Trusted Computing Serving an Anonymity Service

Alexander Böttcher **Bernhard Kauer** **Hermann Härtig**
Technische Universität Dresden
Department of Computer Science
Operating Systems Group
{boettcher, kauer, haertig}@os.inf.tu-dresden.de

Abstract

We leveraged trusted computing technology to counteract certain insider attacks. Furthermore, we show with one of the rare server based scenarios that an anonymity service can profit from trusted computing. We based our design on the Nizza Architecture [14] with its small kernel and minimal multi-server OS. We even avoided Nizza's legacy container and got a much smaller, robust and hopefully more secure system, since we believe that minimizing the trusted computing base is an essential requirement for trust into software.

1 Introduction

Anonymity while using the Internet is widely considered a legitimate and - for many use cases - essential requirement. A use case encompasses protection of privacy by avoiding traces and by preventing to reveal unnecessary private information. On the other hand, the Internet can be considered a least anonymous technology in wide use. Traces are left while visiting web sites on various levels, e.g. connection data on the network level and cookies as well as personal data on the application level. To enable network anonymity, anonymity services have been devised [8, 11]. They obscure the real source address of a user by routing lot of users over one or more proxies.

However, these anonymity services can be compromised as the following scenario, based on a real incident [15] in the authors' group, will show. An unknown criminal to be investigated was supposed to use an anonymity service. The police and later on the German Federal Bureau of Criminal Investigation (FBCI) required to investigate a criminal that uses a known URL. In order to enable criminal prosecution by a given warrant the software was extended to log connection data [6, 5]. This function of the anonymity software [3] was used only with a warrant but without directly notifying the users. After the warrant was reversed the FBCI showed up with a delivery warrant for a log record (which were later on reversed illegal [4]), first at the institute and later on at the institute's directors private home.

This incident shows a whole class of attacks on the anonymity and more general every computing service. Vendor and providers under constraint can reveal the service because of insider knowledge and physical access to those services. Besides reasons like corruption or human social engineering, also external influence such as extortion can cause attacks, which under normal situations would never happen.

Until now changes undermining the anonymity cannot be detected by users relying on the service. Within this paper we investigate how technical solutions can provide users of the anonymity service with additional information about the used software. On the one hand we use trusted computing technology to verify the anonymity software stack and on the other hand we target on strong decomposition and minimizing the trusted computing base in order to minimize attack points and make trusting the software itself justifiable. The work in this paper aims at the server side of the anonymity service and ex-

Part of this work has been sponsored by the European Research Council in the OpenTC and Robin project and by Intel.

cludes investigation at the client side. This would be a topic of research on its own.

The paper is structured as follows. In section 2 we describe the anonymity service and give a short introduction into trusted computing. In section 3 we look into related work, followed by the design and implementation of the extended anonymity service in section 4. In section 5 we evaluate the achieved protection against insider attacks and the size of the trusted computing base. The last section proposes future work and concludes.

2 Overview

2.1 Anonymizer service

We use the anonymizer service AN.ON [1] which is based on David Chaum's mix networks [8]. The general idea is to provide anonymous communication within a network by transferring and by recoding messages over several intermediate network nodes called mixes instead of directly transferring data from the source to the destination. By using mix network technologies senders, receivers or both sides are able to stay anonymous from each other. Mixes are message mediators similar to proxies, however mixes take care that received messages cannot be linked to messages they send forward. Basic methods to achieve this is to collect messages, to recode the message and to send the message in a different order than they were received.

AN.ON, an anonymizer service developed at the German universities of Regensburg and of TU Dresden, is used by thousands of users today. Figure 1 shows the general high level structure of the service, which consists of Java based clients called JAP, C++ based mixes, JAP based InfoServices and cache proxies. Single mix nodes are formed to a mix cascade and one mix serves one and only one cascade. InfoServices provide information about currently available cascades and provide meta information about the current number of users and data traffic workload of cascades. All information are visualized by JAP clients, where users can select a mix cascade. Web browsers of users are configured to use JAP as proxies, however JAP does not only forward messages like typical proxies do. JAP encrypts and decrypts all requests and responses according to the mix protocol, which will be described in detail below.

After a user has chosen a cascade, JAP connects to the first mix and establishes a channel. It uses a public key of the first mix to negotiate a symmetric key for encryption of data. The first mix then establishes a connection to the next mix and the JAP client also negotiates a symmetric key by using the public key of the second mix. This is done for all mixes until the last mix. After this setup phase a JAP client has symmetric keys negotiated with all mixes of the selected cascade. The user's web browser forwards requests locally to JAP which encrypts the data multiple times, first with the symmetric key of the last mix, then with the symmetric key of the mix before the last mix and so on. Finally JAP sends the encrypted packet to the first mix of the cascade. The first mix decrypts received packets and forwards them together with packets of other users to the next mix. Every mix of the cascade decrypts with the negotiated symmetric keys the received packets and forwards them to the next mix. The last mix obtains after decryption the original web browser request and sends it to the Internet. Relaying and encrypting replies of the Internet to users in the backward direction is done similarly but in the inverse order.

Currently the trust into a mix cascade can be based only on two things: a written statement of the mix providers, declaring the intent to do not log any data, and certificates of the public keys. The later ones are used to avoid man-in-the-middle attacks.

2.2 Trusted Computing

Trusted Computing [20, 12] is a technology that gives us two mechanisms that are useful for our scenario: Remote Attestation and Sealed Memory.

Remote Attestation allows a third party to verify which software stack (e.g. BIOS, Bootloader, OS, Applications) is running on a remote computer.

Sealed Memory allows to seal secrets to a particular software stack, thus preventing the disclosure of private data from another - potentially untrusted - software stack running later on the same machine.

Although other ways were discussed in the literature [13, 17, 7], trusted computing based on a Trusted Platform Module (TPM) is now widely accepted and TPMs are deployed in the millions. The TPM, as defined by the Trusted Computing Group [25], is a smartcard-like low-performance cryptographic coprocessor, that is soldered

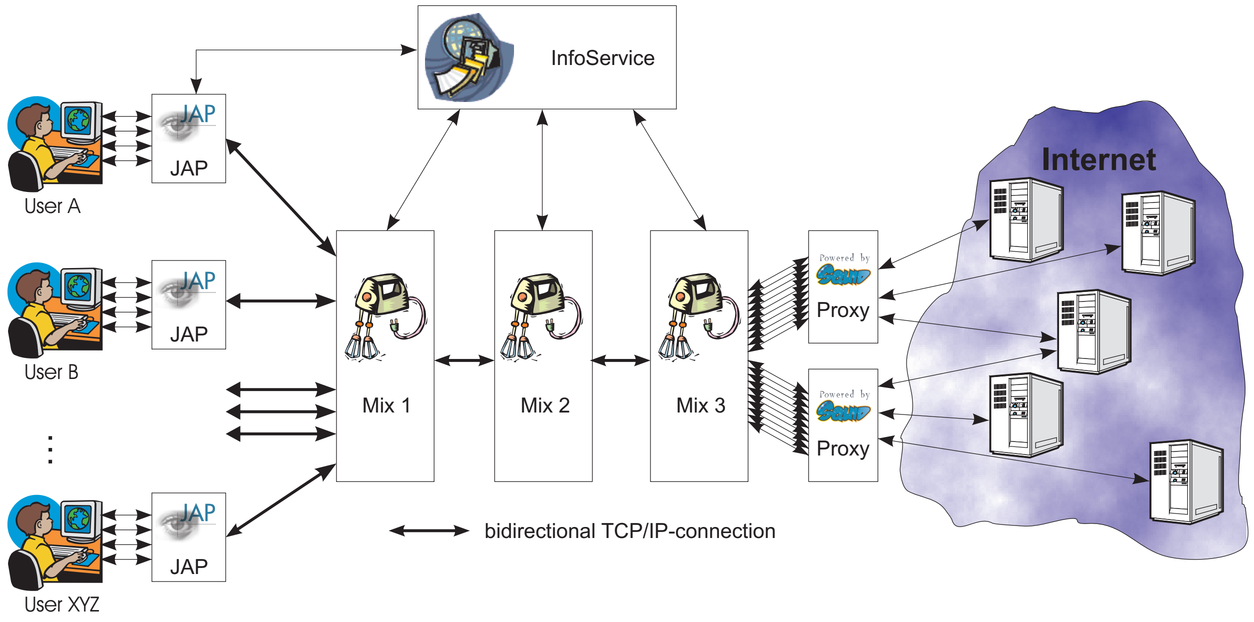

Figure 1: High level overview of AN.ON (figure source: AN.ON Projekt home page at http://anon.inf.tu-dresden.de)

on the motherboard.

In addition to cryptographic operations such as signing and hashing, a TPM can store a chain of hashes of the boot sequence. Using a challenge-response protocol, this chain of hashes can be signed by the TPM and verified by a remote entity. Similarly it can be used to seal data to a particular, not necessarily the currently running, software configuration. Unsealing the data is then only possible when this configuration was started.

A TPM based system has a couple of limitiations. The well known cryptographical limits (RSA assumption, SHA1 collision resistance) will not be targeted in this paper. Instead we have to keep in mind that the trusted computing specification explicitly excluded hardware attacks [16]. We will explain later in the evaluation section how this limit restricts the insider attacks we can tolerate.

3 Related work

Garriss et al. [10] are describing an Internet kiosk scenario based on the IBM's Integrity Measurement Architecture (IMA) [21]. IMA extends a normal Linux with trusted computing functionality.

Another Linux based solution is ProxOS [24] which splits the operating system on the system-call level. System-calls can be redirected to various Linux instances responsible for appropriate tasks. Within this solution multiple specialized and stripped down Linux kernels can be used to serve an application.

We decided not to use a monolithic operating system approach for our scenario because of different reasons. First, solutions based on a big monolithic system, for example on Linux, will not achieve such a small trusted computing base as a decomposed approach promises. Second, monolithic systems make it hard to protect and encapsulate different services from each other at runtime. Successful attacks on a single service running in the kernel can compromise the whole system. Third, our application scenario does not require a lot of functionality provided by monolithic kernels. One example is multi-user support, a feature we do not need. Without it attacks like local privilege escalation are impossible. Finally unneeded functionality of such kernels cannot be arbitrarily decreased with a reasonable effort.

We aim to use a reduced and small computing base to make evaluation of each component possible and trust in the whole computing base revisable. In order to achieve this we base the anonymizer scenario on Nizza [14]. Nizza is an architecture describing how small trusted computing bases can be achieved by using a L4 microkernel and few basic L4 services, using trusted wrappers and legacy containers. The general idea is to split applications into security critical and non security critical parts and to execute them isolated from each other. Typically non critical parts are executed in legacy OS containers and the security critical parts are executed directly beside the legacy OS container on top of an microkernel.

Work based on the Nizza architecture were presented in [23] and [22]. The former work splits applications based on components borders into security critical and non-critical parts. In contrast the latter one splits the application based on the information flow of the scenario. Both reuse large parts of the application within OS legacy containers.

In this scenario we want to go a step further by avoiding the legacy OS container at all. Instead smaller decomposed services will be used. This results in better isolation and a much smaller computing base that is easier to handle, robust in execution and finally more secure. Most required services like network stack, driver support and file system services are already supported to run directly on top of the L4 microkernel Fiasco which makes an implementation of the anonymizer scenario feasible.

Another mechanism to support better isolation and providing a small trusted computing base is used by McCune et al. [18]. Instead of using isolation provided by a (monolithic) kernel they leverage new hardware features introduced by AMD and by Intel to execute security sensitive code underneath the OS. Although this is a nice idea that fills a research gap, a couple of limitations remain. They currently suffer for example from the speed of the TPM and their approach is not multiprocessor friendly as they have to stop all other CPUs while executing in the secure environment.

4 Scenario design

4.1 The Trusted Computing Base

Figure 2 opposes the architecture of the Linux-based approch to the Nizza-based approach. The anonymizer mix is in both cases part of the trusted computing base (TCB) as it crypts user packets and routes them. The trusted computing support is currently part of the TCB as it sees the unsealed private key of the mix. Most of the basic L4 services are also included as either they have access to the memory the mix is using or they provide them with random numbers. Finally, the used L4 microkernel Fiasco itself is naturally also part of the trusted computing base.

4.2 Nizza components

The following central services are needed to run an AN.ON mix on Nizza: network Internet protocol stack, a file system service and drivers for network and TPM.

In order to access network devices we use a framework called device driver environment (DDE), which offers a Linux compatible environment to reuse unmodified drivers. Network drivers are wrapped by the framework into a library which are then used by a network multiplexer ORe to access the devices.

On top of the network multiplexer ORe we run a standalone network stack called FLIPS (Flexible Internet Protocol Stack). This stack is an extracted legacy Linux 2.4 Internet protocol stack which is encapsulated with the help of DDE into a single L4 process. FLIPS contains only the network stack, accesses ORe via inter process communication (IPC) and itself exposes an IPC interface to user applications.

The AN.ON mix requires minimal file handling functionality to load and find the xml configuration file during startup. The L4VFS framework provides Posix like interfaces and services. Posix function calls are forwared by the adapted lib-c for an application to appropriate L4VFS file-system services. The used L4VFS services are a file-system server providing the configuration of the anonymizer mix, a service providing stdin, stdout and stderr functionality and a service mounting the intital services in a file-system hierarchy. In our scenario we do not require drivers to access disks to load the configuration file. Instead it will be loaded during boot time into a

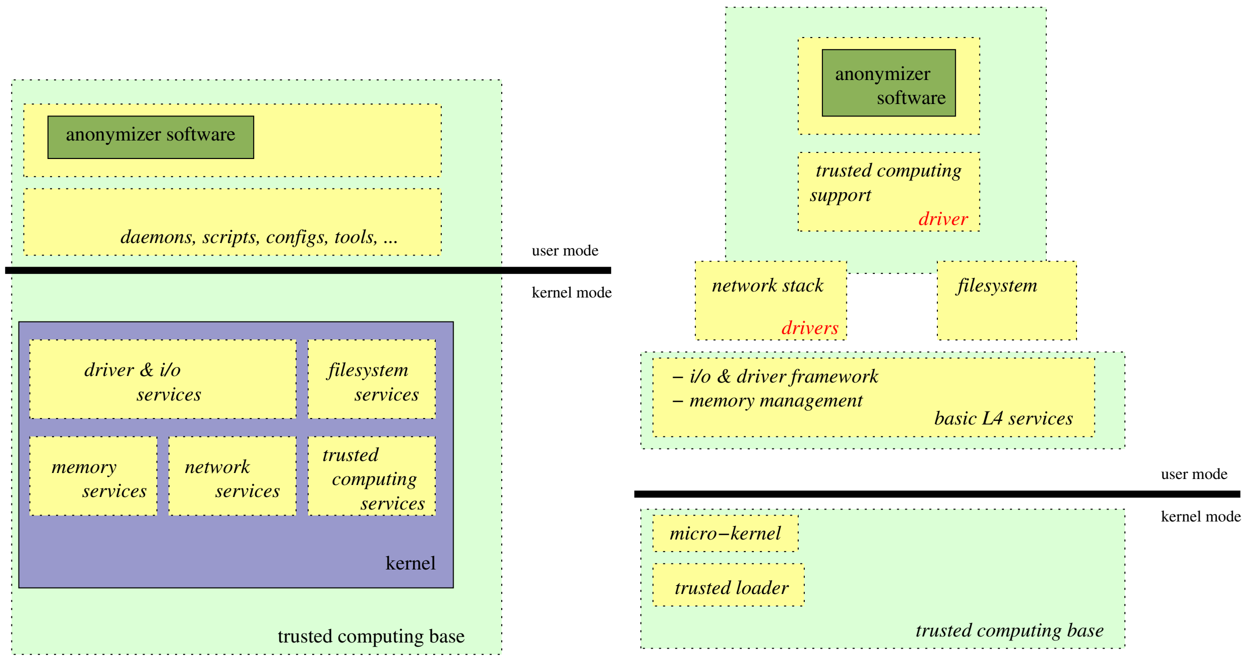

Figure 2: Linux based and Nizza based architecture

RAM disk.

The TPM service (STPM) provides an IPC based interface to user applications that can be used to seal/unseal data or to request quotes from the TPM. The TIS driver from the OSLO project [19] is reused to drive the TPM.

4.3 Minimize the Trusted-Boot Chain

The software chain that needs to be trusted in the traditional trusted computing scenario starts with the BIOS and the bootloaders, which should not be considered secure [16].

To remove them from the trust chain we can use the Open Secure LOader [19]. OSLO is a bootloader that relies on new hardware features, mainly the `skinit` instruction of newer AMD processors, to start a trust chain later in the boot process. This allows to remove the bootloaders and main parts of the BIOS from the trust chain.

4.4 AN.ON mix and protocol extensions

The private key of the mix, used for authentication to avoid man-in-the-middle attacks, is currently stored in the XML configuration file, which is obviously not a safe place. A more secure way would be to seal the key with the TPM. Another solution is to switch from a key external to the TPM to an internal one. This should avoid any propagation of the private key outside the TPM and it finally removes the trusted computing support out of the trusted computing base.

To be able to detect changes to the mix software we added remote attestation into the AN.ON protocol. This was achieved by embedding the TPM quotes as additional information besides similar XML meta-data in the AN.ON protocol.

The embedding of the quote is done in two places. First solely as hint in the data provided by the InfoService. These quotes can be used by JAP clients to determine whether they want to connect to a cascade at all.

Later a connected client can request its own quote from every mix. This operation will guarantee fresh quotes and avoids replay attacks.

If one of the mixes is rebooted after a symmetric key was negotiated, the specific mix will lose the key. Therefore a JAP client will detect this situation and can then renegotiate new symmetric keys and can request a new remote attestation.

4.5 Scenario implementation

The effort to port a AN.ON mix to the microkernel based operating system (TUDOS) turned out to be moderate since almost all parts of required functionality for the scenario were available. For example the uclibc provides required Posix bindings such as memory management, socket and filesystem handling. The mix requires the openssl and xerces xml library which were adapted to the building infrastructure by solving the expected minor issues like configuration, compiling and linking. Changes on the sources of the both libraries were not necessary.

In order to access the trusted computing service (STPM) a small library was added to the AN.ON mix which provides the IPC bindings to request TPM quotes. In our scenario we require an 1.2 TPM in order to leverage the `skinit` facility of newer AMD CPUs. We had to extend the STPM services by 1.2 TPM commands since some older 1.1 TPM commands were deprecated and the TPM declined the execution.

We extended the client side JAP of our anonymizer scenario to evaluate the remote attestation embedded as additional meta-information within the XML messages exchanged between JAP clients and mix servers. A verification routine was added in Java which evaluates the quote by checking the signature and compares it to a user specified expected hash. The GUI of the JAP client presents the result of the verification result besides the normal CA related information.

5 Evaluation and Discussion

Within this chapter we evaluate the effects of different insider attacks and measure the size of the trusted computing base.

5.1 Attacker profiles

We now describe some attacker profiles on our anonymizer scenario in which we assume that the attacker has some insider support. We describe how this can be detected and what can be done against such an attacker.

Trusted computing makes successful and unnoticed attacks harder, since remote attestation of the anonymity service AN.ON enables the users to verify which anonymity software and operating system is running as mix. Depending on this information they can decide on their own how trustworthy a mix is. Furthermore sealed memory is used to protect the private key of a mix during downtimes of the mix.

5.1.1 No physical access but software access

In the first scenario an attacker should have no physical access to the dedicated hardware of a mix, however will have access to the software. That means that an attacker can change the configuration or even boot other software on the mix. Possible imaginable cases are when an attacker compromised insider by extortion, due to social engineering or by corruption. Besides the fact that an attacker has to compromise all mixes of a cascade to be successful, changes on the software of the mix of our work would be detectable by users due to the remote attestation support. In order to avoid suspicion attackers, mix operators should generally provide some declaration why they modified software. Since the mix sources are completely publicly available, the changes can be reproduced by externals, institutions or experienced users. Unexperinced users could rely on a web of trust, similar to the PGP one, to rate the trustworthiness of changes and finally the whole software stack. As long as modifications are not publicly available and no public statement about altering the software is made a mix should be assumed as untrustworthy.

5.1.2 Physical access with limited knowledge about hardware

In the next scenario an attacker has physical access to the hardware on which the mix is running. Further an attacker should be able to exchange hardware, however he is not able, because of missing knowledge or physical protection, to modify or disassemble the security critical parts such as the TPM chip. In this scenario the attacker can exchange or add hardware components, for instance PCI cards or USB devices.

The major risk in this case are DMA attacks. A malicious device can read-out but also modify the memory the mix is using. Fortunately new chipsets can restrict DMA with their IOMMUs. Adding support for them into the OS would remove the threat.

Simple man-in-the-middle attacks, for example replacing the full machine, will not be successfull in this scenario, as the attacker can on one hand not clone the private key of the mix and on the other he will not be able to sign with the very same TPM key during remote attesation when using another TPM.

5.1.3 Physical access and hardware knowledge

If an attacker has full physical access and knowledge about attacking a TPM by exploiting implementation bugs or hardware limitations as described in [16], an attacker would be able to compromise the TPM in the end. Integrating the TPM into the chip-set and moving the TPM nearer to the CPU, the actual place where it is needed, will make successfull hardware attacks much harder. Even a good physical protection will help a lot against hardware attacks.

5.1.4 Exploiting software bugs with insider knowledge

In the next scenario an attacker has no physical access to the mix and no direct access to the software, however he is able to exploit bugs in the used operating services or the anonymizer software because of good insider knowledge of the sources. Depending on which service the attackers are able to compromise possible data about users of the anonymizer service can be leaked.

If attackers are able to exploit bugs in the kernel, the basic L4 services or in the anonymizer software, it is possible that they can enable logging or even get access to the session keys of the mix. Theses keys can be used to correlate the anonymous packets flowing in and out of the mix thus disclosing the anonymity.

If an attacker is able to exploit bugs in the network multiplexer ORe or the network IP stack FLIPS, an attacker would be able to trace source and destination of data packets that arrive and that are sent. As the mix is recoding and reordering the messages, these informations, that can also be collected by wiretapping on the network cable, should be worthless. An exception of this is the last mix of a cascade when users utilize unencrypted protocols like plain HTTP or POP3. Valuable private data like logins and passwords can be obtained at this point, as a recent case within the TOR network shows [26].

5.2 Managing many mix hashes

An open question is how users know, which values of a remote attestation answer are "good" mix services and which are not.

A possible solution is to embed expected hashes of a remote attestation of a mix in the certificate of the mix itself. However, here the responsibility of the certification authorities issuing such certificates would be to verify the hashes. This would include checking the operating system environment and the mix.

Another solution is to provide transparency about the procedure how the operating system and the AN.ON mix is built. If this procedure is reproducible by experienced institutions and experienced users then they could create identical binaries. Since the source code of the anonymity service and the used microkernel environment TUDOS is publicly available this should be possible. Booting the setup on own local machines and determining the final hash value of a remote attestation should match the one specified in the mix certificate or match the publicly known respectively. expected hash value.

Of course here a lot of issues will appear. First the exact build environment must be known, which comprises things like libraries, compiler, linker and so on. A simple difference in options, configuration or versions of the tools will result in different binaries. This is a big challenge for open source environments like Gentoo where everybody compiles on its own, perhaps slightly different, environment. Therefore some infrastructure is needed where the result of building the mix and of building the operating system is listed for example based on a fresh installed operating system distribution. Associated hashes of the binaries, hashes of resulting PCRs and the final remote attestation hash should be provided so that searching for differences is possible for institutions and for experienced users. If several independent people could report same results then such a mix could be considered well built and configured - and finally trustworthy.

5.3 Where Size Matters

We aimed for an minimal trusted computing base. To see how far we got we measured Source Lines of Code (SLOC) of the whole scenario. As SLOCs are only an indirect factor of complexity we also use a second metric: gzipped compressed binary sizes.

Service class	kilo SLOC	gzipped kB
Trusted computing bootstrap (Oslo)	2	5
L4 Fiasco Microkernel and bootstrap	30	85
Basic L4Env services	55	340
Network services	116	700
- tg3 driver	12	
- dde 2.6	36	
- FLIPS, IP stack	13	
- dde 2.4	40	
- others	15	
Filesystem services	25	215
Trusted computing service	2	45
AN.ON mix	460	1300
- xerces xml lib	210	
- openssl lib	200	
- AN.ON protocol and routing	50	

Table 1: SLOC of anonymity service scenario on top of TUDOS

The SLOC in table 1 sum up to nearly 700 thousand lines. To understand this surprisingly large number we have to look a little bit deeper into them.

The biggest parts that contribute to the overall size are the libraries used by the AN.ON mix. The Xerces XML library is a full-fledged validating XML parser used to generate and parse XML documents. As the mixes use only a subset of XML for configuration files and meta data, a simplified, non-validating XML parser and generator could be used. The Bastei [9] XML parser (300 SLOC) for example is at least two orders of magnitude smaller.

The same could be said for the OpenSSL library. Instead of using the standard library for cryptographic operations an embedded version can dramatically reduce the size. For instance MatrixSSL (<10 kSLOC) was used in a similar scenario [23].

There are two additional places to cut down the source: First, the separate IP stack FLIPS is still based on the Device Driver Environment (DDE) 2.4 whereas the network switch ORe uses the new DDE2.6. The 40 kSLOC for DDE2.4 could be omitted if we port FLIPS to the new DDE. Second, the 25 thousand lines for Posix-like filesystem IO are only used to load the configuration file. Using a simpler interface or even linking the configuration to the application would further reduce the complexity.

To sum this up we can conclude that our anonymity-service scenario can be built with less than 300 kSLOC, from which the first third is the basic system, the second third the network environment and the last third the mix application itself. The trusted computing part adds less than 2% (5 kSLOC) to such a scenario.

If we finally compare the current binary size of the scenario against a Linux version we can see that our implementation is much smaller. The complete AN.ON scenario requires 2.8 MB on x86 in gzipped binary form. This is an order of magnitude less than a Linux based solution, as the publicly available AN.ON Live CD [2] for instance requires 57MB. These numbers are not absolute in terms of smallest size, however indicate the range of the different solutions.

6 Conclusion and Outlook

Within this work we presented a server side trusted computing scenario. We achieved a better appreciable anonymous service for users by leveraging trusted computing technology. Based on the described extensions of the service, users are able to verify which software is running on the server. Now they have an additional mechanism to judge the trustworthiness of a mix beside the traditional ones like certificates and written statements of the operators.

This additional criterion together with the reduced trusted computing base of the mixes will make undiscovered attacks harder and should therefore increase the trustworthiness of the service.

Future work are to further reduce the trusted computing base as indicated in the evaluation section and to research how to structure and set up an infrastructure to handle "good" mix hashes and software updates. A real world deployment with a high number of users, should be able to detect performance bottlenecks for example on the TPM operations. Finally we would like to research how the client side of this scenario can benefit from trusted computing.

Acknowledgements

We would like to thank the reviewers for their suggestions to improve the paper. Special thanks go to Stefan Köpsell, a main developer of AN.ON, for information and discussions about internals of the AN.ON service.

References

[1] Project: AN.ON - Anonymity.Online. URL: http://anon.inf.tu-dresden.de.

[2] MixOnCD: AN.ON Live-CD. URL: http://anon.inf.tu-dresden.de/operators/help/mixsetupLiveCD_deployment_en.html.

[3] Anonymisier-Dienst JAP ist wieder anonym. URL: http://www.heise.de/newsticker/meldung/39813.

[4] BKA-Vorgehen gegen Anonymisierdienst JAP rechtswidrig. URL: http://www.heise.de/newsticker/meldung/41690.

[5] JAP, die Macher und die nicht mehr so ganz garantierte Anonymität. URL: http://www.heise.de/newsticker/meldung/39531.

[6] Nicht mehr ganz anonym: Anonymisier-Dienst JAP protokolliert Zugriffe. URL: http://www.heise.de/newsticker/meldung/39508.

[7] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *1997 IEEE Symposium on Security and Privacy*, pages 65–71, Oakland, CA, May 1997.

[8] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[9] N. Feske and C. Helmuth. Design of the Bastei OS architecture. Technical Report TUD-FI06-07-Dezember-2006, TU Dresden, 2006.

[10] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Towards trustworthy kiosk computing. In *Mobile Computing Systems and Applications, 2007. HotMobile 2007*, pages 41–45, Mar. 2007.

[11] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, 1999.

[12] D. Grawrock. *The Intel Safer Computing Initiative*. Intel Press, Jan. 2006.

[13] M. Gross. Vertrauenswürdiges Booten als Grundlage authentischer Basissysteme. In *Verläßliche Informationssysteme, Tagungsband, Informatikfachberichte 271*. Springer Verlag, 1991.

[14] H. Härtig, M. Hohmuth, N. Feske, C. Helmuth, A. Lackorzynski, F. Mehnert, and M. Peter. The nizza secure-system architecture. In *CollaborateCom*, 2005.

[15] C. G. Helmut Bäumler, Hannes Federrath. Report on the proceedings by criminal prosecution authorities against the project "an.on anonymity online". 2003.

[16] B. Kauer. OSLO: Improving the Security of Trusted Computing. In *16th USENIX Security Symposium*, pages 229–237.

[17] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[18] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. An Execution Infrastructure for TCB Minimization. Technical Report CMU-CyLab-07-018, Carnegie Mellon University, Dec. 2007.

[19] OSLO - Open Secure LOader. URL: http://os.inf.tu-dresden.de/~kauer/oslo.

[20] S. Pearson, editor. *Trusted Computing Platforms*. Prentice Hall International, Aug 2002.

[21] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[22] L. Singaravelu, B. Kauer, A. Boettcher, H. Härtig, C. Pu, G. Jung, and C. Weinhold. Enforcing configurable trust in client-side software stacks by splitting information flow. Technical Report GIT-CERCS-07-11, Georgia Institute of Technology, Atlanta, GA, May 2007.

[23] L. Singaravelu, C. Pu, H. Härtig, and C. Helmuth. Reducing tcb complexity for security-sensitive applications: three case studies. *SIGOPS Oper. Syst. Rev.*, 40(4):161–174, 2006.

[24] R. Ta-Min, L. Litty, and D. Lie. Splitting interfaces: making trust between applications and operating systems configurable. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 279–292, Berkeley, CA, USA, November 2006. USENIX Association.

[25] TCG: Trusted Computing Group. URL: https://www.trustedcomputinggroup.org.

[26] Embassy leaks highlight pitfalls of Tor. URL: http://www.securityfocus.com/news/11486.