter actually would be able to detect all IP packets belonging to the signaling flow, but it would not be able to identify any packets belonging to the data flows. Therefore, these packets in most cases would be simply discarded. This, of course, represents a very strong restriction to the usage of such services and therefore a serious, generally applicable solution has to be found.

NATs, on the other hand, are frequently used for mapping and aggregating private IP addresses to public ones and vice versa, thus facilitating the connection of local area networks with a private address space to the public Internet. However, if IP addresses are dynamically negotiated between two peers and carried as part of special control messages—what is the case for the considered services—, these addresses cannot be mapped by a conventional NAT. Therefore, a peer in the public Internet possibly would try to send data packets to a private IP address, what in most cases naturally would fail. Consequently, this means, that in case of NATs, problems do not only arise due to the use of non-well-known port numbers, but also due to the dynamic negotiation of IP addresses. However, it has the same effect, namely that a reasonable communication is not feasible.

Similar effects appear in conjunction with virtually any other kinds of middleboxes, but for simplicity, we restrict to considering only packet filters and NATs for the rest of this paper, as these two entities are the currently most prevalent form of such devices. Nevertheless, the general principles and conclusions hold for almost all other kinds of middleboxes analogously.

Basically, there exist several different approaches for dealing with these problems. One of them is the so-called MIDCOM architecture, which is developed and specified by the IETF (Internet Engineering Task Force) MIDCOM working group. The basic idea of this approach is to analyze the control flow of a session—for example with the help of an intermediate proxy server—, to extract the dynamically negotiated connection parameters of the data flows and to configure the middlebox on the basis of this information on the fly. Even though being fundamentally well-suited for coping with the considered issues, it may come to some problems with respect to the interworking between application layer protocols, such as SIP, and the MIDCOM architecture. This fact was the major reason for writing this paper. In the following, we want to make aware of the potential problems, that may arise when applying the MIDCOM architecture, and we want to point out the issues that should be taken into account when designing new application layer protocols or extending existing ones in order to make them fully interoperable with this approach. Our goal is to provide a systematic and comprehensive overview of these issues and to formulate specific, generally applicable mechanisms that should be supported by 'MIDCOM-friendly' protocols.

The remainder of this paper is structured as follows: In section II, several existing solution approaches for coping with the considered problems are presented and shortly evaluated with respect to their suitability. Section III subsequently especially focuses on the MIDCOM architecture, which is illustrated considering the concrete example of enabling SIP-based communications through a firewall. Afterwards, in section IV, the potential problems concerning the interworking between the MIDCOM architecture and some services showing the characteristics as presented before are put together, and we present possible solutions for coping with them. This leads to some general guidelines, that should always be taken into account when designing new services which rely on the MIDCOM architecture or when extending existing ones in order to make them fully compliant with this approach. Finally, in section V, we conclude with a short summary of the most important observations made.

II. Existing Solution Approaches

In the following subsections, the basic solution approaches for dealing with the problems arising in conjunction with middleboxes are shortly presented, always considering a packet filter as middlebox.

A. Removal of the Packet Filter

The trivial solution would be to simply remove the firewall, so that all data traffic is forwarded, independently of the actual service the respective data packets are belonging to. Even though this could be done to cope with the specified difficulties, it is not recommended to do so, because by having the firewall removed, the network would become vulnerable to potential attackers, as no longer any traffic control is performed. For that reason, this approach can be labeled as completely unsuitable.

B. Setting up a Permissive, Static ACL

Basically, it would also be possible to keep the static configuration of the firewall, but in that case, a sufficient number of permanent pinholes has to be set up. For enabling the use of SIP-based IP-telephony, for example, all UDP traffic using port numbers larger than 1024 ought to be forwarded by the firewall by default, as any of these ports numbers might be used for a dynamically negotiated media stream.

Of course, this proceeding would be somewhat better than removing the firewall completely, but nevertheless, it still remains dubiously whether the firewall can guarantee a sufficient degree of security for a local network if such a high number of ports is opened by default. Therefore, it should also be refrained from applying this alternative.

C. Establishing an HTTP Tunnel

Most firewalls grant users access to the world wide web. This fact is frequently capitalized on to enable the use of services the firewall normally would block and hence represents another possibility for coping with the considered problems. The basic idea behind this approach is to tunnel arbitrary binary data—in our case, we would tunnel whole IP packets—through the HTTP protocol. For doing that, all packets belonging to dynamically negotiated data flows have to be encapsulated in an appropriate way and then can be sent over the HTTP tunnel. At the other end of such a tunnel, the actual user data has to be decapsulated again, whereupon the original data flow can be reconstructed. For bearing the HTTP connection, a

TCP connection has to be established, using the well-known port number for HTTP (80/tcp). This way, a simple packet filter can only observe TCP segments with this well known port number, but it cannot recognize the tunneled packets.

Even though an HTTP tunnel basically could facilitate the use of any kind of services through a firewall, it basically circumvents the packet filter and its access policy. Therefore, this approach cannot be advocated as a good solution from a security point of view. Furthermore, it is an indispensable prerequisite that both peers involved in a session support the establishment of an HTTP tunnel, what does not hold in general, of course. On the other hand, the end-to-end delay for the data transmission naturally would increase significantly, as the encapsulation and decapsulation process might be very time-consuming and the transport over TCP itself also introduces some additional delay, due to congestion control or repeated retransmission mechanisms, for instance. However, for many real-time critical services, like IP-telephony, for example, such a high delay would most likely be unacceptable.

D. Using an Application Layer Gateway

The usage of an application layer gateway (ALG), acting as a proxy server for both, the control and the data flows, represents another alternative to overcome the problems presented before. The fundamental principle of this approach is to forward all messages belonging to the control or the data flows via an intermediate ALG rather than exchanging them directly between the involved peers. Hence, the ALG terminates any flow and re-establishes it again to the other peer, thus actively participating in any message transfer. The access control for the respective service is not performed in a packet filter, but in the ALG itself, which in that case consequently functions as the policy enforcement point.

However, it has to be ensured, that all messages first of all are sent to the ALG. For the control messages, this could be achieved by an adequate configuration of the end devices, for instance, or alternatively by using some kind of transparent proxy. For the actual data, on the other hand, it is a little bit more complicated, as the corresponding connection parameters are dynamically negotiated by means of the control flow. Therefore, it is in the responsibility of the ALG itself to enforce that all data is sent via it. This can be achieved by manipulating the messages of the control flow accordingly. For example all IP addresses specifying the end points of a data flow have to be replaced by the IP address of the ALG and the port numbers normally also have to be modified in a proper way, in order to be able to share one ALG among several users. Similarly to a NAT—but operating on the application layer—the ALG then has to perform some address/port mapping for all received packets and forward them to the actual recipient. Figure 1 shows the basic arrangement of such a solution.

Basically, this approach is rather well-suited for

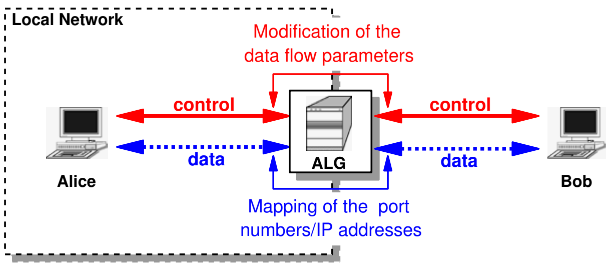
Local Network
Modification of the data flow parameters
control
control
data
data
ALG
Alice
Bob
Mapping of the port numbers/IP addresses

Fig. 1. Using an application layer gateway

dealing with the dynamically negotiated data flows. However, the redirection of all messages to the ALG and the analysis and modifications performed by this device also introduce an additional delay, which might be not acceptable for a multitude of different services with comparatively high real-time requirements. Furthermore, an arrangement as depicted in figure 1 has only a very limited scalability. This is mainly because an ALG always has to process any messages exchanged between two peers and keep state of all active sessions, so that it shows a rather high complexity. In the end, we can say, that it depends on the actual service and especially the requirements of that service to the end-to-end delay whether this approach represents a suitable solution or not.

E. Dynamic Configuration of the Packet Filter

Another approach is the dynamic configuration of the packet filter on the basis of the dynamically negotiated connection parameters of one or more data flow(s). For that purpose, the control flow has to be analyzed in order to be able to determine all relevant parameters and to open or close corresponding pinholes in the firewall on the fly. In this context, there are by and large two degrees of freedom for realizing this analysis:

1. *How* the analysis is performed
2. *Where* the analysis takes place

Concerning the way the analysis is undertaken, one has to differentiate between a passive and an active solution. Passive means, that the device, which is responsible for the analysis, operates exclusively on the network and transport layer and that it especially does not terminate the control flow. In this case, the application layer messages can only be evaluated by looking at the payload contained in individual IP packets passing by. When applying an active analysis, on the other hand, the control flow is terminated by an intermediate proxy server and re-established again to the other peer. This is more powerful than analyzing messages passively, as a much more detailed analysis of the flows is possible, but it takes usually more time than a passive solution, thus causing a higher end-to-end delay.

The other degree of freedom is where the analysis actually is carried out. This could be done either in the same device the firewall is integrated in or alternatively by another device, which then has to communicate the required pinholes to the firewall, using some special control protocol, for instance.

E.1 Passive analysis

If the analysis is done passively and inside the firewall itself, this is frequently referred to as a modular firewall. A passive analysis in an external instance basically does not make any sense, because on the one hand, the gain in performance, which can be obtained through a passive analysis of the control flow, would get lost again by the lavish communication between the firewall and the analyzing instance, so that you just as well could use a more powerful proxy server. On the other hand, it is rather hard to ensure, that all messages are re-directed to the device which is responsible for the message analysis, if this device operates exclusively on the network and the transport layer. For these reasons, one should desist from applying such a solution.

Modular firewalls are made up of the core packet filter functionality and additional application-specific modules, which are responsible for analyzing the payload data contained in the IP packets belonging to a particular service in order to be able to set up or remove pinholes in the firewall dynamically. An example for such an application-specific module is a *protocol helper* for Linux's netfilter system, which already exists for FTP and a couple of other services. However, such modular firewalls have some serious drawbacks:

Aside from the limited scalability of this solution, a modular firewall is not able to analyze all messages belonging to a control flow, as no TCP/UDP re-assembly is performed and therefore application layer messages being spread across several IP packets cannot be reasonably evaluated. Furthermore, in the case that the dynamic configuration of the firewall fails—for example due to potentially existing conflicts with high-level policies—the firewall has no means to apprise the involved parties of this fact. So consequently, no data exchange could take place any more, without the end users being aware of the actual reasons for that. Therefore, such a solution is only applicable in a rather restricted, lowly loaded scenario, but not in the access network of an Internet service provider, for instance.

E.2 Active analysis / The MIDCOM Approach

Most of the problems arising in conjunction with modular firewalls can be overcome when analyzing the application layer messages actively, for example with the help of a special proxy server. Due to the fact, that a proxy server terminates any control flow and re-establishes it again, it is able to reassemble TCP and UDP segments if necessary and it can directly influence the signaling between the peers involved in a session, for example by generating or modifying corresponding application layer messages. This is especially important for the case, that the dynamic configuration of the firewall could not be performed successfully, as this way it is possible to inform the affected end devices about the failure in a proper way.

On the one hand, such a proxy server could be directly integrated in a firewall, performing the dynamic configuration with the help of a special application programming interface, for instance. However, due to the direct coupling of the proxy server and the firewall and the affiliated increase in complexity of that device, such a solution still shows a very limited scalability. On the other hand, however, the active analysis could also be realized by an external instance, which controls the firewall using some special control protocol. This approach is frequently referred to as the *MIDCOM* approach. Basically, the device which is responsible for the analysis of the control flows—the so-called *MIDCOM agent*—could be either an intermediate proxy server or alternatively one of the end devices being involved in a session. However, if the dynamic configuration is performed by the end devices themselves, this introduces a couple of new problems: In this case, it is for example not possible to use arbitrary devices, as they necessarily have to support the corresponding functionality. Furthermore, it is certainly a rather delicate issue to empower the end devices to open and close arbitrary ports in the firewall, because by doing so, the configuration of this device and hence the enforcement of the access policy, for which the firewall originally was installed, is getting out of control of the network administrator.

Therefore, it is advantageous to use a dedicated proxy server—a so-called *MIDCOM proxy*—, which could be used to enforce some high-level policies introduced by the network administrator, for example. In general, this approach also shows a very high scalability, as load balancing is possible. The basic arrangement of such a solution is depicted in figure 2.

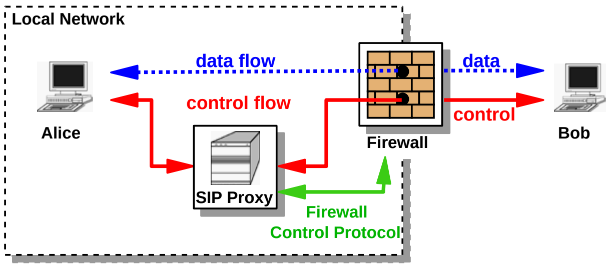

Fig. 2. Arrangement of the MIDCOM approach

For the rest of this paper, we will only consider the MIDCOM approach with a MIDCOM proxy, as this alternative seems to be one of the most promising solutions and because it was also chosen by the IETF for dealing with the problems brought up before.

III. The MIDCOM Approach for SIP

In order to identify specific problems of the interworking between SIP and the MIDCOM architecture, we implemented an arrangement as depicted in figure 2 for enabling the use of SIP-based IP-telephony through a firewall. An example for a typical call flow of such a scenario is shown in figure 3. All SIP messages exchanged between the involved peers are always sent via an SIP proxy server. This proxy is responsible for the analysis of these messages, takes adequate action if necessary and normally forwards them to the actual recipient afterwards.

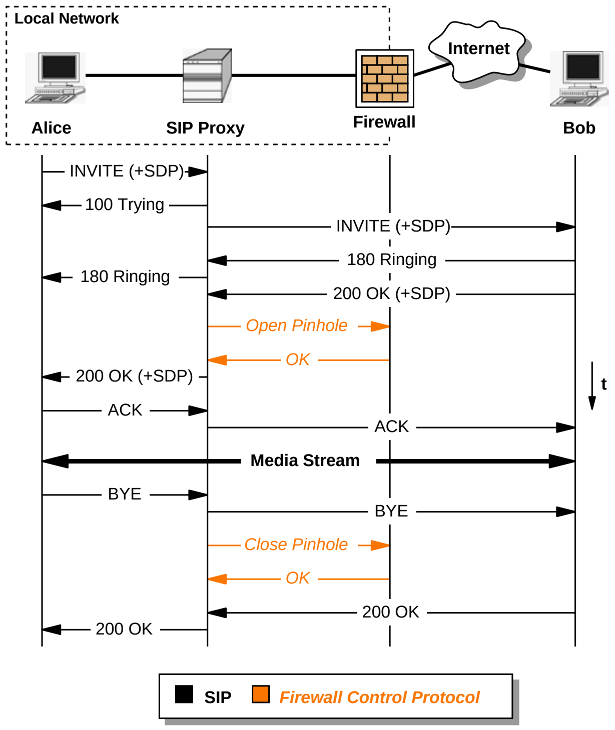

Fig. 3. The MIDCOM approach for SIP

The fundamental procedure for establishing an SIP session is basically always the same: The initiator of the call, who in the considered example is called Alice, first of all issues an INVITE request, indicating that she wants to set up a new session. Upon reception of this message, the called party (in our case Bob)—provided that he is willing to accept the session—replies with a ‘200 OK’ response, which finally has to be confirmed by Alice with a concluding ACK request again. Afterwards, a media stream is set up between the two peers, which subsequently can be used for exchanging any kind of data, like voice or video, for example. In contrast to the SIP messages, the data flow is usually not re-directed to a proxy server, but directly exchanged between the involved peers.

The parameters for the media streams are usually dynamically negotiated by means of the *Session Description Protocol* (SDP). For that purpose, either the initial INVITE request and the corresponding ‘200 OK’ response (like in the considered example), or the ‘200 OK’ response and the concluding ACK request contain such SDP messages, which are used for specifying the IP addresses and port numbers representing the end points of these data flows as well as other parameters (e. g., codec types, etc.). In order to be able to make the exchange of media data possible, the SIP proxy server consequently has to extract these parameters and as soon as it has enough information, it should request the firewall to open a corresponding pinhole. After having received a positive acknowledgement confirming the dynamic configuration, the proxy server finally can forward the initially detained ‘200 OK’ response to the actual destination, thus continuing the session establishment procedure. The pinhole is closed again, as soon as one of the involved parties terminates the session by sending a BYE request to the other peer.

In addition to the already mentioned SIP messages for establishing a new session, figure 3 shows also some so-called provisional responses, having a response code starting with one. However, for the MIDCOM principle itself, provisional responses actually are neither important nor required and therefore, they are not treated in more detail here.

An issue, which was not addressed so far, is the question, which protocol would be most suitable for the communication between the proxy server and the firewall. Of course, this is a pivotal point of the MIDCOM approach and therefore the Internet Engineering Task Force (IETF) brought the so-called MIDCOM working group into being, which inter alia is dealing with this question. In this context, they have already worked out some general protocol requirements, detailed semantics of such a protocol and they evaluated several existing IETF protocols with respect to their suitability [6], [9], [8], [3]. The working group concluded, that the Simple Network Management Protocol (SNMP) seems to be the most appropriate alternative. However, there are also some efforts to design a totally new protocol from scratch, being completely consistent with the established requirements and semantics. Such a new protocol is the *Simple Middlebox Configuration Protocol* (SIMCO), for example, which was only published as an Internet draft, so far [7]. SIMCO—specifically tailored to the MIDCOM approach—is much more lightweight than SNMP and hence obviously easier to handle. However, as it is not a crucial factor for the fundamental considerations which control protocol is actually being used, in figure 3 only some textual circumlocutions are used.

IV. Inherent Problems

In principle, the MIDCOM approach is well-suited to cope with the difficulties arising in conjunction with the traversal of middleboxes. However, as many application layer protocols have been designed without considering the requirements of this approach, it frequently comes to some problems. Such problems predominantly appear if the dynamic configuration of the middlebox fails, if the session, for which the middlebox had been configured dynamically, is released abnormally or if certain security mechanisms are used. Subsequently, both the inherent problems as well as possible solutions for coping with them are presented and clarified again considering the concrete example of enabling the use of SIP-based communications in presence of a packet filter. The outcome of our analysis is a set of specific protocol mechanisms, which are indispensable prerequisites for a reasonable application of a system pursuant to figure 2 and which consequently should always be taken into account when designing new application layer protocols or when extending existing ones in order to make them fully interoperable with the MIDCOM architecture.

A. MIDCOM Error Handling

Theoretically, of course, it is always possible, that the dynamic configuration of a middlebox fails. In case of a packet filter, for example, this would mean, that a pinhole—which actually is required for a data flow—cannot be opened successfully. Possible reasons for that are a disturbed communication link between the proxy server and the middlebox or potentially existing conflicts with some high-level policies, for instance. Concerning this matter, at least two different issues have to be considered: On the one hand, how the end users can be notified about the failure of the dynamic configuration in a proper way and on the other hand, how an effect frequently referred to as 'ghost ringing' could be avoided.

A.1 Notification of the end users

If the dynamic configuration of a middlebox fails, it is normally not possible to exchange any data between the involved peers, as the IP packets belonging to the data streams never reach their destination. For that reason, the responsible proxy server has to inform all users participating in the corresponding session about this circumstance—otherwise, they would not be able to take any notice of it, apart from recognizing the disturbed data flow. In this context, the most important question certainly is, in which way the failure could be appropriately communicated to the end users. In the majority of instances, the most reasonable reaction certainly would be the termination of the complete session, as it normally does not make any sense to maintain a session if no data exchange is possible.

For aborting a session, the proxy server theoretically could rely upon existing protocol messages, which are normally used for terminating a session under control of the respective end devices. In case of SIP, for example, the proxy server could send a BYE request, thus initiating a regular connection release. However, if it is done this way, the involved peers are still not able to recognize, that the actual reason for the canceled session is the failed configuration of the middlebox. In their opinion, the session rather would have been terminated by the respective communication partner in the normal way. Therefore, it is desirable to have specific protocol elements, explicitly indicating the reason for why a session is being terminated. Such protocol elements could be either specific requests or error messages, which may only be sent by the device responsible for the dynamic configuration of the middlebox, or some extensions for existing messages—like additional header fields—, specifying the respective reason for why these messages have been sent.

Considering again the concrete example of SIP-based communications, one possible measure would be the introduction of additional SIP headers, called *reason headers.* Reason headers actually specify the reasons for why the respective messages have been sent [5]. In contrast to the creation of new message types, these extensions have the advantage, that even devices which are not aware of them can handle the corresponding messages more or less reasonably, as all unknown header fields normally are just ignored. However, in that case, the end devices again would not be able to recognize that the message have been sent by a proxy server due to the failed configuration of a middlebox, but at least they would realize the desire to terminate the session.

A.2 Ghost Ringing

Another concomitant phenomenon appearing as a consequence of the failed configuration of a middlebox is something frequently referred to as '*ghost ringing*'—especially in conjunction with IP telephony. 'Ghost ringing' can occur during the establishment of a new session and means, that the called party is notified about the request to set up a new connection, even though the session can never be completely established, as the dynamic configuration of an intermediate middlebox fails. In case of IP telephony, for instance, this would mean, that the phone of the callee rings, he picks up the receiver, but nobody is on the line. Of course, such a bothering effect like 'ghost ringing' is not desirable at all and therefore possible remedies have to be found.

The ultimate goal is to defer the notification of the called party about the request to establish a new session until it can be assured, that all middleboxes on the data path between the caller and the callee can be successfully configured. For that purpose, it is essential, that the called party announces the parameters used for its peer of the data flow(s) before the actual end user is notified about the wish to set up a new session. For SIP, for example, this is normally not the case, as the callee's phone starts ringing as soon as an INVITE request is received. In order to be able to realize such a behavior, naturally some specific protocol mechanisms are required. This could be for example done by an additional message exchange between the called party and intermediate proxy servers, using some dedicated message types.

If we consider the concrete example of enabling SIP-based communications through a packet filter again, such a mechanism could be realized by extending the SDP messages contained in some SIP messages during session establishment by some additional fields as proposed in [1]. These SDP fields actually represent some preconditions, that must be met, before the called party is explicitly notified about the request to establish a new session. If the called party receives for example an INVITE request without the precondition being met, it replies with a '183 Session Progress' response, including the parameters that will be used for its side of the data flow, but without beginning to ring. This response subsequently may be analyzed by an intermediate proxy server, which afterwards is aware of all parameters specifying the data flow, so that it is able to open the corresponding pinholes in the packet filter, or at least able to decide, whether it is possible to open them or not. Thereupon, the proxy server may send another INVITE or UPDATE request [4] to the called party, this time indicating that the dynamic configuration of the packet filter does not introduce any problems, so that the end user can be notified about the wish to establish a new session and the session setup procedure can proceed as usual.

B. Improper Session Release

As the dynamic configuration of a middlebox is indubitably a highly sensitive issue, it has to be assured, that all policy rules in a middlebox are removed again as soon as the corresponding session, for which they have been set up, is terminated by one of the involved peers. This is important, as the proxy server, which is responsible for the middlebox configuration, has to allocate resources and to store session-related data for each individual session, thus representing a potential target for denial of service attacks, for example. Furthermore, particularly in case of a packet filter, all pinholes which are no more needed by active sessions should be closed again in order to limit the number of open ports to the absolutely required minimum and to impede the sending of unauthorized traffic.

However, there is a potential problem concerning this matter, as there might be some situations in which the proxy server is not able to detect the end of a session by just analyzing the messages exchanged between the involved peers. This holds for example for situations, when a session is released improperly, due to a non-standard-conformant behavior of the respective end devices (e. g., operating system crash). In addition to that, potential attackers might precipitate such a situation intentionally. For these reasons, the proxy server, which is responsible for the dynamic configuration of the middlebox, should be able to determine on its own, whether a certain session is still active or not. For that purpose, the application layer protocol, which is used for the control flow, has to support some special protocol mechanisms. By and large, there are two different alternatives for realizing such mechanisms:

1. Specific message types could be specified, being at the disposal of a proxy server for actively enquiring the involved peers about the status of a particular session.
2. Some kind of keep-alive mechanisms could be introduced, i. e. that the peers participating in a session would have to periodically exchange some refresh messages as long as the session is active.

Basically, it is quite obvious, that the first approach, namely the specification of dedicated control messages, is in some ways advantageous to using some keep-alive mechanism, as that way, the proxy server is in the position to ascertain at any time whether a session is still active or not. If making use of the second approach, this would only be feasible after a certain period elapsed without any refresh message being sent. However, if an existing application layer protocol should be made compliant to the MIDCOM approach, it is presumably easier to extend the existing protocol by some keep-alive mechanisms than to introduce completely new message types, as this in most cases would entail some incompatibility problems with older entities being not aware of these new message types. This also holds for SIP, for example.

The base specification of SIP itself does not know neither any keep-alive mechanisms nor some special messages for actively querying the status of a SIP session. Therefore, again some protocol extensions have to be introduced in order to enable a proper application of the MIDCOM approach. One possibility for such an extension are so-called *SIP session timers*, as suggested in [2]. Basically, an SIP session timer is nothing else than an additional header field, specifying the time period after which a refresh message (e. g. a new INVITE request or an UPDATE request) has to be sent to the respective other peer as long as the session is alive. If a scheduled keep-alive message is missing, the proxy server subsequently could close any pinhole associated with the corresponding session again, without having seen any BYE message.

C. Encryption and Digital Signatures

Frequently, users want to establish a secure communication path between each other, in order to make eavesdropping or unauthorized data manipulation more difficult. In this context, the term *'secure'* normally refers to confidentiality, authentication and data integrity. In general, confidentiality can be achieved by encrypting messages, whereas authentication and data integrity can be realized by means of digital signatures. However, these mechanisms also represent potential problems with respect to the interworking between application layer protocols making use of such mechanisms and the MIDCOM approach: On the one hand, an intermediate MIDCOM proxy must be able to analyze the control flow of a session, what is generally not feasible if the corresponding messages are encrypted. On the other hand, a MIDCOM proxy sometimes also has to modify these parameters—for example if it is responsible for configuring a NAT—, whereupon the digital signature of this message would become invalid, so that the whole message probably would be discarded by the actual recipient.

One possible solution for dealing with this problem was proposed in [10]—especially for the case of SIP. The basic idea behind this solution is that a user is able to explicitly authorize another device to encrypt and sign messages on behalf of him. If, for example, a MIDCOM proxy has to be passed in order to configure a middlebox dynamically, the respective user could first of all encrypt all messages using the public key of that proxy and sign them with his own private key. The messages then are sent to the proxy server together with a corresponding authorization, which entitles the proxy to encrypt and sign messages on behalf of the user. Consequently, this way, the proxy server is able to analyze and—if applicable—modify the messages, as it can decrypt them using its private key. After taking adequate action, the proxy encrypts the (modified) messages again using the public key of the actual recipient and signs them using its own private key. Of course, this procedure can be applied recursively, i. e. a proxy server could encrypt and sign messages on behalf of a user and forward them to another proxy, which then could encrypt and sign them on behalf of the first proxy and so on. However, a crucial point of this approach certainly is that one can fully trust any intermediate MIDCOM proxy.

In order to be able to apply this solution in a proper way, an application layer protocol should support at least the following mechanisms:

1. A MIDCOM proxy should be able to explicitly request permission from a user or another proxy to encrypt and sign messages on behalf of him.
2. An end device or a proxy server should have some means for authorizing a MIDCOM proxy to encrypt and sign messages on behalf of him. This authorization must be checkable by any following device. In [10], for instance, a special SIP header is introduced for that purpose, specifying the proxy server which is entitled to act on behalf of a certain user. This header then is signed by the user/proxy granting this permission and attached to the original message.
3. It should be possible to inform the sender of a message about potentially arising problems concerning the delegated encryption and signing of messages, so that the user can decide in which way he wants to react in such situations. Appropriate reactions could be to try another route through the network or to send the messages unencrypted, for instance.

D. Further Issues

Another prerequisite for a proper application of the MIDCOM approach, which has not been mentioned so far but which basically is more or less self-evident, is the required support of proxy servers by the application layer protocol used for the control flow of a session. Theoretically, of course, it would always be feasible to establish some workaround, for example by installing a transparent proxy server, but such a kluge is definitely not an ideal and desirable solution and therefore should be avoided if possible.

V. Summary and Conclusions

In this paper, we have shown why some session-oriented services consisting of a control flow and one or more data flows frequently cause problems in conjunction with middleboxes, such as packet filters or NATs. These problems mainly originate from the fact, that the connection parameters specifying the end points of the data flows are dynamically negotiated by means of the control flow.

In a first step, the general solution approaches for dealing with these issues were presented and shortly evaluated with respect to their suitability. Afterwards, especially the so-called MIDCOM approach was outlined a little bit more in detail. The basic idea of this approach is to configure a middlebox on the fly, what could be done under control of an intermediate proxy server, for example. Subsequently, we pointed out some potential problems concerning the interworking between the application layer protocol used for the control flow of a session and the MIDCOM architecture. As was shown, for a proper application of the MIDCOM approach, the respective protocol should support at least the following mechanisms:

- The protocol has to be 'proxy-friendly'
- Some specific message types or other protocol elements are required, for being able to notify the end devices in case of a failed configuration of the middlebox appropriately.
- A certain user should only be notified about the desire to establish a new session, if it can be ensured, that the dynamic configuration of the middlebox can be carried out successfully.
- There must be either some keep-alive mechanisms or some special message types, which can be used by an intermediate proxy server to detect whether a session is still active or not.
- In order to be able to cope with encryption of messages and digital signatures, it should be possible to authorize MIDCOM proxies to encrypt and sign messages on behalf of certain users or other proxy servers. This includes mechanisms for requesting, granting and checking such permissions.

These issues should always be taken into account when designing new application layer protocols showing the characteristics as listed in the introduction, or when extending already existing ones in order to make them fully interoperable with the MIDCOM approach.

References

[1] G. Camarillo, B. Marshall, and J. Rosenberg. Integration of Resource Management and Session Initiation Protocol (SIP). RFC 3312. IETF, October 2002.

[2] S. Donovan and J. Rosenberg. Session Timers in the Session Initiation Protocol (SIP). Internet Draft (work in progress). IETF, January 2004.

[3] M. Barnes (Editor). Middlebox Communications (MIDCOM) Protocol Evaluation. Internet Draft (work in progress). IETF, November 2002.

[4] J. Rosenberg. The Session Initiation Protocol (SIP) UPDATE Method. RFC 3311. IETF, September 2002.

[5] H. Schulzrinne, D. R. Oran, and G. Camarillo. The Reason Header Field for the Session Initiation Protocol (SIP). RFC 3326. IETF, December 2002.

[6] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan. Middlebox Communication Architecture and Framework. RFC 3303. IETF, August 2002.

[7] M. Stiemerling, J. Quittek, and C. Cadar. Simple Middlebox Configuration (SIMCO) Protocol Version 3.0. Internet Draft (work in progress). IETF, January 2004.

[8] M. Stiemerling, J. Quittek, and T. Taylor. MIDCOM Protocol Semantics. Internet Draft (work in progress). IETF, January 2004.

[9] R. Swale, P. Sijben, P. Mart, S. Brim, and M. Shore. Middlebox Communications (MIDCOM) Protocol Requirements. RFC 3304. IETF, August 2002.

[10] K. Umschaden, J. Stadler, and I. Miladinovic. End-to-end Security for Firewall/NAT Traversal within the Session Initiation Protocol (SIP). Internet Draft (work in progress). IETF, May 2003.