# Some Notes on Security in the Service Location Protocol Version 2 (SLPv2)

Marco Vettorello, Christian Bettstetter, and Christian Schwingenschlögl
Technische Universität München (TUM), Institute of Communication Networks, Munich, Germany
*Email: Marco.Vettorello@gmx.it, Christian.Bettstetter@ei.tum.de, Christian.Schwingenschloegl@ei.tum.de*

**Abstract.** Service discovery protocols help users of a communication network to find services, applications, and devices that are available in the network. This feature is especially useful for mobile users in foreign networks and for groups of users that form a spontaneous (ad hoc) wireless network. As the demand for service discovery is growing, security is becoming an important concern. This paper discusses security in the IETF Service Location Protocol version 2 (SLPv2). We comment on a few security leaks in SLP and propose simple modifications in the SLP specification that make the protocol more resistant against replay attacks. Our method stores the timestamps received in service registration and deregistration messages. Furthermore, we suggest to change authentication keys after a service deregistration. Our SLPv2 implementation (TUM–SLP) demonstrates the fundamental service discovery features as well as authentication mechanisms based on DSA signature verification.

**Keywords.** Service discovery protocols, service location protocol, authentication, security, ad hoc networking.

## 1 Introduction

Service discovery protocols enable users of communication networks to find services, applications, and devices that are available in the network. In particular, they allow to search or browse for a service that fulfills a certain task. This feature is especially useful for mobile users (with notebooks, network–enabled PDAs, or mobile phones) in a foreign network and for groups of users that form a spontaneous (ad hoc) wireless network.

In such an environment, services automatically advertise themselves supplying details about their features and information required to access them. Users or applications can locate a service by asking for a particular service type (e.g. printer) and may make an intelligent service selection in case multiple services of the desired type are available. Besides the added value for the user, service discovery protocols also remarkably reduce the network administration load, especially when new services are to be introduced in a large network.

The following example illustrates the usefulness of service discovery: A business man is traveling on a plane. There are several facilities installed on the plane, such as a printer, a scanner, a fax machine, and a network access point to the Internet. If the business man likes to print out some handouts or slides for his presentation in the afternoon, he will first have to find out what type of printer is present, then install the device drivers to be able to print. If a service discovery protocol is installed, it will automatically detect and configure these services, relieving the user from any manual setup operation. Service discovery also provides useful characteristics of the discovered services, e.g. printer resolution and color capabilities.

Examples for service discovery protocols include the Service Location Protocol (SLP) [1][2], Jini [3], Universal Plug and Play (UPnP) [4], and Bluetooth's SDP [5].

This paper considers SLP version 2 [6], discusses its security features, and introduces our implementation TUM–SLP. After an overview of basic functionality of the SLPv2 protocol (Section 2), we explain its security mechanisms (Section 3). In Section 4 we discuss possible replay attacks against SLP. For example, we consider that an attacker eavesdrops on an SLP message and is able to change attributes of the service at a later time or even fake the identity of a particular service. We then propose extentions to the protocol that make SLP more robust against these attacks. Finally, in Section 5, we present our TUM–SLPv2 implementation. Section 6 concludes this paper.

## 2 Service Location Protocol Version 2 (SLPv2)

The service location protocol (SLP) has been designed and developed by the SrvLoc working group of the Internet Engineering Task Force (IETF). The specification of version 2 of the protocol can be found in [6]. It is designed for TCP/IP networks, and scales from small networks with only a few devices up to large enterprise networks. The protocol defines three agents that process SLP information: User Agents (UA) are searching for a service on behalf of the user or application; Service Agents (SA) are entities providing the location and description of a service; and Directory Agents (DA) work as a central repository for SLP information.

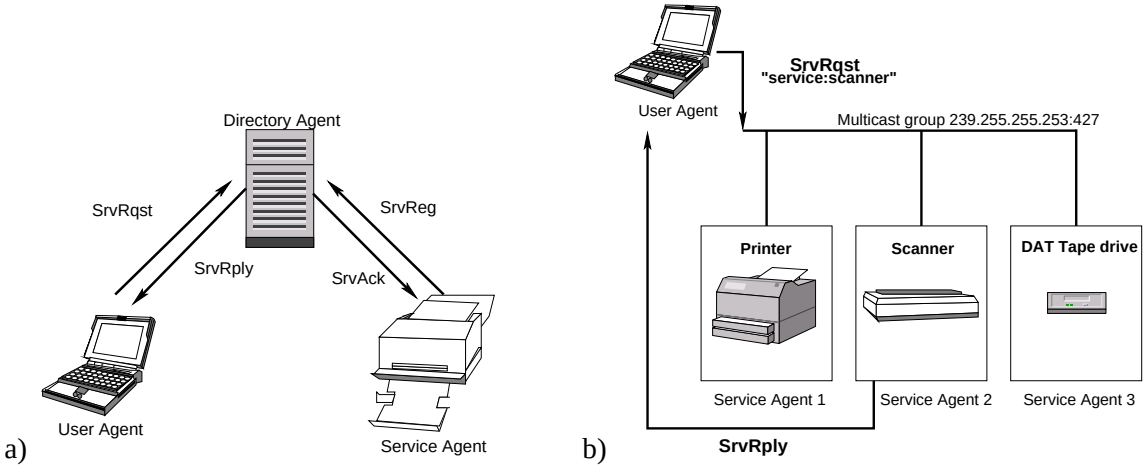


Figure 1: SLP architecture: (a) with DA, and (b) without DA

Figure 1a illustrates the basic functionality of the protocol. The SA, which is e.g. located inside a printer, registers the device with the DA. During a registration process, the SA informs the DA about the URL (Uniform Resource Locator) and a set of attributes that describe the service (*SrvReg* message). If a user, represented by the UA running on the notebook, wishes to discover a printer, he or she can send a service request (*SrvRqst*) to the DA. Optionally, he or she can specify some attributes that the service should have (e.g. color printer). If the DA finds a service in its repository that matches the request, it returns a *SrvRply* message that informs the user about the URLs of the found services. SAs can deregister their entry using a *SrvDereg* message.

Since the address of the DA is not available to any other agent, the SA and UA can perform active DA discovery. They multicast a request specifying *service:directory-agent* as the desired service to the SLP multicast group address 239.255.255.253 on port 427 (both specified in [6]). All DAs in the network listen to this port and reply with a unicast *DAAdvert* message to the requesting agent. DAs also advertise themselves by sending unsolicited *DAAdverts* to announce their presence. This discovery method is called passive DA discovery, and

SAs and UAs can extract the DA's address from the *DAAdvert*, as they do when performing active DA discovery. The third method to obtain the address of a DA is through static discovery via DHCP (Dynamic Host Configuration Protocol [7][8]).

Like most of the other service discovery protocols, SLP is administratively scoped. This means that the protocol locates resources (devices and services) available in a network within an administratively defined network domain. Users belonging to a certain scope may only discover services that are offered in this scope; other services are hidden by the DA.

As an optional feature, SLP allows users to request the attributes of a certain service or a type of service. To do so, a UA can send out an attribute request message (*AttrRqst*). A query of the attributes of a specific service is done via the service URL. If a UA sends an *AttrRqst* by service type, the DA returns all attributes of all services of the requested type in the scope. The latter feature is particularly interesting if a user wants to browse through the services.

The system architecture shown in Figure 1a is not the only possibility to deploy SLP in a network. In fact, a DA is only needed in large networks. When a DA is not present, UAs can apply active SA discovery in the same way they discover DAs. The UAs receive a *SAAdvert* in reply. Furthermore, UAs can send their *SrvRqsts* to the SLP multicast group. If a certain SA supports the service specified in the query, it will send a unicast *SrvRply* back to the requesting agent. SAs that do not advertise the requested service discard the *SrvRqst* message and continue to listen to the multicast address for incoming requests. Figure 1b shows such a scenario, in which the user is searching for a scanner. He or she sends a *SrvRqst* to the multicast group specifying *service:scanner* as the desired service type. SA2 supports the scanner and responds unicasting a *SrvRply* to the UA. As the DAs in the previous scenario, SAs can periodically multicast *SAAdverts* to inform UAs about their presence in the network. In this paper, we do not consider this mode of operation but assume that a DA is present.

# 3 SLP Security

As we have seen in the previous section, service discovery means that users may use services in foreign networks. This obviously creates immense security problems for both the mobile user and the network owner. In this section, we describe the inherent authentication features of SLPv2.

## 3.1 Authentication using Digital Signatures

Authentication is an optional feature in SLPv2. It is based on public key cryptography and enables the protocol to guarantee that the received service information has been transmitted by trustworthy SAs and DAs. The sender of an SLP message includes a digital signature, which is calculated over selected parts of SLP messages. Signatures are generated for URL entries, attribute lists, *SAAdverts*, and *DAAdverts* [6].

The trust relationship between the DA and the SAs is established by the network administrator. He or she sets up the services in the network and supplies them with the correct public and private keys. This ensures the mutual trustworthiness of both agents.

The trust relationship between the DA and the UAs involves two main problems: First, on the DA's side, the question is how to make sure that UAs do not access restricted services. This task must be solved by a higher level protocol and is not a duty of SLP. Second, the problem for the UA is how to ensure that the DA is trustworthy. To eliminate the risk that a UA communicates with a bogus DA, it is necessary to supply the UA with the public keys of the legitimate DA in a secure way. With a correct public key of the DA, the UA can discard incorrectly signed *DAAdverts*.

Let us give an example: An SA includes a digital signature in its *SrvReg* messages. A DA then verifies the signature before registering or deregistering any service information coming from this SA. The SA generates the

signature with its private key, and the DA can check the signature with the public key of the SA. Furthermore, the DA includes the signature of the respective SA in *SrvRply* messages to UAs. With this principle, UAs can be sure that replies to *SrvRqsts* come from trustworthy agents. Service information with incorrect signatures is discarded. We can request e.g. a service of type *service:ldap* and trust that the service URLs obtained in the *SrvRply* message are trustworthy LDAP servers.

### 3.2 Authentication Block

The digital signature is placed in an authentication block that contains additional information needed to verify the signature. The authentication block has the format shown in Figure 2.

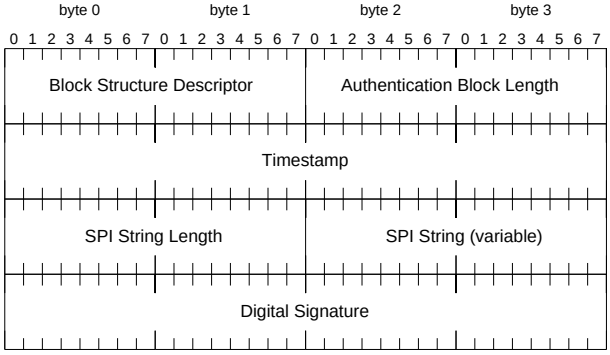


Figure 2: SLPv2 authentication block

The last field in the block contains the signature. The Block Structure Descriptor (BSD) identifies the algorithm that was used to calculate the signature. The default digital signature algorithm is the Digital Signature Algorithm (DSA) [9] with Secure Hash Algorithm 1 (SHA–1) [10], although other algorithms may be used. A timestamp informs the receiver when the signature expires. The Security Parameter Index (SPI) identifies the keying material and algorithm parameters for authentication. With the SPI and the BSD, the receiving agent has enough information to verify the signature. In the next sections, the generation and verification of the signature are described in more detail.

### 3.3 Signature Generation

As mentioned above, authentication blocks are included in *SAAdverts* and *DAAdverts* and whenever a URL entry or an attribute list is present in a message. To give an example, Figure 3 shows the relevant bytes used to authenticate a URL entry. This sequence of bytes is first processed by the SHA–1 algorithm (see Fig. 4), which creates a hash of the data. Applying the DSA algorithm, the signature generator then calculates the signature over the hashed bytes. Finally, the signature is ready to be included in the authentication block of Fig. 2.

### 3.4 Signature Verification

Upon receipt of an authentication block, the receiving agent derives from the BSD the algorithm to be applied to authenticate the signature. Subsequently, it checks whether the SPI in the block matches an SPI in its own list of supported SPIs. If the SPI is known by the agent, it can select the public key and the other parameters indicated by this SPI and then attempt to verify the signature. This involves two steps (see Fig. 5): First, it creates a hash over the relevant input bytes. Then, it decrypts the signature by applying the public key, thus obtaining the hash that the signing agent computed. If both hashes are equal, the signature has been verified.

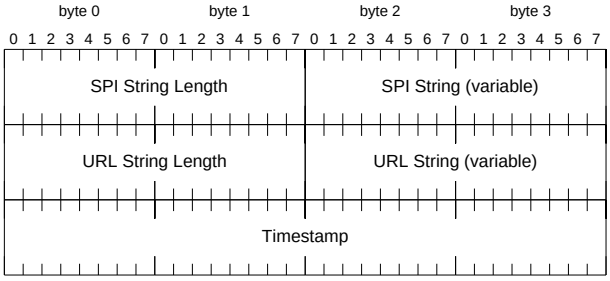


Figure 3: URL signature input bytes

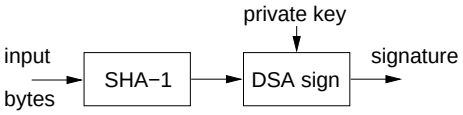


Figure 4: Signature generation

If the SPI in the block does not appear in its list of supported SPIs, an *Authentication_Unknown* error is returned. If the signature fails to verify, the agent returns an *Authentication_Failed*.

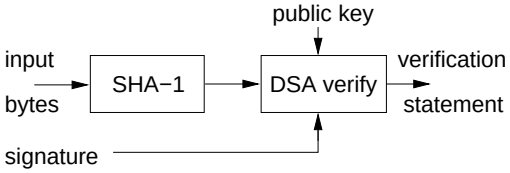


Figure 5: Signature verification

# 4 Discussion of SLP Security

Although the current SLP authentication mechanism offers basic protection against active security attacks, it has a few leaks that make the protocol vulnerable in certain situations. This section presents some realistic attack scenarios and proposes appropriate countermeasures.

## 4.1 Replay Attacks and Countermeasures

Many attacks against computer security are based on eavesdropping on messages during a transaction between a client and a server. A recorded message can be sent out by the attacker in a later moment. The attacker might thus be able to spoof a legitimate service, to impersonate it, and to fool other agents that do not realize its fake identity. In a service discovery scenario, a malicious device could e.g. record a message sent from an SA to a DA and replay it at a later time. Such replay or impersonation attacks represent a serious threat for SLP.

**Scenario 1** We consider the scenario illustrated in Figure 6: An SA deregisters a certain service and an intermediary attacker eavesdrops on the *SrvDereg* message. If the SA re–registers the service, the attacker can fake the identity of the SA and reissue the stored *SrvDereg.* The fooled DA successfully authenticates the message and deregisters the service.

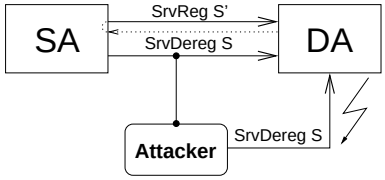


Figure 6: Replay attack, example 1

**Scenario 2** A harmful replay attack would also occur in the situation depicted in Figure 7: At time $t_1$ an SA registers the service $S$ whose lifetime expires at time $t_4$. An attacker eavesdrops on the *SrvReg* message and stores it. The network administrator decides to modify some attributes associated with $S$ and therefore re–registers the updated service at time $t_2$, which entirely replaces the entry for $S$. The attacker now replays the registration made by the SA at $t_1$. If the lifetime of $S$ has not expired yet (i.e., $t_4$ is still in the future), the attacker's message replaces the legitimate registration made at $t_2$.

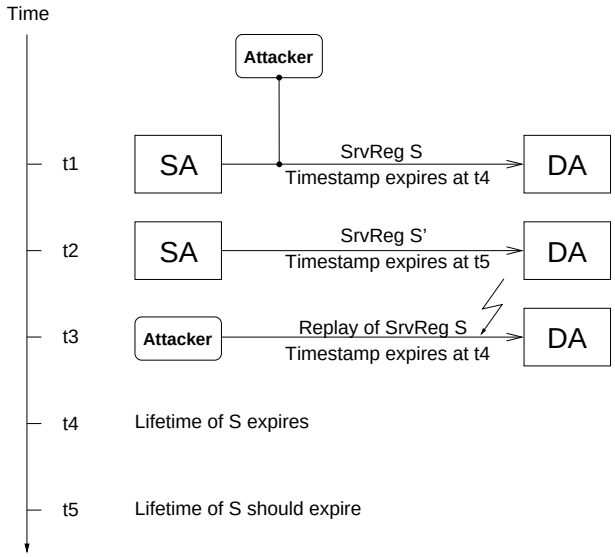


Figure 7: Replay attack, example 2

**Scenario 3** The scenario in Figure 8 is slightly different from the previous example. An SA registers a printing service $S$ at time $t_1$ that expires at time $t_{10}$. The attacker eavesdrops on this *SrvReg* and caches it. At time $t_5$ the administrator decides to shut down the service and deregisters $S$. The attacker has now enough time to impersonate the SA (e.g. through IP spoofing). During the time interval $t_5 \ldots t_{10}$ he or she receives all documents that users send to the pretended printing service.

**Proposed Security Features** The risk for the described replay attacks can be limited by simple modifications in the SLP specification. We propose the following principle: The DA caches the timestamps of received authentication blocks. To send a new message, an SA always uses a timestamp $t''$ that is larger than the timestamp included in the previous message ($t'$). The DA compares the timestamp of a received message ($t''$) with the old (cached) timestamp ($t'$) and accepts only messages with $t'' > t'$. Since the timestamp in a *SrvDereg* is set to the current time and cannot be larger than the expiration time of the advertisement, two

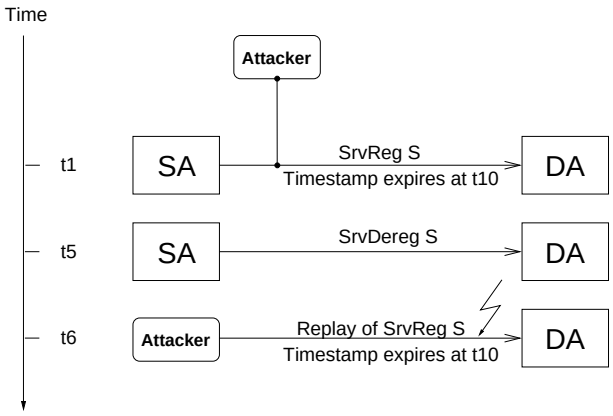


Figure 8: Impersonation attack

separate timestamp caches are used for *SrvReg* and *SrvDereg* messages. The mentioned replay attacks can be driven back with these methods, since the smaller timestamp in the replayed messages causes the DA to reject them.

A problem occurs if an SA erroneously sends a *SrvReg* with an unreasonable long lifetime, and an attacker eavesdrops on this message. By employing the discussed method, there is no possibility to set the lifetime back to a reasonable value. The administrator could perform a manual change in the DA to set back the cached timestamp, but this would not hinder the attacker to replay the recorded *SrvReg* at a later time. A solution to this problem would be to let the SA change (or even delete) its public/private key pair after the *SrvDereg*. If after a *SrvDereg* performed by the legitimate SA the attacker issues the eavesdropped *SrvReg*, the DA will not authenticate it. This method could also be adopted to reject the impersonation attack of Scenario 3.

## 4.2 Denial of Service Attacks

The proposed countermeasures can provide efficient defense mechanisms against the presented replay attacks. However, another sort of attack is still possible. Imagine a situation where the attacker modifies the timestamp in an authentication block of an eavesdropped *SrvReg* and sets it arbitrarily larger in order to have the DA accepting it. The DA would not discard the *SrvReg* straight away, rather it would attempt to verify the signature. Although the message will be discarded, the DA consumes processing resources to verify the signature. During this process it is inhibited from responding to other requests.

## 4.3 How much security does SLP need?

We can say that SLP security is for authenticating service advertisement messages and not the services that are advertised. Furthermore, it is a property of the network rather than of one single agent.

When security is enabled, important limitations are imposed on how a DA handles the reply to a request by a UA. Since the DA does not process the authentication blocks accompanying a *SrvReg* in any way, it must include exactly the same bytes submitted by the SA during the registration. The UA calculates a hash on the raw bytes of the message, thus the blocks would be invalidated if there are any differences in case or ordering. For this reason, when security is enabled and a UA performs an *AttrRqst*, no attribute must be specified in the query, that is to say the full list of attributes must be returned by the DA to avoid the invalidation of the blocks.

The implementation of security mechanisms adds a considerable amount of complexity to the protocol. One might even take the position that authentication may invalidate the claim of SLP to reduce administrative

workload, as a certain amount of overhead is required to establish key information trust and is a sort of disincentive to SLP deployment. On the other hand, if security is not implemented, the protocol will be exposed to several attacks that would not encourage the deployment of SLP. A good decision would be to always add security features to SLP implementations and allow network administrators to enable security if they feel it is necessary and worth the work of key delivery and related configuration.

# 5 Our SLPv2 Implementation (TUM–SLP)

The first beta release of our SLPv2 implementation, presented in [11], had the aim to demonstrate the fundamental functionality of SLP in action. The essential features were implemented in C and allowed basic transactions between the three agents over UDP sockets. The current second release (TUM–SLP v0.2) adds security mechanisms and service lifetime expiration ("leasing") to the protocol implementation.

The TUM–SLP implementation is based on an architecture that consists of one DA, one or more SAs, and one or more UAs. The presence of a DA is mandatory. All agents can be run on the same or on different Linux computers.

Once running, the DA announces its presence in the network by multicasting *DAAdverts* and then listens for messages from SAs and UAs. An SA can register a service using a *SrvReg* message. The description of the service is then stored in a file following the service template notation of SLP [12]. Updating, adding, or removing an attribute from the present list of attributes is also possible with *SrvReg.* Service deregistration is performed by *SrvDereg* messages. UAs can send *SrvRqsts* to the DA and specify the desired attribute. The optional messages *SrvTypeRqst* and *SrvTypeRply* are not implemented yet.

A sample output of a DA, upon receipt of a *SrvRqst* coming from an agent that performs active DA discovery, is the following:

```
*** SLP-packet received from 129.187.222.100:1181
Version:   2
Function Code:   1  ---->  SrvRqst
Message Length:   58
Flags:  O=0 F=1 R=1
Extension Offset:   0
XID:  26466
Language tag:   en
Previous responder string:
Service type string:   service:directory-agent
Scope list string:   lkn
Query string:
*** End of packet
Send DAAdvert to 129.187.222.100:1181
XID 26466
```

## 5.1 Security Features

SLP authentication can be activated by selecting the option `-s` on startup of an agent. If authentication is enabled, a UA can only perform a URL–based *AttrRqst* without specifying any attributes in the query. Furthermore, only removal of the entire service registration is allowed for SAs.

| DA | | SA | | UA | |
|---|---|---|---|---|---|
| SPI | Key | SPI | Key | SPI | Key |
| *marc* | private | *marc* | public | *marc* | public |
| *chris* | public | *chris* | private | *chris* | public |

Table 1: Sample agent configuration

The hashing of the data, the generation and verification of the signatures are performed by the Cryptlib v3 library [13].

All agents are pre–configured with SPIs that refer to public/private keys. Let us assume, the agents are configured as shown in Table 1. The SA calculates the signature with SPI *chris* and includes this SPI in the authentication block. The UA and the DA, having access to the public key associated with SPI *chris*, can verify the signature. On query, the UA specifies the SPI *marc* when it is looking for a DA and the SPI *chris* when it issues a *SrvRqst*. This leads the receiving agent to reply with blocks calculated with the same SPI as specified by the UA. Basically, the agents that generate signatures (DAs and SAs) store the private and public keys and parameters in a file. We store the keys generated by the DA and the SA in two distinct files: *da_keyset* and *sa_keyset*, respectively. Each private and public key in a file is identified by the SPI. When an agent receives an authentication block, it extracts the SPI and attempts to read the corresponding public key. The private keys are protected by a password, which is known only to the agent owning the file. Each agent should have access to both key set files: The DA must authenticate the messages it receives from the SA. The SA must authenticate the replies and *DAAdverts* it obtains from the DA. The UA must verify *DAAdverts* (signed by the DA) and the replies (containing the signatures originally calculated by the SA).

### 5.2 Interworking with K&A SLP

To evaluate the compliance of the TUM–SLP agents to the SLP standard, we made interworking tests between between TUM–SLP and K&A SLP (Kempf and Associates SLP, available at `http://www.srvloc.org`). The K&A SLP DA daemon is running on a Windows PC, while SAs and UAs have been started on Linux PCs. The two implementations successfully communicate over the SLP port 427, although ephemeral ports are used for replies. Figure 9 is an extract of the output of the K&A SLP DA daemon log file. It shows the repository of the DA daemon after a *SrvReg* from the TUM–SLP SA. The interworking of the security features of TUM–SLP could not be tested, since K&A SLP does currently not support authentication.

## 6 Conclusions and Outlook

The service location protocol SLPv2 enables automatic service discovery in dynamic networking scenarios. It offers much comfort for (mobile) users and network administrators. However, security is a critical (and yet unsolved) issue in these scenarios. In this paper, we have identified security leaks in SLP that make the protocol vulnerable against replay attacks. We proposed solutions that make the protocol less vulnerable. The presented TUM–SLP v0.2 is one of the first implementations to demonstrate the security mechanisms of SLPv2. Its source code will be released under GNU public license (GPL) in the near future. At the moment, we are developing a graphical user interface, which will add more user comfort to service searching and browsing.

A topic for further research is efficient key distribution in SLP. A potential approach to make key distribution easier, could be to have SLP itself distribute the keys to all agents, maybe by introducing a new message type which allows to configure the security parameters of an agent.

```
Fri May 11 12:10:56 2001:From client ‘‘129.187.222.179’’ on UDP interface:port (129.187.222.168:427):
Registering service URL:service:printer:lpr://129.187.222.129/
Fri May 11 12:10:56 2001:  #****** Service Table Dump Start *******
Fri May 11 12:10:56 2001:  #Remaining Time:2000
Fri May 11 12:10:56 2001:  service:printer:lpr://129.187.222.129/,en,2000,service:printer
Fri May 11 12:10:56 2001:  scopes=lkn
Fri May 11 12:10:56 2001:  printer-uri-supported=lpr://lkn.ei.tum.de
Fri May 11 12:10:56 2001:  uri-authentication-supported=none
Fri May 11 12:10:56 2001:  uri-security-supported=none
Fri May 11 12:10:56 2001:  printer-name=lj4050
Fri May 11 12:10:56 2001:  printer-location=Mobile Communications Lab
Fri May 11 12:10:56 2001:  printer-info=B/W laser printer
Fri May 11 12:10:56 2001:  printer-make-and-model=HP LaserJet 4050 N
Fri May 11 12:10:56 2001:  natural-language-configured=en
Fri May 11 12:10:56 2001:  natural-language-supported=en
Fri May 11 12:10:56 2001:  printer-document-format-supported=unknown
Fri May 11 12:10:56 2001:  compression-supported=none
Fri May 11 12:10:56 2001:  #******Service Table Dump End *******
Fri May 11 12:10:56 2001:From client ‘‘129.187.222.179’’ on UDP interface:port (129.187.222.168:427):
Return message is: Version:2  Function code:5  Length:18  Flags:0x0  XID:0xcd48  Option  offset:0
Language tag:en  Character encoding:UTF8  Status code:0  Previous responders:
```

Figure 9: Interworking: TUM–SLP and K&A SLP


### Acknowledgement

The authors would like to thank Erik Guttman for the discussion with him about SLP vulnerability.


## References


[1] E. Guttman, “Service location protocol: Automatic discovery of IP network services,” *IEEE Internet Computing*, pp. 71–80, July 1999.

[2] J. Kempf and P. S. Pierre, *Service Location Protocol for enterprise networks*. Wiley, 1999.

[3] “The community resource for Jini technology.” http://www.jini.org, 2001.

[4] “Universal plug and play forum.” http://www.upnp.org, 2001.

[5] “Bluetooth specification (version 1.1) - core, part E: Service discovery protocol (SDP),” 2000.

[6] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service location protocol, version 2 (IETF RFC 2608),” June 1999.

[7] R. Droms, “Automated configuration of TCP/IP with DHCP,” *IEEE Internet Computing*, pp. 45–53, July 1999.

[8] C. Perkins and E. Guttman, “DHCP options for service location protocol (IETF RFC 2610),” June 1999.

[9] *Digital Signature Standard*. NIST, 2000. FIPS PUB 186-2.

[10] *Secure Hash Standard*. NIST, 1995. FIPS PUB 180-1.

[11] C. Bettstetter and C. Renner, “A comparison of service discovery protocols and implementation of the service location protocol,” in *Proceedings EUNICE 2000*, (Twente, Netherlands), Sept. 2000.

[12] E. Guttman, C. Perkins, and J. Kempf, “Service templates and service: Schemes (IETF RFC 2609),” June 1999.

[13] P. Gutmann, *Cryptlib v3.0*. http://www.cs. auckland.ac.nz/˜pgut001/cryptlib, 2000.