Technical Report
KN–2014–DiSy–003

Distributed System Laboratory

Adding Privacy to Multicast DNS Service Discovery

Daniel Kaiser **Marcel Waldvogel**
Distributed Systems Laboratory
Department of Computer and Information Science
University of Konstanz – Germany

Abstract. Multicast DNS Service Discovery (mDNS-SD), made fashionable through Apple's *Bonjour*, is a prevalent technique allowing service distribution and discovery in local networks without configuration (Zeroconf). Possible application areas are device synchronization, instant messaging, VoIP, file and screen sharing. It is very convenient for users, because they can connect to and offer services when they enter a network without any manual configuration. However, it requires the public exposure of the offering and requesting identities along with information about the offered and requested services, even when services do not need to be public. Some of the information published by the announcements can be very revealing, including complete lists of family members. In this paper we discuss the privacy problems arising when using mDNS-SD and present our privacy extension, which allows hiding all information published while still not requiring any network configuration except for an initial pairing. A key feature of our solution is the ease of upgrading existing systems, a must for widespread deployment and acceptance. To show the feasibility of our mDNS-SD privacy extension, we developed an implementation based on the open-source *Avahi* daemon.

Table of Contents

1 Introduction

Multicast DNS Service Discovery (mDNS-SD) is a prevalent technique widely used to distribute services in local networks without configuration. It uses the upper two layers of the Zeroconf stack[1], namely, DNS Service Discovery[2] built on Multicast DNS[3], and brings great user experience. For example it allows a student who enters his campus network to automatically connect his mobile devices (smartphone, tablet, notebook) to each other, allowing file sharing and synchronization; it further allows to automatically connect to friend's devices on campus allowing to chat or share data and to connect to infrastructure devices like printers. This works seamlessly, without user interaction and regardless of the IP addresses and ports the corresponding services use.

Since Zeroconf is built on multicast, every machine in the same network will automatically receive all the announcement traffic and thus obtain a lot of information about the users in the network without having to send a single packet itself. Using mDNS-SD, devices publish their hostnames, commonly containing the user's name, when entering a network, followed by information about offered and requested services. When a user named Daniel enters the campus network, his Notebook publishes "Daniel's notebook joined the network" to all devices in the network. Many users are completely unaware of how chatty their devices are [4]. Most users do not consent to this information being published whenever they approach a McDonald's or Starbucks [4]. However, there is no user-accessible mechanism to limit or prevent this chattyness.

The device might further publish:

- "I want to sync Daniel's mobile folder with Daniel's smartphone."
- "I share the folders `/home/daniel/share`, `/home/alice/share`, and `/home/bob/share`."
- "I am online using *iChat* and my status is *gaming*."

Offering shares might allow inferring names of family members, furthermore opening pathways to social engineering attacks, while a chat application shows the user's activity status to *everyone* in the same network. Most users do not even know how much information is published via mDNS-SD every time they connect their mobile devices to a network or come close to a known WLAN access point.

In this paper, we present a privacy extension for Zeroconf daemons, which gives users the choice of what they want to have published without affecting their applications. Our mechanism

- adds privacy to mDNS-SD by hiding all probably private discovery data that has to be multicast,
- is transparent to client software using mDNS-SD,
- is transparent to the existing network infrastructure,
- allows automatic service discovery like standard mDNS-SD,
- is fully backward compatible,[1]
- is very efficient, in all of network traffic, memory consumption, CPU time, and wall clock time.

[1] Public services' operation remains entirely unchanged; private services *seem* entirely unchanged within the trusted group.

To grant these features, a relationship between an offered service and a service querier has to be established. This is done by an initial pairing during which a shared secret is exchanged. After this initial pairing, service discovery requires no further configuration. The IETF Zeroconf charter[2] states that minimal configuration is tolerated for security's sake; we assume this goes for privacy as well.

We grant privacy by substituting service instance keys, which only authorized hosts can understand, for service instance names and encrypting other possibly private data before transmitting. The service instance keys allow us to check if a packet is relevant or not in $\mathcal{O}(1)$ time, avoiding the attempt to decrypt irrelevant packets.

Our solution limits the uncontrolled application-layer distribution of private data. To identify a machine, its MAC identifiers and physical layer characteristics can be used. Solutions to that problem are outside the scope of this paper, but are tangentially discussed in section 6.

[2] http://datatracker.ietf.org/wg/Zeroconf/charter

2 MDNS-SD Privacy Problems

Multicast DNS Service Discovery uses DNS records that are queried by sending them to the multicast address 224.0.0.251 (IPv4) and FF02::FB (IPv6) and answered via multicast by hosts that offer a corresponding service instance. The discovery of a service can be divided into three stages:

Browsing for a service means asking for a list of services of requested types; this is done by asking for PTR records, which are used as indicator for the existence of an instance of a certain service type in addition to the usual reverse lookup query (see first query and response pair in Figure 1). When a querier is asking for the service type `_presence`, which is used by chat clients like iChat and Pidgin allowing chat in local networks, it sends a PTR query with the label `_presence._tcp.local`. This query is answered by all hosts that offer a service instance of type `_presence` by multicasting an answer typically containing the information that Figure 2(a) shows.[3]

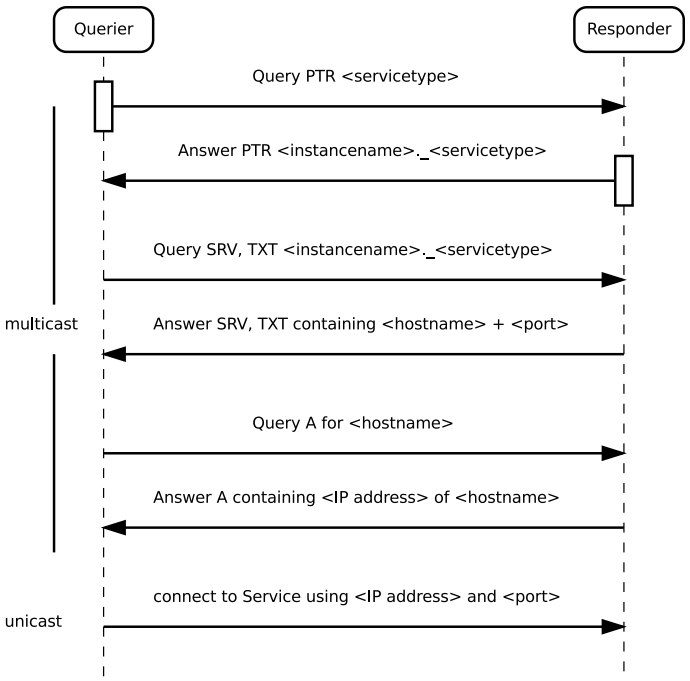

Fig. 1. Queries and responses sent using mDNS-SD.

There are two privacy problems arising from this. First, the name and type of the service instance can be seen by anyone listening to the network traffic. Secondly, anyone searching for a service of this type will be offered this service instance, even if its not meant for him. In the example anyone using the

[3] The figures containing resource records show the subset of the information as shown by Wireshark which is relevant for the purpose of showing the privacy problems.

```
_presence._tcp.local: type PTR,
  daniel@Daniel's Notebook._presence._tcp.local
```

(a) PTR record containing the device's hostname, which per default typically contains the user's name.

```
daniel@Daniel's Notebook._presence._tcp.local:
  type SRV,
  port 5298, target Daniel's Notebook.local
```

(b) SRV record showing the hostname and port.

```
daniel@Daniel's Notebook._presence._tcp.local:
  type TXT,
  vc=! ver=2.10.6 node=libpurple
  port.p2pj=5298 txtvers=1
  status=gaming
  last=Kaiser
  1st=Daniel
```

(c) TXT record that contains several critical key value pairs like the first and last name of the user, the chat status and the version of the service.

```
Daniel's Notebook.local: type A, addr 134.34.10.36
```

(d) A record presenting a mapping of hostname to IP address.

Fig. 2. Resource records multicast when using a chat application that is based on the `_presence` service. Each of these resource records violates privacy.

`_presence` chat service sees Daniel coming online. This is not a constructed example; most operating systems choose a default hostname which contains the username, and chat applications using the `_presence` service use the pattern `<username>@<hostname>` as chat alias. This problem is more severe than it might look at first glance; everyone in the same network can see everyone else coming online, even users that are not in the buddy list. It is not even necessary to use sniffing tools like Wireshark; using a simple service browser or a chat client like Pidgin suffices.

Resolving a service means getting information about the service to be able to connect to it. The resource records used for resolving are SRV and TXT (see second query and response pair in Figure 1). A host who has browsed for existing services can use the instance names gained from the PTR records to ask for the corresponding SRV and TXT records. The one host in the network who offers the requested service instance will answer by sending the requested SRV and TXT records. Example answers for the label `daniel@Daniel's Notebook._presence._tcp.local` are shown in Figures 2(b) and 2(c), respectively. Service resolving yields another privacy problem. Hostname, service name, and service type are contained in both SRV and TXT records; the SRV record further shows the port number the offered service uses. Since the port number is often a synonym for a service type, e.g. 22 for ssh, the port number is critical as well. The published port number may also yield a security problem; though not relevant for our chat example, it is a problem for protected services, allowing attackers to attack the service without the need of a portscan, which would render them suspicious on a network monitoring tool. The TXT resource record contains further information about the offered service instance in form of arbitrary key value pairs. These may contain a significant amount of private information accessible by anyone in the same network. The `_presence` service transmits first and last name in the TXT record, if they were entered during setup. A further problem arises due to the version number sent in the TXT record, allowing to identify hosts running a vulnerable version in order to attack them.

Asking for the IP address is the last step corresponding to the hostname gained from the SRV record, which is shown by the third pair of query and response in Figure 1. Figure 2(d) shows an example answer. Once again the hostname is publicly announced. Due to publishing the IP address, it is possible to get a mapping from hostname to IP address allowing to infer further information about a host. Even without offering any service instances the Zeroconf daemon immediately offers an A (and AAAA) resource record when entering a network. Depending on the configuration there is also an HINFO resource record published, which also contains the hostname plus information about the host's CPU and operating system.

3 Requirements

The following requirements should be met by an mDNS-SD privacy extension in order to be feasible.

Information hiding: All information published using mDNS-SD should only be accessible by paired devices. This includes the hostname, the service instance name and service type, the port, and TXT records.
Network efficiency: The addition of privacy to mDNS-SD should not cause a heavier network load than standard mDNS-SD.
Runtime efficiency: Hosts must be able to process incoming multicast queries and answers in $\mathcal{O}(1)$ and the constant calculations have to be efficient.

If those requirements are not met, the following attacker types can be successful (see Table 1).

Passive: The passive attacker wants to get as much information as possible by just listening to the multicast traffic. All plaintext information will be gained by this attacker type.
Active: An active attacker wants to get information by sending queries for services he is interested in. He might ask for all `_presence` service instances, extract the version numbers from the TXT records, identify the vulnerable versions and attack the corresponding hosts. He can also offer (fake) services to make someone connect.
DoS: This attacker wants to bring the host running mDNS-SD to a halt by flooding the network with multicast messages in the mDNS-SD multicast group. This attacker can be successful if the runtime efficiency or network load efficiency requirement is not met. A motivation for a DoS attack on mDNS-SD could just be considering it fun to drain the battery of many devices.

Unmet Requirement	Enables attack type		
	Passive	Active	DoS
Runtime Efficiency	–	–	✓
Network Efficiency	–	–	✓
Information Hiding	✓	✓	–

Table 1. Unmet requirements open pathways to attacks

Solutions that are based on encrypting whole packets and sending them without an application layer identifier are not feasible as they allow the DoS attacker to succeed. This is due to the fact that these solutions demand every incoming packet on the multicast DNS socket to be decrypted. Using *symmetric encryption*, each incoming packet has to be tested with keys corresponding to each paired service instance, requiring $\mathcal{O}(\#\text{pairings})$ time. Using *asymmetric encryption*, including some broadcast encryption mechanisms [5,6], the asymptotic run time goes down to $\mathcal{O}(1)$, yet the calculations needed for asymmetric encryption are very expensive. Assuming the sent packets contain 2048 bits of data encrypted using RSA, a DoS attacker can send roughly half a million such packets

per second on a gigabit Ethernet link, while a modern notebook or desktop processor can only decrypt about 600 packets in the same time[4]. A further drawback of these methods is the battery power wasted.

[4] as shown by `openssl speed rsa`

4 Privacy Preserving mDNS-SD

Our architecture realizes a privacy extension for mDNS-SD that meets the requirements stated in section 3.

4.1 Pairing

Pairing is the required process of establishing a relationship between a service instance and a querier by transmitting a static shared data structure called *privacy service data* that is created by the service offerer for the service it wants to offer privately and contains

- the service instance name,
- the service type,
- a service instance key used as substitution for the service instance name,
- cryptographic key material
- and additional data depending on the implementation.

The service data is stored in two hash tables; one indexed by the service instance name and one indexed by the service instance key. If this service has already been paired with another service querier, the service data creation and storing is omitted. The service data is transmitted to the service querier via a secure or out-of-band channel, such as Bluetooth, NFC, photographing a QR code, encrypted Email, or SMS. The service querier then saves the service data in his hash tables, which completes the pairing process. It is important for the service data to be stored in hash tables, because this allows $\mathcal{O}(1)$ access time. Pairing has to be done once per pair of a service instance and a requesting host. After pairing once no further configuration is necessary.

4.2 Hiding the Instance Name

When using mDNS-SD, the service instance name and service type allow the receiver of a packet to decide in constant time which service instance it is related to and if he is interested in this service. Since this service identifying data should be hidden from the public, we change the service instance name to the service instance key which is agreed upon during pairing. This is a simple solution for hiding the information while still guaranteeing $\mathcal{O}(1)$ lookup time, because when receiving a query or answer the receiver can look this key up in his local hash table. If the hash bucket the key refers to contains the service name, the receiver knows which service instance the query or answer was related to, if not, the packet is irrelevant for this receiver.

4.3 Hiding the Service Type

To hide the service type, we need some sort of *type keys* which only partners should be able to map to the corresponding service type. In order to meet the network efficiency requirement (see section 3), we want a single service instance to be represented by exactly one pair of service instance key and type key. The advantage of one single obfuscated representation corresponding to a single service instance is that important features[3] such as known answer suppression, duplicate query suppression and duplicate answer suppression work like they would

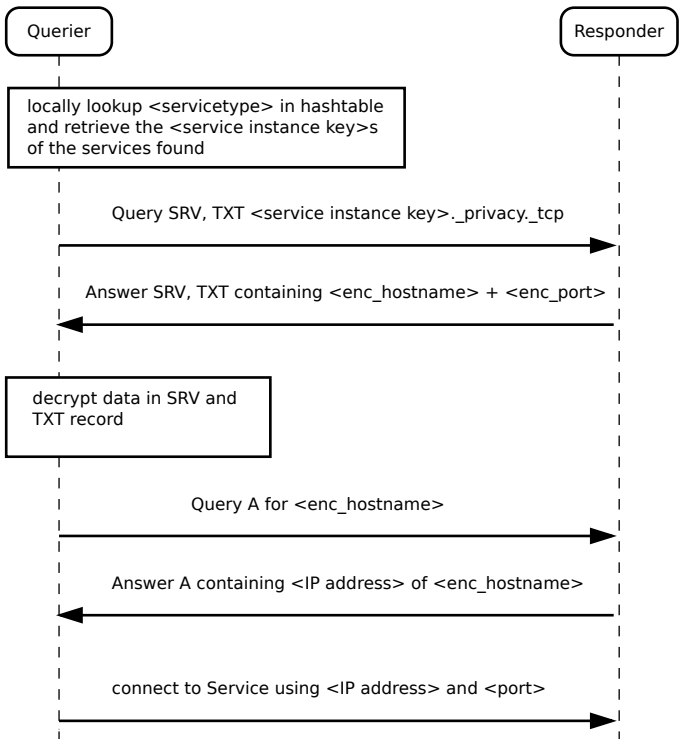

Fig. 3. Resolving a Service using our privacy extension (compare to the standard process shown in Figure 1). Instead of browsing for service instance names corresponding to a certain type, they are retrieved from a local hash table built during pairing. Private data is either substituted by random strings exchanged during pairing or encrypted.

without the privacy extension, because the mDNS-SD subsystem recognizes it as one service instance. Establishing this mapping for the service instance name is easy, because a service instance name is mapped to a single service instance offered by a single service offerer. This is different for the type. Each group of paired users[5] has to use the same type key, to allow for a single pair of service instance key and type key. This is unpractical for two reasons. On the one hand all users of such a relation had to agree on one type key, which is not possible using a simple, convenient pairing strategy; on the other hand it would allow all users in that potentially large group to know the service type in a query or answer of any of this group's members.

To solve this problem, we introduce the special service type `_privacy._tcp` for all privacy-enabled service instances. Thus the published service type has no relation to the real service type at all. This renders browsing for services in the usual way described in section 2 infeasible, because browsing for `_privacy._tcp` would return all privacy services; e.g. when using a chat client with a privacy aware presence protocol not only the chat buddies would pop up as friends, but all other privacy services as well. To solve this, we have to change the way privacy

[5] all users A and B, who are in the transitive closure of the relation *A is paired to B regarding a certain service instance S*

services are browsed. Browsing for a service type returns PTR resource records showing service instances of this service type (including their name), which can then be resolved. Since we already obtained this information during pairing and stored it in local hash tables, we do not have to browse for a service type using multicast at all. Our privacy browsing happens locally and the matching service instances are resolved directly. We call this "direct resolving" because it directly resolves a service instance without browsing for the corresponding type.

An advantage of direct resolving is the reduction of multicast packets, because no browsing queries and answers have to be sent. Additionally, the number of resolving queries is also reduced, because queries and answers are only sent for service instances a user can really use. Normally, a resolve query is sent for each PTR record received during browsing. Depending on the number of service instances offered, a significant number of resolving queries or answers can be avoided using direct resolving.

The `_privacy._tcp` type has another advantage. All services that are found using a service browser but are not meant for the browsing user are listed in a single section `_privacy` and thus the visual clutter caused by those inaccessible services can be easily hidden. A privacy extension aware service browser is also able to not show privacy service instances the user is not paired to, without any user interaction.

4.4 Querying and Responding

In order to hide published data from unauthorized hosts, all the privacy problems presented in section 2 have to be addressed. When a querier asks for a resource record, the query is altered before sending, substituting the service instance key for the service instance name and the service type for `_privacy._tcp`. Receivers that paired one of their service instances with the querier are able to interpret the request. Before sending the answer possibly private data in the resource record is encrypted, which is especially important for the key value pairs in TXT records. When receiving a packet the querier again recognizes it by looking up the service instance key in the hash table. Figure 3 shows the service discovery process using our privacy extension.

4.5 Implementation

We implemented our privacy extension based on the open source Zeroconf daemon Avahi[6]. We changed very little code in the avahi-daemon source files, just adding hooks to our loosely coupled privacy module before queries and answers are sent to and received from the client, respectively. When a client asks the daemon to discover a service, we update the packet by applying our privacy operations as described above and let the daemon handle the updated packet. We alter the packet in a way that it remains a valid MDNS-SD packet, making the daemon believe the client asked for a `_privacy` service instance. When getting the answer, the daemon will recognize it as being queried before, processes the answer and eventually wants to give this answer to the client; before this happens, we undo all privacy operations and present the service to the client. The client software does not need to know about the privacy subsystem at all

[6] `http://avahi.org`

and the daemon thinks the client wants to query and offer service instances of type `_privacy`, allowing us to get all benefits of the daemon. Because of the very limited code changes and the loose coupling, it is easy to merge updates from Avahi to our modified daemon.

5 Results

When using our privacy extension all data that has to be multicast to discover and offer services is transmitted in a privacy preserving way. Resource records multicast when publishing a `_presence` service instance on starting pidgin using our mDNS-SD privacy extension are shown in Figure 4. The service instance key is substituted for the service instance name, a shared random string is substituted for the hostname, key values pairs are encrypted, and the port is XORed with cryptographic key material exchanged during pairing. All the key value pairs in the TXT records are encrypted as one unit and then split to fit the maximum length. This is done to prevent information inferring based on the number and length of TXT record key value pairs. Figure 4(c) shows the encrypted TXT record. Figure 5(b) shows the output of `avahi-browse` [7] asking our modified daemon on a paired host for the service we published above. This shows that client software, in this case the command-line service browser `avahi-browse`, works with our privacy extension without being changed. When executing `avahi-browse` asking for the same service type on an unmodified daemon or a modified daemon of a unpaired host, the above published service instance will not be found. Browsing for the `_privacy` service type yields the result shown in Figure 5(b).

[7] We use `avahi-browse` do demonstrate data received at the host. Users do not have to bother using a service browser; an mDNS-SD capable application handles the service discovery transparently.

```
_privacy._tcp.local: type PTR,
  prvGK8cApytjxKRd._privacy._tcp.local
```

(a) PTR record hiding the service instance name. The **_privacy** service type is substituted for the **_presence** service type.

```
prvGK8cApytjxKRd._privacy._tcp.local: type SRV,
  port 8320, target prvq0XhpZA0zeYqa.local
```

(b) A privacy aware SRV record. The port is XORed with the key exchanged during pairing. The hostname is substituted for a randomly generated string that was also agreed upon during pairing.

```
prvGK8cApytjxKRd._privacy._tcp.local: type TXT,
Text: p01=hjl6ACDCtvEOWuU8cZ41wPrhbhGBcu/VAvbtzzBZO
  hJ5rkL3SGR2HMbciNuycVhtEEx3Blgl7mzqlvBGt0HWhW3o6u
  PrKx0LJJQ4/gpZhnJ9nWndY1FTnf531Ai
Text: p02=kou/ddoOGrpkRnyfH5ahbnueP5g==
```

(c) TXT record showing encrypted key value pairs. The key value pairs were concatenated before encrypting and split afterwards to avoid inferring of the service based on typical key value pair quantities and lengths.

```
prvq0XhpZA0zeYqa.local: type A, addr 134.34.10.36
```

(d) A record with hidden hostname. A mapping of hostnames containing user's names to IP addresses is no longer available.

Fig. 4. Resource records multicast by our privacy extension when using a chat client that is based on the **_presence** chat service. Critical data is substituted by identifiers randomly generated during pairing or it is encrypted.

```
eth0 IPv4
daniel@Daniel's Notebook       iChat Presence  local
hostname = [Daniel's Notebook.local]
address = [134.34.10.36]
port = [5298]
txt = ["vc=!" "ver=2.10.6" "node=libpurple"
  "status=gaming" "port.p2pj=5298"
  "last=Kaiser" "1st=Daniel" "txtvers=1"]
```

(a) Output of `avahi-browse -r _presence._tcp` on a paired host. The host gets all information about the `_presence` service instance published by a chat application of a friend.

```
eth0 IPv4
prvGK8cApytjxKRd                privacy         local
hostname = [prvq0XhpZA0zeYqa.local]
address = [134.34.10.36]
port = [8320]
txt = ["p01=hjl6ACDCtvEOWuU8cZ41wPrhbhGBcu
        /VAvbtzzBZOhJ5rkL3SGR2HMbciNuycVhtEEx3
        Blgl7mzqlvBGt0HWhW3o6uPrKx0LJJQ4/gpZhn
        J9nWndY1FTnf531Ai"
  "p02=kou/ddoOGrpkRnyfH5ahbnueP5g=="]
```

(b) Output of `avahi-browse -r _privacy._tcp` on a unauthorized host showing encrypted or substituted data. If the unauthorized host would browse for `_presence._tcp` it would not discover this service instance at all.

Fig. 5. Output of the command line service browser `avahi-browse` on a paired and unauthorized host, respectively. `avahi-browse` is used to show data received at the host.

6 Related Work

We are not aware of any publication regarding privacy extensions for mDNS-SD. Nevertheless, the problem of mDNS-SD publishing device names has been addressed [7,4].

Aura et al. [7] investigate private information published on different network layers when connecting devices to a network. While they mainly look at other protocols, they also mention the privacy problem of device names published by mDNS-SD. As a solution, they propose network location awareness, that is allowing service discovery only in trusted networks. We consider that too restrictive and want to give the user the possibility to request and offer services in a privacy preserving way even if the network cannot be trusted. Further we want to mitigate the configuration overhead of making assumptions about the trustworthiness of the network in use.

The public announcement of device names by mDNS-SD is discussed in more detail in [4]. The authors conducted studies showing that almost 60% of the published device names contain real user names and that 90% of the users consider this as privacy problem. They propose making users aware of the problem and changing hostnames as a solution and also refer to network location awareness and identifier free networks[8]. The privacy problem of SRV and TXT records being published is not mentioned. We consider it important to hide all possibly private information, because it can be seen by anyone as easily as the hostname and might deserve more privacy protection (e.g. TXT records). Furthermore, users should not have to change hostnames.

Much research has been done in the area of privacy in wireless networks. Especially finding access points in WiFi-networks is related to our paper, because it also has to solve the task of finding entities in a network using broadcast, where everybody in the vicinity can listen to the messages. [9] presents a privacy preserving protocol for discovering WiFi access points. Since it is an extension to the 802.11 MAC protocol, it does not meet our requirement of not changing any deeper protocol layers.

Greenstein et al. [8] present an identifier free wireless OSI layer 2 protocol, which allows to find access points and communicate in the wireless network in privacy. Since it provides privacy on the link layer, it also solves the private service discovery problem. But to allow private service discovery, the whole network infrastructure has to be updated, while our solution allows private service discovery by just updating the local Zeroconf daemon. Nevertheless, an identifier free network fulfills more privacy requirements than those stated in section 3. It allows to hide information about a certain device being online in a network, because there are no MAC addresses published; thus allowing users to be untrackable. While we consider it beneficial to be able to meet more privacy requirements, we also consider it important to give users the possibility to easily hide the cleartext information published, while being independent of the network infrastructure.

Generic protocols for privacy preserving service discovery [10] and presence sharing [11] have been proposed. Since they use a central entity [10] and depend on a trusted broker [11], respectively, we did not use them because we want to keep mDNS-SD decentralized and do not want to rely on trusted third parties.

There is much research in the area of service discovery in pervasive computing environments[12] and mobile ad hoc networks[13]. Zhu et al. present a model

for privacy preserving service discovery [14,15]. Since it is a generic model, we consider it unnecessarily complex for our application area. Nevertheless it is a sophisticated model and we want to incorporate ideas like the usage of bloom filters in our privacy extension in future work.

Mechanisms such as IPv6 privacy extensions[16] and locally administered MAC addresses also aim to reduce the ability of others to trace the whereabouts of end systems. These methods work independently of our privacy extension, on OSI layers 3 and 2, respectively, and thus can be used at the same time.

Device fingerprinting [17,18] can be used to track the devices, even if the protocol in use has no explicit identifiers. It is outside the scope of this paper to address tracing using information leaked by the physical layer [19] and tracing of the users by other 'metadata,' such as the number or rate of connections devices open, or where they connect to [20].

7 Conclusion and Future Work

In this paper we pointed out privacy problems in mDNS-SD and presented a privacy extension which hides all private information contained in published resource records, while being efficient and transparent. Figure 4 presents resource records published by our solution, showing that none of the cleartext strings are published, like mDNS-SD does without our extension (see Figure 2). It is efficient because incoming packets can be identified in $\mathcal{O}(1)$ whether being relevant, and thus reduce the efforts of even the efficient symmetric cryptographic operations we use. The major advantage of our extension to existing solutions (see section 6) is that none of the existing OSI layer protocols and none of the existing client software has to be altered. Only the Zeroconf daemon running on the users devices has to be modified, while afterwards still being able to exchange service information with unmodified daemons.

In the future, we plan to include two user friendly pairing methods: for users meeting each other, allowing them to pair their devices using NFC, and when they cannot easily meet. Our goal is to limit pairing to once per pair of users who want to share service instances instead of the current mode of once per pair of service instance and user. Since mDNS-SD has shown to cause significant load in huge networks[21], we also want to further reduce the number of multicast packets sent. Furthermore, we hope to improve the privacy protecting abilities such that the possibilities of inferring information is reduced, while still remaining compatible with standard mDNS-SD.

The daemon alone is not sufficient without a user-friendly service browser, extended to give users straightforward control which services they want to be published publicly, which services they want to be published in private, and which services they do not want to be published at all. Together they empower the users with mDNS-SD privacy and transparency.

References

1. D. H. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide*. O'Reilly, 2006. 1
2. S. Cheshire and M. Krochmal, *DNS-Based Service Discovery*, ser. Request for Comments. Internet Engineering Task Force (IETF), 2013, no. 6763. 1
3. ——, *Multicast DNS*, ser. Request for Comments. Internet Engineering Task Force (IETF), 2013, no. 6762. 1, 4.3
4. B. Konings, C. Bachmaier, F. Schaub, and M. Weber, "Device names in the wild: Investigating privacy risks of zero configuration networking," in *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*, vol. 2. IEEE, 2013, pp. 51–56. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6569062 1, 6
5. C. Delerablée, "Identity-based broadcast encryption with constant size ciphertexts and private keys," in *Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security*, ser. ASIACRYPT'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 200–215. [Online]. Available: http://dl.acm.org/citation.cfm?id=1781454.1781471 3
6. Q. Wu, B. Qin, L. Zhang, and J. Domingo-Ferrer, "Ad hoc broadcast encryption," in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 741–743. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866416 3
7. T. Aura, J. Lindqvist, M. Roe, and A. Mohammed, "Chattering laptops," in *Privacy Enhancing Technologies*. Springer, 2008, pp. 167–186. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-70630-4_11 6
8. B. Greenstein, D. McCoy, J. Pang, T. Kohno, S. Seshan, and D. Wetherall, "Improving wireless privacy with an identifier-free link layer protocol," in *MobiSys '08: 6th International Conference on Mobile Systems, Applications, and Services*, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1378600.1378607 6
9. J. Lindqvist, T. Aura, G. Danezis, T. Koponen, A. Myllyniemi, J. Mäki, and M. Roe, "Privacy-preserving 802.11 access-point discovery," in *Proceedings of the second ACM conference on Wireless network security*, ser. WiSec '09. New York, NY, USA: ACM, 2009, pp. 123–130. [Online]. Available: http://doi.acm.org/10.1145/1514274.1514293 6
10. S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1999, pp. 24–35. [Online]. Available: http://dl.acm.org/citation.cfm?id=313462 6
11. L. Cox, A. Dalton, and V. Marupadi, "Smokescreen: flexible privacy controls for presence-sharing," in *Proceedings of the 5th international conference on Mobile systems, applications and services*. ACM, 2007, pp. 233–245. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247688 6
12. F. Zhu, M. W. Mutka, and L. M. Ni, "Service discovery in pervasive computing environments," *IEEE Pervasive computing*, vol. 4, no. 4, pp. 81–90, 2005. 6
13. F. Outay, V. Vèque, and R. Bouallègue, "Survey of service discovery protocols and benefits of combining service and route discovery," *IJCSNS*, vol. 7, no. 11, p. 85, 2007. 6
14. F. Zhu, M. W. Mutka, and L. M. Ni, "A private, secure, and user-centric information exposure model for service discovery protocols," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 4, pp. 418–429, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1599409 6
15. F. Zhu, W. Zhu, M. W. Mutka, and L. M. Ni, "Private and secure service discovery via progressive and probabilistic exposure," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 11, pp. 1565–1577, 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4339200 6

16. T. Narten and R. Draves, *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*, ser. Request for Comments. Internet Engineering Task Force (IETF), January 2001, no. 3041. 6
17. T. Kohno, A. Broido, and k. claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, no. 2, pp. 93–108, May 2005. 6
18. L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee, "Identifying unique devices through wireless fingerprinting," in *Proceedings of the first ACM conference on Wireless network security*, ser. WiSec '08. New York, NY, USA: ACM, 2008, pp. 46–55. [Online]. Available: http://doi.acm.org/10.1145/1352533.1352542 6
19. K. Bauer, D. McCoy, B. Greenstein, D. Grunwald, and D. Sicker, "Physical layer attacks on unlinkability in wireless lans," 2009. 6
20. B. Greschbach, G. Kreitz, and S. Buchegger, "The devil is in the metadata - new privacy challenges in decentralised online social networks," in *Proceedings SESOC, PERCOM*, 2012. 6
21. S. Hong, S. Srinivasan, and H. Schulzrinne, "Measurements of multicast service discovery in a campus wireless network," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE.* IEEE, 2009, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5426121 7