FH JOANNEUM
University of Applied Sciences

Scanning the Internet for Security

Master Thesis
submitted in conformity with the requirements for the degree of
Master of Science in Engineering (MSc)
Master's degree programme **Advanced Security Engineering**

FH JOANNEUM (University of Applied Sciences), Kapfenberg

Supervisor: Wilhelm Zugaj, FH JOANNEUM Kapfenberg
submitted by: Gerhard Reithofer
personal identifier: 1310419037

November 2016

Abstract

The RSA method has been developed by relying on the fact, that the prime factorization of numbers has an exponential time complexity. In combination with none predictable random numbers as prime factors it is typically considered as unbreakable algorithm, because only the knowledge of the private key makes it possible to decrypt an RSA encrypted message.

D. J. Bernstein from University of Illinois has found a method to calculate some factors of coprimes in near linear time if factors are a used multiple times. In that context bad quality of random numbers which do not generate unique numbers will be a serious threat for the RSA encryption and all other algorithms which rely on prime factorization.

The University of Applied Sciences FH-Joanneum plans to develop a public security service which searches actively for these broken public keys in existing RSA certificates. It shall provide the possibility to end users to check their certificates against this weakness and to become actively informed if a broken certificate is found if desired. Therefore it will be necessary to scan regularly the Internet for available RSA keys and all found certificates will be investigated for broken public keys by finding the repeated factors. Finally the results are communicated to the users of the compromised certificates and users can inform themselves if a specific certificate is vulnerated.

This work describes the part for the security service which has been developed to collect as much as possible public RSA keys in short time which will be used for the factorization and how to keep the key database topically.

Graz, Nov. 14 2016

Academic adviser:
Wilhelm Zugaj

Gerhard Reithofer

Formal Declaration

I hereby declare that the present bachelor's thesis/diploma thesis/master's thesis was composed by myself and that the work contained herein is my own. I also confirm that I have only used the specified resources. All formulations and concepts taken verbatim or in substance from printed or unprinted material or from the Internet have been cited according to the rules of good scientific practice and indicated by footnotes or other exact references to the original source.

The present thesis has not been submitted to another university for the award of an academic degree in this form. This thesis has been submitted in printed and electronic form. I hereby confirm that the content of the digital version is the same as in the printed version.

I understand that the provision of incorrect information may have legal consequences.

Graz, Nov. 14 2016

Gerhard Reithofer

Acknowledgements

Thanks to my family which once more did support me in writing a scientific work while having a full time job as IT professional. This high workload was only possible to master because all members in my family endured my massively reduced time for others, they supported me in the practical screening and writing and showed consideration for this education.
In the scientific and educational aspect my academic advisor Wilhelm Zugaj was always helpful and supported all parts of this work independent if open issues had to be organized or was available for basic discussions if decisions were necessary to continue the work. Finally my colleague and friend Martin Wind always provided vital ideas to improve the whole project.

Contents

List of Figures

List of Tables

List of Abreviations

AWK	Aho, Weinberger, Kernighan
CBL	Composite Blocking List
CLI	Command Line Interface
CPU	Central Processing Unit
CRL	Certificate Revocation List
CSV	Comma Separated Values
DB	Data Base
DER	Distinguished Encoding Rules
DJB	Daniel Julius Bernstein
DNS	Domain Name Service
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
GB	Gigabyte
GBit	Gigabit
GMP	GNU Multiple Precision
GUI	Graphical User Interface
HTML	Hypertext Markup Language
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IO	Input Output
IP	Internet Protocol
IPS	Intrusion Prevention system
IPV4	Internet Protocol Version 4

IPV6	Internet Protocol Version 6
IT	Information Technology
JSON	Javascript Object Notation
NIC	Network Interface Card
OS	Operating System
PEM	Privacy Enhanced Mail
PKCS1	Public Key Cryptography Standards 1
PKI	Public Key Infrastructure
RAID	Redundant Array of Independent Disks
RAM	Randon Access Memory
RBL	Relay Black List
REST	Representational State Transfer
RNG	Random Number Generator
RSA	Rivest, Shamir and Adleman
SHA1	Secure Hash Algorithm 1
SNI	Server Name Indication
SOHO	Small Office Home Office
SPOF	Single Point Of Failure
SQL	Structured Query Language
SSD	Solid State Disk
SSL	Secure Socket Layer
TCL	Tool Command Language
TCP	Tranmission Control Protocol
TK	ToolKit
TLS	Transport Layer Security

Chapter 1

Introduction

RSA is nowadays the most used encryption system and has been a considerable issue in the academic world and well-known crypto scientists. In combination with none predictable random numbers as prime factors RSA can be seen as “unbreakable”, because only the knowledge of the “secret” (in case of RSA this is the private key) allows to decrypt an RSA encrypted message(Kerckhoffs, 1883). A practical implementation using the RSA crypto system is TLS(Dierks and Rescorla, 2008) in combination with PKI as key distribution mechanism where an identity is bound to a public key by a certificate which is signed by a trustworthy issuer.

This work is part of a planned security service at the FH-Joanneum which will actively scan for public certificates, analyze their content for specific vulnerabilities and inform the affected users about weaknesses on request and provide a possibility that users can check their certificate for the investigated implementations flaws actively.

The main target of this work can be expressed in a single question:

> How to collect as much as possible RSA based public keys and managing the data in a way to have a longterm consistent and repeatable dataset to support the planned security service in an efficient way?

A method to scan the complete IPV4 address range for public RSA keys efficiently will be developed. The main goal after this step is to have a most complete set of keys within a specific time frame. Methods to manage the data and keep it up to date will be acquired and the data life cycle will be considered.

Content overview

After this introductorily chapter the vision of the project will be explained and an overview including definitions of the single project’s parts and the delimitation of the content of this work from the various areas will be given. In the next chapter an overview about existing and related projects will be given. In chapter 4 the technical details based on defined requirements are described and the necessary environment, the interfaces and constraints will become defined. As result in one chapter the test setups, the developed and found scanning and management tools are tested and evaluated for doing the practical research in the following step. In the next chapter an answer how the main work for this thesis (the optimal collection of certificates) has been implemented will be given for the final implemented prototype.

In chapter 7 the practical part of the investigation will be described. Chapter 8 introduces the interface to the factoring module also including the found obstacles and back-draws. Statistical analysis of the data will work out further basic approaches for further research steps. Finally in chapter 9 a discussion shall give incitations for future developments and possible road-maps for similar projects. It will touch still open issues and the conclusion in chapter 10 will finish this work.

1.1 Research objective

> The American mathematician and popular science writer Martin Gardner published in August 1977 in the journal "Scientific American" a number with 129 digits which was a product of two prime numbers. The number was 114 381 625 757 888 867 669 235 779 976 146 612 010 218 296 721 242 362 562 561 842 935 706 935 245 733 897 830 597 123 563 958 705 058 989 075 147 599 290 026 879 543 541 and he left the exercise to the reader's community to find the two prime factors. The factors were found 16 years later in April 1994 whereas ca. 600 volunteers managed the calculation many days and nights distributed organized via the Internet using the software written by Arjen K. Lenstra from Bell Communications Research Center(Buchmann, 1994, p.80).

The factoring of this 432 bit number shows the basic principle of the RSA crypto algorithm, which relies on the fact that factoring of prime numbers for all known algorithms has an exponential time behavior. There are three required mechanisms which ensure the "secrecy" of the RSA crypto system.

1. The requirement that the factoring of prime numbers behaves exponentially.
2. The requirement that random numbers are not predictable (i. e. real random numbers).
3. The requirement that the methods using 1. and 2. are implemented correctly.

Whereas topic 3. can only be answered in the context of the specific implementation this work is concentrated on the first two.

The mathematician, cryptologist and programmer Daniel Julius Bernstein (DJB) from Eindhoven University of Technology and research at the University of Illinois at Chicago has found a method to factoring numbers into coprimes in essentially linear time(Bernstein, 2005). With a very large number of certificates and this algorithm supported by massive parallel computing power the RSA system could be attacked.

The research question is:

> Is it possible to get and manage a large number of RSA certificates and factoring the public keys of these certificates into primes in a time range which is smaller than the validity of a certificate?

If this question can be answered positive the RSA system can be seen as "broken".

Remark: This paper only concentrates on the collection of the certificates, the factoring part is answered in the master thesis "Copri - Factoring RSA keys" by Martin Wind at our University.

1.2 Problem statement

Security must not be seen as a static situation, it is a continuous process of setting up a secure environment and in further activities it is necessary to monitor and update all security relevant issues in near real-time. Monitoring, reacting on findings, optional adapting the parameters are typical to a “system”(Bertalanffy, 1968).

Cryptography is the basic method to ensure the privacy of data which is transferred over public Internet lines. Most cryptographically methods are based on a very few mathematical algorithms, in sense of broad acceptance and wide usage. The RSA method had been developed by relying on the fact, that the prime factorization of numbers has a time complexity for all known prime factorization algorithms which is exponential.

The random number aspect can be seen as secure because the number of primes in the 512-bit number space is ca. 10^{151} compared to estimated 10^{80} atoms in the universe. This high improbability guarantees that it “practically won’t appear” to get a random number more than once - as long as the distribution is equably.

The two main factors for cryptographic security in this context are the mathematically sound algorithm and a strong random generator which ensures that no factor will be used more than once. If this requirement is not fulfilled the factors can be recalculated and the algorithm is broken. The factorization time can be computed in case of linear complexity because the mathematical power is well known for actual computer hardware as this aspect is on the focus of security researches.

1.2.1 Short RSA Overview

The RSA cryptosystem developed by Ron **R**ivest, Adi **S**hamir, and Len **A**dleman, was first published in the journal “Scientific American” in August 1977.

Mathematical algorithm:

The RSA algorithm is very calculation intensive because it computes very large numbers. Therefore it is usually not used to encrypt/decrypt large amount of data (stream encryption/decryption). It is mostly used to enable a secure key exchange. In contrast to other asymmetric methods where one key is used for encryption and the other one for decryption, two components are necessary for encrypting (the puplic encryption exponent and the modulus) the messages in RSA. For decryption a different private exponent is used.

Encrypt message m with public key e:

$$E(e, m) = m^e \mod n = c$$

Decrypt message c with private key d:

$$D(d, c) = c^d \mod n = m$$

Mathematical background:

Three very large positive integers e, d and n which are all modular for exponentiation of the messages m must be chosen:

$$(m^e)^d \equiv m \mod n$$

Statement: It is extremely difficult to find d, even knowing e and n or even m.

The four steps of RSA

1. Generate a Key (p, q, n):

1. Choose two random prime numbers p and q:
2. Compute the modulus: $n = p \times q$
3. Compute Euler's totient φ : $\varphi(n) = \varphi(p) \times \varphi(q) = (p - 1) \times (q - 1) = n - (p + q - 1)$
4. Choose an integer e between 1 and $\varphi(n)$. e and $\varphi(n)$ must be coprime, i. e. $gcd(e, \varphi(n)) = 1$
5. Calculate $d = e^{-1} \mod \varphi(n)$, i. e. d is the modular multiplicative inverse of $e \mod \varphi(n)$

   Public key is defined as n, e, it is transferred to the encrypting partner.

   Private key is defined as n, d, it is never disclosed.

2. Distribute the Key n, e by transferring the public key (n, e) to the sender.

3. The sender encrypts the message m to cypher text c:

$$c \equiv m^e \mod n$$

4. The receiver decrypts the cypher text c to message m:

$$c^d \equiv (m^e)^d \equiv m \mod n$$

A comprehensible article about "The Mathematics of Public-Key Cryptography" was published in (Hellmann, 1979).

Statement: If one factor p for n is found, a simple division uncovers the second factor q and the encryption is broken.

Chapter 2

Project Overview

This chapter gives a general overall view about the project defined in section §2.1.

As the complete project would exceed the scale of this thesis, parts which are out of this work will only be roughly described herein. This document describes further details for the first step of the solution and the chosen name for the project was:

> **FJreSafe**
>
> which is a shortcut for **FH-J**oanneum **R**eal **E**nhanced **S**ecurity **A**pplication **F**or **E**ndusers

The different areas are worked out by different authors because the various single tasks are relatively heavyset. Probably additional project names will be used in addition, which define other project steps.

The project view described herein is focused on the part of the project for collecting data for the calculation step. Methods for managing it and some minimum interfaces to interact with the other project components will be described.

2.1 Security service

Under the mentioned constraints defined in section 1.1 it has been decided to develop an IT service which collects and manages the necessary data, organizes the methods to collect it regularly, keep it up to date and to calculate broken keys regularly. Furthermore the affected users shall be informed actively if broken keys have been found and a possibility to provide security checks for existing TLS certificates by the owner itself shall be implemented and managed in a secure way.

The security service has to cover three areas. The first two areas are time sensitive as they should keep the time span between the online activation of a certificate and the possible vulnerability detection small.

1. Collect as much as possible public RSA keys - this part is covered by this document in detail.
2. Calculate the broken RSA factors - this is covered by the Master-Thesis “Copri - Factoring RSA keys” of Martin Wind.
3. Provide the service interfaces and manage the related data for secure information interchange including user related data - this part will only be touched herein because the detailed working out would be exceed the goal of this work.

2.2 System overview

Figure 2.1 shows the main components of the project and the data relations in a functional view. The content of this work is only the area within the light gray background.

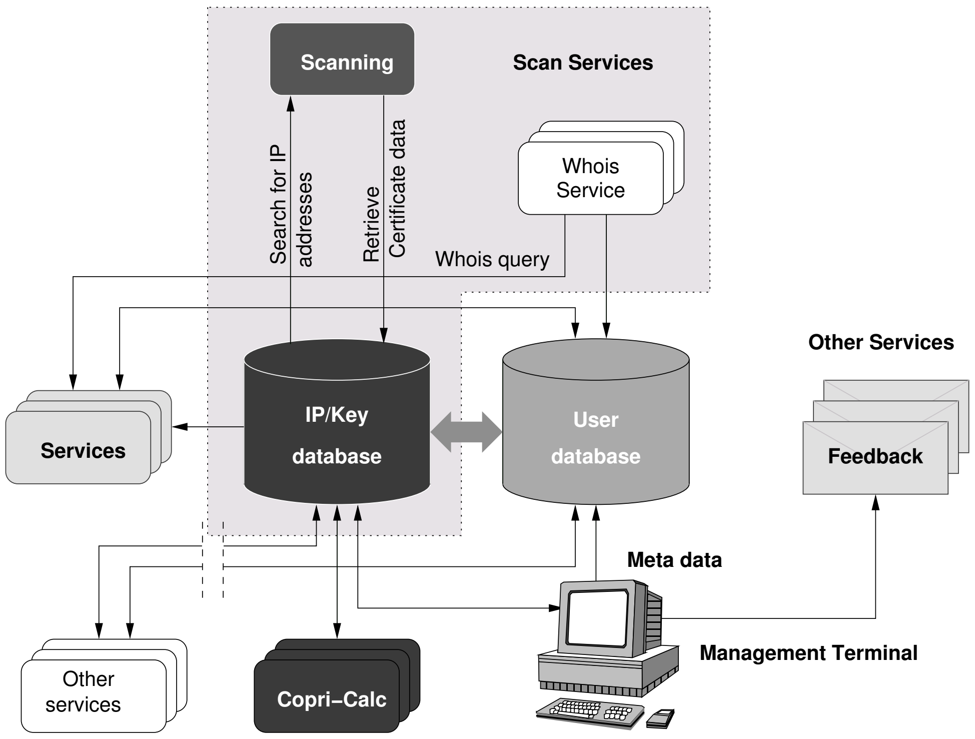


Figure 2.1: System Overview

The system shall finally contain at least the following components:

1. Scanning modules which collect the necessary information from the Internet. Several tools for managing the process fall into this category too.
2. A database where the technical data (certificates from collected data or parts of it) is stored and managed.
3. Another database for user related data. This is only for the planned service necessary and not directly related to the scanning actions.
4. The Copri-Calc interface to extract key data from the technical database for factoring the keys. The relevant results are the factors of broken keys.
5. A management console which connects technical data and user related data for contacting affected persons. This module will not be considered further in this work.

6. Various interfaces, like a WHOIS(Daigle, 2004) retrieval service, the linking of the calculation results with the technical database, web applications for providing user access from outside, etc.

As the necessary data has to be collected and managed the various data views must be created and stored in some way for later use. And finally the results must be presented to its users.

2.3 Internet data collection

The primary data for calculating vulnerable certificates keys are the public keys of TLS/SSL[1](Freier, Karlton, and Kocher, 2011) related services.

> The **N**etwork **S**ervice provides for the transparent transfer of data (i.e. NS-user-data) between NS users. It makes invisible to these NS users the way in which supporting communications resources are utilized to achieve this transfer(ISO8348, 2002).

This service definition in the ISO standard includes the term "*NS-user-addressing*" which is vital to this type of service. The service identification is defined by the following properties:

- As transport protocol the connection-based TCP is defined(Jon Postel, 1981).
- The IP address of the service identifies the server where the service is located.
- In addition the connection port separates different services for the same IP address.

Definition: A "service" connection is identified by a specific IP on the specific network port using the protocol TCP.

2.3.1 Scanning data management

Scanning is an iterative process. It begins with an initial load from the Internet and in further steps this data is updated with more actual information from scanning process (see figure 2.2).

The content covers different areas of data which is handled variously and the areas can be classified as follows:

1. The input is always the "complete" Internet with a defined list of services. This is a fixed amount of information and can be predetermined, the corresponding scanning step is called **Level I**.
2. From "all supposed services" of "all IP addresses" (the outcome from Level I) the TLS related part must be ascertained. This is done in step called **Level II** by trying to retrieve the presumed certificate data.
3. As result from the previous step the presumed data is retrieved and stored in the database.

The resulting data can be assorted into two categories i. e. the information in the first category is new "active" data which is valid and actively in use. It will be handled with a higher priority because it is relevant for the calculation step (refer to section §2.1) and it is important to get the results in short time to provide a responsible security service.

[1]As TLS (Transport Layer Security) 1.0 is the successor of the SSL (Secure Socket layer) version 3.0, sometimes referred to as SSL 3.1, the term TLS will be used exclusively for all related transport based protocols in this document.

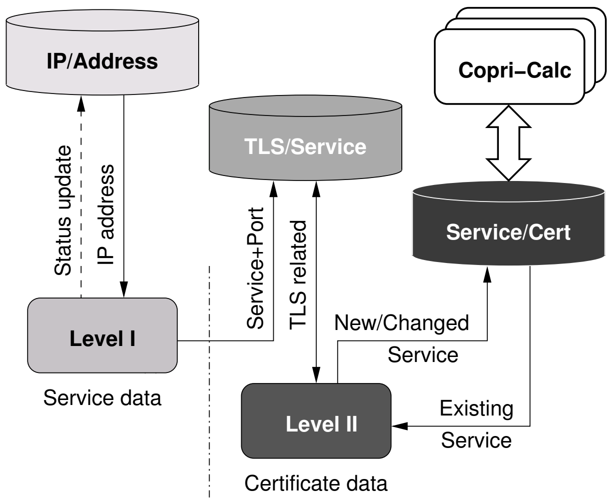


Figure 2.2: Data collection scheme

The other category covers “historical” data (e. g. expired certificates, deactivated services, etc.) and it is used only for getting a large number of RSA keys. This information will typically not be used for the security service to contact affected users, because it is from the security perspective outdated but it can be used to increase the coprime base for calculating broken keys[2].

The following prototypes of data management objects have been created:

1. User information web site - general contact point for users (refer to section §2.1).
2. Linear address table for full IPV4 range control (refer to section 4.2.1).
3. The SQL database for possibly TLS related IP addresses. In the same Database also the certificate related data is stored.
4. A Web-Query search form for information retrieval for RSA factors in HTML format or downloadable text based on single key input.
5. Another input method provides a possibility to upload lists of keys in JSON format which contains broken keys. The results can be downloaded as TAB separated CSV text file (refer to section 2.5.1).
6. Additional information can be retrieved if affected services are found by providing a link to a WHOIS(Daigle, 2004) query in the result records.
7. A certificate download possibility if affected services are found. The found certificates are stored as PEM(Linn, 1993) encoded files in the file system.

[2]Theoretically all someday collected keys can be used for factoring keys but even if the time increase of calculating the primes is linear, in practical life the amount of keys will be limited to a sensible size. This may differ depending on investigations, calculation strategies and other aspects.

2.4 Factoring into coprimes

D. J. Bernstein's algorithms are used to calculate the factors of a coprime base. This is described in (Bernstein, 2005) and the implementation from Martin Wind is available in his master thesis "Copri - Factoring RSA keys" at FH-Joanneum.

2.5 Service interfaces and data management

A service of that type has to provide possibilities to retrieve relevant data for users. A security service must do this in a trustworthy and secure way. Even if the final interfaces of service details are not defined, some basic implementations have been created which may be taken as basis for the final solutions.

The project has to cover the following areas:

1. Collect data by searching TLS related IP addresses (prototype implemented).
2. Retrieve the TLS certificate data (prototype implemented)
3. Store project specific data into a database (prototype implemented).
4. Calculate broken public keys (will be implemented by Martin Wind).
5. Provide a service where user can verify their keys (prototype implemented).
6. Inform users about found vulnerabilities (not implemented).
7. Share other service relevant information with registered customers (not implemented).

This work covers only topic 1. to 3. and gives some recommendation for topic 5. to 7. and topic 4. is covered by the work "Copri - Factoring RSA keys" of Martin Wind.

2.5.1 Copri user interface

The most important interface is the export of the public keys in the binary "MPZ-format" of the GMP Arithmetic Library(Granlund and GMP development team, 2014). The current implementation uses PostgreSQL(Development-Group, 2016) as database backend. The data structures are described in appendix B on page 126.

The interface is described in more detail in section 4.5.

2.6 User related data

One of the planned service's aim is to inform affected users about vulnerable keys. This implies that non-public personal data must be managed by the system, like physical addresses, email addresses, contact data, etc.

Technical related information is based on publicly available sources but the person related information (e. g, the users of the service) is not and must never be disclosed, except the owner of the data explicitly authorizes this step for some reason. If security holes may be become common currency before the user has the possibility to eliminate these defects it may lead to massive impairment and maybe financial loss.

Special cases WHOIS query and certificate data:

Some data is also available in public accessible WHOIS(Daigle, 2004) records and additional information about the owner of an IP address can be found in the TLS meta data.

In the current implementation any certificate is also stored as file in the file system in X509 PEM format(Linn, 1993). This was implemented also as fall back possibility in cases where the certificate reading failed and as source for additional X509 data retrieval if the database stored information is not sufficient.

2.6.1 User data vs. technical data security contexts

Person related data is confidential by itself and worth to be protected by law. The involved data is partially public and it is a basic requirement in cryptography that security is only depending on non-public secrets (private key). On the other side the users strictly claim the protection of their personal user related data.

To control this contradiction a very secure and clear separation of these two areas must be guaranteed. The author of this work strongly recommends to store these different classified information in separate databases and only an non-identifiable cross link information should be used to access the opposite world.

Statement: Independently from this information it is necessary for a trustworthy service that any non-public user data will be managed in a very secure way. The highest possible security constraints should be applied in this private context.

Chapter 3

Related Work

Extensive scanning the Internet is not a very special task. Several tools exist which are designed to scan IP addresses for specific reasons. The motivations may be benign or with malicious background. Typically port scans are interpreted as malicious attacks.

3.1 Scanning related works

The main reason for this work is security improvement. A similar project was "The Internet-Wide Scan Data Repository" (https://scans.io) which had also it's focus on security(Durumeric, Wustrow, and Halderman, 2013b).

> The Internet-Wide Scan Data Repository is a public multi-institutional archive of research data collected through active scans of the Internet that I am leading. The repository was founded as a collaboration between the University of Michigan and Rapid7 and currently hosts several terabytes of data including our regular scans of the HTTPS ecosystem, copies of the root HTTP pages, comprehensive reserve DNS lookups, and banner grabs from dozens of other protocols(Durumeric, 2014).

The project follows a similar approach, the aberrant details are that the "Internet-Wide Scan Data Repository" is focused on different payloads like FTP banners, the heart bleed vulnerability, etc. The "FJreSafe" Project limits its scan activities to RSA keys only and the scans are an important but "just one specific" project's part (refer to section §2.2 for details).

For the "interactive" use and also providing a data query interface a new project called "Censys" (https://censys.io/) has been launched in October 2015:

> Censys is a search engine that allows computer scientists to ask questions about the devices and networks that compose the Internet. Driven by Internet-wide scanning, Censys lets researchers find specific hosts and create aggregate reports on how devices, websites, and certificates are configured and deployed(Durumeric, Adrian, et al., 2015).

Very helpful was "HTTPS Ecosystem Scans"(Durumeric, Kasten, et al., 2013), a publication about the HTTPS certificate ecosystem.

A very informative source about the PKI system is the Ph.D. work of Ralph-Günter Holz(Holz, 2014).

3.2 Scan collections

The Censys Data Collection contains a large amount of scanning data (https://www.censys.io/data). The first collection of the data is classified as “Regularly Scheduled Scans” with 35 collections (see appendix C.1 on page 130). The second part is classified as “Historical Data” and is no longer run regularly. This part has not been factored in for this project as argued below.

In contrast with the simple download possibility of the ScansIO repository Censys provides various additional interfaces for registered users. Available are a REST based query language, direct SQL database access (for researchers), web search forms and predefined views as web sources.

3.3 Why re-scanning the Internet?

The ScansIO and Censys repositories contain partially “old” data for specific projects, which are not related to security only, like the effect of the Hurricane Sandy on the Internet 2012 or it was used for issues which rely on specific bugs, like Heartbleed(cve@mitre.org, 2013) which was disclosed 2013, etc.

The second but very important reason is that an Internet service, which tries to analyze a common security weakness or a secondary influence factor or common implementation flaws, must fulfill some dedicated demands.

3.3.1 Service locality, trustworthy and data owning

A service which is planned to support local customers and Internet users should be located where the user itself acts. In this environment it is more likely, that local security laws and security requirements will be applied. Furthermore it is important that the service users have faith in the data and even more in the use and the handling of the service related information. Also a near relationship between the service provider and the service owner is a stable basis for a good the cooperation.

3.3.2 Timing constraints and data actuality

Security issues have a time dependent impact on the user and organization. After becoming aware of a vulnerability the reaction time on it should be as less as possible.

In the case of an RSA weakness because of bad RNGs used for creating certificates, it is even possible to find broken keys before attackers may use them. The probability to find a weak key by active investigations before it can be used by others will probably be higher than to find a programming error in the TLS implementation by chance. Nevertheless the obligation to react as soon as possible when a security issue is found commits a necessary reaction with as less as possible delay.

This leads to a contradictory requirement:

1. Keep the data which is used to analyze as much as possible actual.
2. Beyond the negative constraint that a large amount of data is necessary to get relevant results, this increases the time to calculate broken keys.

Finally there will be an optimal compromise to get the results in short time and to have a reliable amount of data and computing power for the calculations.

Chapter 4

Project Requirements

This chapter describes technical details to define the environment in a way that all steps can be repeated if the required environment is present. This information can also be used to port it to other environments if parts are changed, like another software component, platform, programming language, etc.

Quality and quantity aspects are discussed to provide basic data for changing the architecture for scaling considerations to lower needs, e. g. to reduce costs by using cheaper hardware or limited data perimeter, like less service ports. On the other way this sketch should also be helpful if someone wants to increase the power for shorter scanning times like upgrading the Internet connection to higher throughput or creating a more sophisticated parallel collection environment.

4.1 Basic assumptions

To get a rough picture about the time line of data processing an estimation had to be calculated to get a quantitative idea which is necessary to define the perimeter of the possible services. The target is to check the complete IPV4 address range. The limit is in this case the theoretical amount to $2^{32} = 4294967296$ possible IP addresses.

This leads to the following model calculation assuming that for scanning an address and the storing the certificate one second is needed. The complete scanning time would be:

$$2^{32} = 4294967296\,seconds = \frac{4294967296}{3600 \times 24} \approxeq 49710.2\,days \approxeq 136.1\,years$$

This is out of any acceptable range, therefore another approach was taken to stipulate a maximum runtime and calculate the necessary number of scans per second. The target was to finish a full scan in two weeks:

$$\frac{4294967296}{3600 \times 24 \times 14} \approxeq 3550\frac{IP}{sec}$$

Remark: This was taken as ambitious target.

4.1.1 Preliminary investigation

Another open issue was to determine the amount of data which has to be transferred via the network line to get the network rate. The universal command line tool “openssl” (http://www.openssl.org/) was used to contact randomly selected IP addresses. By default it requests a TLS handshake and delivers the amount of transferred data. The address list was taken from the Wide Scan Data Repository (refer to section §3.2) with 10.000 randomly selected addresses. The address list has been stored in the file “20150209_hosts_uniq” with one IP address per line.

```
$ cat 20150209_hosts_uniq|while read IP
do echo "quit"|openssl s_client -connect $IP:443 1>xfer_${IP}_443.out 2>xfer_${IP}_443.err
done
```

The result of this command is a list of files named like “xfer_108.80.37.171_443.out” and “xfer_-108.80.37.171_443.err” containing the standard output and standard error output from the openssl call. A small AWK[1] script (refer to appendix G.1.19 on page 152) was created to calculate the average amount of data from all successful TLS handshakes.

```
$ ./sta/stats_xfer.sh
7622 handshakes, bytes read: 19363.3KB, wrote: 3962.29KB
this is avg. of 2601.42 bytes read and 532.326 bytes wrote per conn
smallest rsize: 538, largest rsize: 31441
smallest wsize: 186, largest wsize: 912
```

This test shows that an approximate size of read data per TLS handshake is 2601.4 bytes and 532.3 bytes are written per successful connection. Using these values the necessary amount of data and the data rate can be calculated:

$$2.6\frac{Kb}{IP} \times 3550\frac{IP}{sec} \times \frac{7622}{10000} \approxeq 7035\frac{Kb}{sec} \approxeq 7.0\frac{Mb}{sec}$$

Taking into account that all addresses are TLS related a minimum throughput of 7.0 Mb/sec for transferring the certificate data only would be necessary.

Definition: The minimum network throughput of a **100 MBit** line has to be used for the certificate retrieval step.

4.2 Scanning steps

The main goal is to find TLS certificates (refer to chapter 2). Retrieving certificate data requires a non-negligible amount of data transfer (ca. 2.6 Kb - see section 4.1.1) whereas a TCP-SYN-Scan only creates a payload of ca. 40 bytes. This fact makes it sensible to separate these two data collection steps. One overall port scan, which just searches for possible TLS relevant IP addresses and a second step which checks for TLS data and retrieves the relevant information. The two steps behave very unequal which leads to different processing:

Level I scanning This process will be highly parallelized because mainly the connection management will probably limit this step in the minimum setup. The amount of data can be almost neglected in the planned speed range but when the scanning speed becomes increased massively also the network rate may limit the process (refer to section 7.2.5).

[1]“awk” is a widely used Unix tool for field based pattern processing programmed by Alfred V. **A**ho, Peter J. **W**einberger and Brian W. **K**ernighan.

Level II scanning The TLS data collection will produce a considerable amount of network transfer and this data must be stored in a structured manner for later use[2].

4.2.1 Data preparation

The plan is to scan the complete IPV4 address range. The definition "complete" contains the predefinition that data once read will not be read again within one scanning cycle.

Port scanning software often provides the possibility to "randomize" the access of a defined IP range. But this mode cannot guarantee that a "complete" range will be scanned if dividing the scan into several steps, like it will be necessary for parallel scanning tasks or if scanning parts must be repeated.

To cope this problem some sort of "database" has to be installed which documents and controls the scan progress. The minimum amount of disk space for storing an IP address is the four bytes. One additional flag-byte have been reserved as management space (256 possible flag values or eight flag bits) to have the possibility to store additional information to each IP address. Written into one single file this gives as file size of:

$$(4 + 1) \times 2^{32} = 21474836480 \approx 20GB$$

With this file a specific ordered list of unique IP addresses will be prepared and with a file offset pointer each address can be accessed repeatable and the corresponding flag can be read or written.

Remark: This "Randomized Linear IP-table" ensures that multiple scans can be executed with equal edge conditions and in exactly the same order.

4.3 TLS related data

Data related to certificate information is bound to a specific IP address and a connection port. Two key records are used to identify the source of a specific public key, which will become calculated by the "Copri-Service", i. e. the key factoring step (refer to section §2.3).

The result is a list of broken keys and the contained common factors. With this information a database lookup can be made to find the specific service's information.

Preclusion: Special setups which use the TLS extension SNI(Eastlake, 2011) are not taken into account. The main reason is that this extension requires a relatively complex handshake because the certificates are bound to a FQDN instead of a server address. These host names cannot be handled in a way like a clear defined gap less list of 4-byte integer numbers in case of IPV4 addresses which can be retrieved by a mathematical algorithm and which have also mathematically defined boundaries.

From a known server name it is necessary to do a DNS lookup for getting the IP address, using this address the connection can be made, i. e. the standard procedure for address bound services. To get a server specific certificate (virtual hosting) the requested server name must be transferred to the server (*server_name*-Parameter) to get the corresponding certificate.

Furthermore also the upcoming IPV6 addresses are not supported by the project. The reasons are that the IPV6 address range is far beyond of a full scan range (it uses 128 bit addressing) and the majority of publicly available servers still use IPV4. Another argument against SNI is that no specification for IPV6 exists. Therefore IPV6 is often proposed as future-proof solution in multi-server environments.

[2]Writing to database was another limiting resource in the test environment which will be explained later.

4.4 Technical environment

Without knowing the detailed technical needs for this step only the criteria before were used to search a provider. The parallel processing aspects, like distributed processing via different servers and providers have not been considered in detail and the "single root server" offers from the providers did not include enough information to factor in these specific aspects into the evaluation.

Some basic rules have been introduced into the acquisition requests for a server:

- Many CPU cores are vital in this case, because of the massive parallel processing.
- Large memory (32 GB RAM minimum) makes it easier to calculate with large amount of data, like four billion IPV4 addresses (2^{32}).
- The 100 MBit backbone guarantees a short latency time in the multi endpoint access (much routing) a one GBit NIC was recommended.
- Large and fast disk space is required. SSDs allow fast read/write requests especially for the database access. But the costs are significantly higher than for magnetic disks of comparable size. The tablespace for the database used 120 GB and the overall project data use was ca. 600 GB for both project phases. The reasons was mainly due to much log data for analyzing.

Aspects which have been underestimated:

- The single IP address became obvious to a SPOF because of easy detection and blocking by IDSs and IPSs. This is discussed in section §8.4 in more detail.
- While scanning and managing the service data on one machine with a single disk[3] each read/write intensive process hindered other massive IO related actions, this is also a topic in section §9.2.

More details to the final used hardware can be found in appendix D on page 134.

4.4.1 Disk storage

As already mentioned in section §4.2 two different storage classes have been defined for the scanning part.

1. The **Level I data** consists of the complete IPV4 address space. For each address entry (4 bytes) one single byte of additional information has to be reserved which results in a disk space 20 GB. For more details regarding the "Full Scan Linear Table" see section 4.2.1.
2. The more structured **Level II data** is best stored in a database. Also the conception for the user related data is to become stored in a database, see also section 4.4.3.1 and 4.4.3.2.

4.4.2 Collector agents

The application for the TLS key collection and possible WHOIS(Daigle, 2004) data is described in section §6.2. For details about the data management for collecting TLS keys, IP range definition, status and different collection types (update, search, full scan, . . .) refer to section §6.5.1.

[3]Even the fact, that two physical disks were installed the same limitations appeared because the disks were organized as RAID I.

4.4.3 Data storage

The managed data encompasses two different security contexts.

4.4.3.1 Technical Database

Non-person related technical data, IP/Key (low security level) is publicly available and necessary data for processing the key data. It does not use any kind of non-public user related information by intention.

Technical data (IP/Key) The technical database stores different type of information:

- IP related information (Level I data)
- SSL data related information (Level II data)
- Management data (anonymous index pointers to user related data)

4.4.3.2 User related Database

The user related Database contains all none disclosed contact data, management data (very high security context) and additional management information for fulfilling the key service needs.

User related data (meta data) The user database stores different type of information which is partly related to persons:

- Person related data
- Management data
- Logical connection to Technical data

Remark: As the user related data part was not implemented for this work, it will not be covered further in this work.

4.4.4 Database system

The chosen database system was PostgreSQL. PostgreSQL is a powerful, Open Source object-relational database system which was also used by (Durumeric, Wustrow, and Halderman, 2013b).
The chosen database name was “keyservice” the user name was “keymaster”[4].

```
postgres=# create user keymaster with SUPERUSER LOGIN password 'secret';
CREATE ROLE
postgres=# create database keyservice with owner=keymaster;
CREATE DATABASE
```

[4]Due to the length of the project (> year) more databases were used for the different phases. Relevant details will be described in the corresponding document sections.

4.4.4.1 Technical database setup

The technical data information (IP addresses, service and certificate related information) is stored in three tables. A small SQL script was written to install and initialize the structures (refer to appendix B.1.4 on page 128 for the listing).

```
keyservice=# \i sql/keyservice_inst.sql
CREATE TABLE
ALTER TABLE
 ...
INSERT 0 1
INSERT 0 1
```

More detailed information regarding the database structures can be found in appendix B.

4.5 Copri interface

The implementation is based on an existing TCL program which has been developed in an earlier project phase in 2014. This project had the stated objective to validate the algorithm of Daniel Bernstein (refer also to section 1.1 for further information). The file read/write functionality for the MPZ format used by the GMP library(Granlund and GMP development team, 2014) was already implemented and the database specific parts have been added.

More detailed information about the usage of this tool is given in section §8.1 and in appendix A.1.2 on page 107.

The result from the Copri-Calculation is for each broken key:

1. The broken public key “**n**”, the product of two prime factors.
2. One factor “**p**” and
3. the other factor “**q**” of the product.

To identify the affected service a reverse lookup (search the public key in the “rsa_certs” database table) must be done. A simple user web-interface has been programmed which allows the input of a single key value or optionally a list of broken keys in JSON format can be and uploaded and the found service entries will be returned. Additionally links are contained in the results which provide the complete certificate in PEM format(Linn, 1993) and real time WHOIS(Daigle, 2004) information.

The usage if this interface is described in more detail in section §8.2.

Chapter 5

Current state and preliminary investigations

This chapter describes the current technology level to get the necessary information needed for this work and the tests and the decisions which have been made for the different technologies.

Several tools have been tested to evaluate the final option for the four steps of the scan process.

1. Scan Level I - port scanning for discovering relevant IP addresses. Get all possible addresses which may serve TLS services and store the necessary service information for achieving the following step.
2. Scan Level II - check for a TLS service and try a complete handshake. Get all necessary certificate data and verify it. Store new found information in the database, modify changed and delete invalid or unused data.
3. Certificate Management - necessary certificates data becomes stored in the filesytem. The data can also be used for statistical reasons and as input for re-extracting RSA data.
4. User Service Interface - the resulting RSA key calculation will be presented to users to allow querying for broken keys. A limited prototype has been developed.

Whereas for step 1. several tools exist, no satisfying solution have been found for second one. Thus a new solution has to be developed to retrieve certificate data.

At the same time the tools for writing the data into the database (step 3.) were developed, based on the same scripting technology which was used to glue together the different parts into the main management environment. In addition several Shell scripts have been written as utilities to wrap some program calls or simple tools to extract statistic data (e. g. AWK command scripts for filtering log files) or just to make it simpler to call several programs within one command.

Finally a small User Service Interface has been created as web service.

5.1 Scanning - Level I

Port scanning can be seen a “common” IT task and is used in many areas. This may be analyzing the own network for monitoring reasons or hardware checks and it is used also massive for illegal tasks to search for vulnerable Internet services and servers, etc.

Remark: This is also the main reason, that port scanning is often interpreted as malicious action.

Many tools are in use for security reasons to protect the Internet infrastructure of persons and companies. Typical exponents are Firewalls, Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs).

The most common used tool to check the existence of an IP address and a valid connection is the command **“ping”**, which uses the ICMP protocol(J. Postel, 1981) for verifying a correct remote node connection. The protocol is used to transfer failure error information across the network. As this mechanism is typically for internal use only and it’s not necessary for the service itself many firewalls block this protocol and repeated ICMP access is often interpreted as network attacking misuse. Therefore this mechanism was not used for port scanning.

The final requirement necessitates to store the found data as it is used for the following scanning level.

5.1.1 Scanner Nmap

The tool **“nmap”** is a very powerful network/port standard scanning tool. It’s main usage is analyzing available services in networks. It supports range-scanning and various types of scan methods to be more thorough or to “hide” the scanning access. This Open Source tool is available from https://nmap.org/ and is included in most Linux distributions.

The fact that nmap’s strength is the large functionality and not the scanning speed forced the decision to search for a more speed oriented tool.

Several scanners have been tested regarding different aspects whereas the necessity for automation was indispensable.

5.1.2 Scanner ZMap

As described in section §3.1 there is already a similar project available in the Internet, the “Internet-Wide Scan Data Repository”. The same project group developed a high speed Open Source scanner called **“zmap”** (https://zmap.io/), refer to appendix A.2.1 on page 109 for the program usage.

The program was downloaded and compiled on the current 64-bit Linux Debian 7.

```
git://github.com/zmap/zmap.git
sudo apt-get install build-essential cmake libgmp3-dev libpcap-dev gengetopt
    byacc flex

cmake -DENABLE_DEVELOPMENT=OFF -DWITH_JSON=OFF
make
```

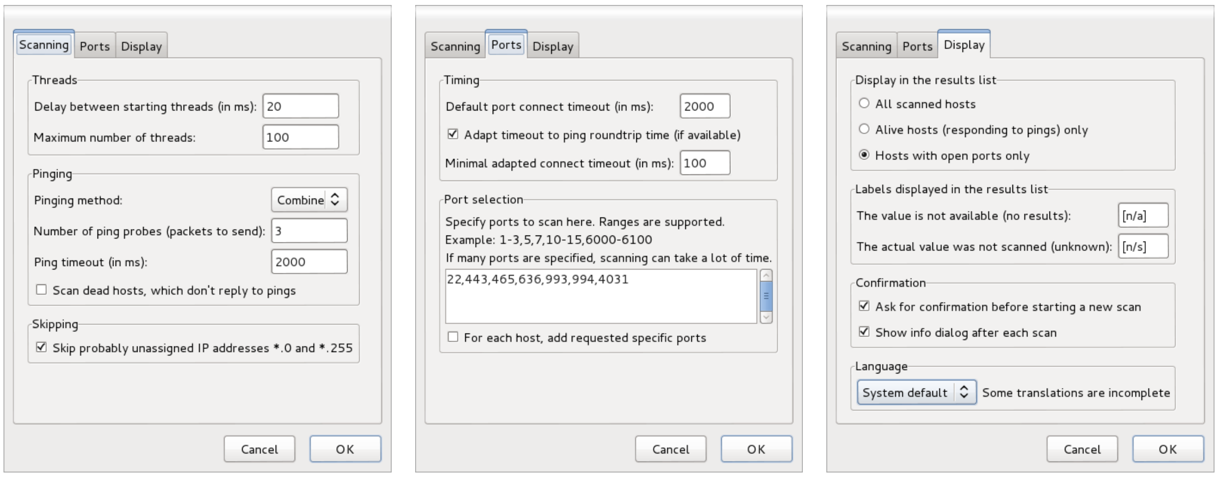
According the paper “ZMap: Fast Internet-Wide Scanning and its Security Applications”(Durumeric, Wustrow, and Halderman, 2013a) the tool is able to scan the complete IPV4 range in very short time. Unfortunately the first tries provoked complains about presumed security attacks, refer to appendix F.1 on page 143.

Either the scanner was "too fast" or it did not distribute addresses in a way that fortuity could be assumed. Another aspect against this scanner was that it did not support the scanning method in the way as it was intended. The plan was to use predefined address lists from files as this "packet" handling supports scaling, repeatability and controlling the completeness of the requested information.

5.1.3 Angry IP Scanner

> Java based multi-threaded Open Source scanner (http://sourceforge.net/projects/ipscan/) with good performance and scalability.

The GUI of the scanning tool **"ipscan"** allows to specify the scan parameters in graphical dialogs and the corresponding settings can be saved in files for later (re)use. This makes it possible to work with predefined address lists and ports, which was the presumed working method. Refer to figure 5.1 for the settings dialog examples.



Scanning

Threads
Delay between starting threads (in ms): 20
Maximum number of threads: 100

Pinging
Pinging method: Combine
Number of ping probes (packets to send): 3
Ping timeout (in ms): 2000
☐ Scan dead hosts, which don't reply to pings

Skipping
☑ Skip probably unassigned IP addresses *.0 and *.255

Ports

Timing
Default port connect timeout (in ms): 2000
☑ Adapt timeout to ping roundtrip time (if available)
Minimal adapted connect timeout (in ms): 100

Port selection
Specify ports to scan here. Ranges are supported.
Example: 1-3,5,7,10-15,6000-6100
If many ports are specified, scanning can take a lot of time.
22,443,465,636,993,994,4031
☐ For each host, add requested specific ports

Display

Display in the results list
○ All scanned hosts
○ Alive hosts (responding to pings) only
● Hosts with open ports only

Labels displayed in the results list
The value is not available (no results): [n/a]
The actual value was not scanned (unknown): [n/s]

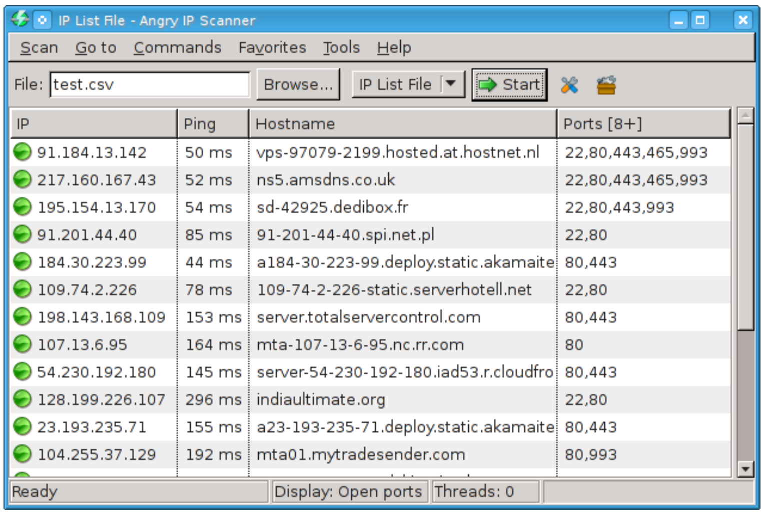
Confirmation
☑ Ask for confirmation before starting a new scan
☑ Show info dialog after each scan

Language
System default Some translations are incomplete

Figure 5.1: Examples for used settings

The main reason for trying this scanner was the simplicity as it has only a very few but important amount of settings and good configurability. Other favorable features were the promised scalability with a configurable number of parallel working threads and that it is possible to provide the respective IP addresses in a file. This supports the consideration for "intensive" or "sparing" scanning and other basic requirements for the planned service.

Refer to appendix A.2.2 on page 110 for the command line syntax and in figure 5.2 a typical result output is shown.


IP List File - Angry IP Scanner
Scan Go to Commands Favorites Tools Help
File: test.csv
Browse...
IP List File
Start

IP	Ping	Hostname	Ports [8+]
91.184.13.142	50 ms	vps-97079-2199.hosted.at.hostnet.nl	22,80,443,465,993
217.160.167.43	52 ms	ns5.amsdns.co.uk	22,80,443,465,993
195.154.13.170	54 ms	sd-42925.dedibox.fr	22,80,443,993
91.201.44.40	85 ms	91-201-44-40.spi.net.pl	22,80
184.30.223.99	44 ms	a184-30-223-99.deploy.static.akamaite	80,443
109.74.2.226	78 ms	109-74-2-226-static.serverhotell.net	22,80
198.143.168.109	153 ms	server.totalservercontrol.com	80,443
107.13.6.95	164 ms	mta-107-13-6-95.nc.rr.com	80
54.230.192.180	145 ms	server-54-230-192-180.iad53.r.cloudfro	80,443
128.199.226.107	296 ms	indiaultimate.org	22,80
23.193.235.71	155 ms	a23-193-235-71.deploy.static.akamaite	80,443
104.255.37.129	192 ms	mta01.mytradesender.com	80,993

Ready | Display: Open ports | Threads: 0

Figure 5.2: Angry IP Scanner - GUI

5.1.3.1 IPScan test series

A series of tests have been made, varying several parameters and using different hardware. A typical examples is:

- Maximum number of threads: **256**
- Pinging Method: **TCP port probe**
- Skip probably unassigned IP addresses *.0 and *.255: **activated**
- Port selection: **22,80,443,465,636,993,994,4031**
- Display in the results list: **Hosts with open ports only**

A scanning call example using **“ipscan”** as plugin module and the output from the scanner management tool **“scan_sequence.tcl”** is shown here.

```
$ tclsh 0.5/scan_sequence.tcl
New scanning table offset is 51911, 2048 records written to job table '51911_2048'
Saved results to /net/pub/ASE/2015SS/MasterThesis/Software/keyservice/dat/51911_2048.csv
2048 IPs scanned in 00:00:66 (31.0/sec), 2038 SSL related IPs (99.51%) found (30.88/sec)
```

Refer to appendix A.6 on page 114 for more information about the scanner plugin module functionality.

5.1.3.2 Parallel scan test

The first test was made by calling four scan jobs at the same time on a Quadcore CPU with 256 threads. Refer to appendix C.3.1 on page 131 for a typical program output. Test results with different IP address ranges using the settings mentioned in the example before are shown in table 5.1.

job	start	end	blksize	secs	ips/sec	found	found/sec	%	traffic
104135	10:35:24	10:36:14	2048	50	41.0	33	0.66	1.61	3572964
106183	10:35:25	10:36:18	2048	53	38.6	32	0.60	1.56	3730304
108231	10:35:26	10:36:24	2048	58	35.3	32	0.55	1.56	3785842
110279	10:35:27	10:36:22	2048	53	38.6	29	0.55	1.42	3547280
summary	10:35:24	10:36:24	8192	60	136.5	126	2.10	1.66	238 kB/sec

Table 5.1: Angry IP Scanner - quad core, 100 threads

5.1.3.3 Scalability tests

The next approach was to vary the number of threads and the block size. The tests have been made on three different computers, the quad core and two dual core machines, see table 5.2.

addresses	secs	cores	threads	ips/sec	found	found/sec	%	remarks
8192	60	4	256	136.5	129	2.15	1.57	port 80 only
8192	63	2	1024	130.0	137	2.17	1.69	port 80 only
8192	72	2	256	113.7	151	2.10	1.86	port 80 only
1024	60	4	100	17.06	24	0.40	2.34	12 ports
1024	30	2	512	33.86	9	0.30	0.90	5 ports
1024	20.4	2	1024	50.20	20	0.98	1.95	5 ports

Table 5.2: Angry IP Scanner - timing values

These tests already show that the results are far away from the targeted goal. The application is GUI based and requires a graphical frontend (no text-only CLI) which makes it harder to automate and which is most probably also an important fact for the bad result.

5.1.4 Massive Scanner: masscan

Excerpt from README.md of the tool **"masscan"** on the developer's page on https://github.com/-robertdavidgraham/masscan (visited on 2016-04-18).

> This is the fastest Internet port scanner. It can scan the entire Internet in under six minutes, transmitting ten million packets per second.It produces results similar to nmap, the most famous port scanner. Internally, it operates more like scanrand, unicornscan, and ZMap, using asynchronous transmission. The major difference is that it's faster than these other scanners. In addition, it's more flexible, allowing arbitrary address ranges and port ranges.

At evaluation time no powerful hardware was available which is necessary for productive use. A free Amazon EC2 instance T2 Micro (https://aws.amazon.com/ec2/instance-types/) running on Ubuntu was used to make the first tests.

```
ubuntu@ip-172-31-12-170:~/keyservice$ sudo tclsh 0.5/scan_sequence.tcl -n 100000
[80051_100000]: scan job start: 2015-05-25/22:21:27
dat/exclude.conf: excluding 122 ranges from file
Starting masscan 1.0.3 (http://bit.ly/14GZzcT) at 2015-05-25 22:21:41 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 86741 hosts [7 ports/host]
[80051_100000]: scan job finished: 2015-05-25/22:22:08
[80051_100000]: approx. traffic: 3798212 bytes
[80051_100000]: 100000 IPs scanned in 00:00:41 (2439.0/sec), 187 SSL related IPs (0.19% -
     4.56/sec), 16 certs
```

The results were very promising as with this minimal setup a scan rate of **2439.0** IPs/sec was reached using a random IP package of 100.000 addresses. This tool was chosen as Level I scanner for the project. Refer to table 6.2 on page 43 for further test results.

5.2 Scanning - Level II

The Level II handshake expects an IP address and a port number (i. e. the output from Level I - see section §5.1) for the TLS service which is sufficient for connecting to this port and trying the TLS handshake to extract all necessary data. If the connection has been established the state of the connection is checked to confirm the validity of the connection and further information which is necessary for the project.

A server certificate contains all necessary data for the TLS connection security but also additional information which is not necessary for extracting the RSA keys only.

The necessary steps for a task are:

1. Connect to a TLS related service by initiating a TLS handshake.
2. Retrieve the X509 certificate data or at least the fields which are necessary for extracting the RSA keys.
3. Manage the data which is extracted in the two prior steps in a persistent way.

The last step needs to manage all stored data to identify a specific service. This is necessary to repeat scans for updating the data and to implement some sort of "data life cycle" (refer to section 6.5.3). This concerns also to the separation of the relevant actual from other historically based data and it should not distract the service itself (like invalid certificates, etc.).

According to these constraints again a database was chosen for storing all necessary data. Additionally the certificate files have been saved into the file system as it is easy to store, manage and retrieve it on demand.

Remark: The stored certificate files can also be seen as "backup" of the stored data because all information can be re-extracted if necessary but this step is not compulsive for the data management in general.

5.2.1 Connecting to a service

Network communications are widely used and implemented in all sort of products. Base level technologies exist as system libraries, command line tools and as extensions for most scripting languages.

Remark: It was foreseeable that highly specialized tools (like high speed scanners) in combination with many data management steps (e. g. parsing the output of other programs or managing the input for parallel scanning, etc.) and different favored tools for specific working steps will be used. This would make the use of a scripting language worthwhile for controlling the whole process.

5.2.2 TLS-Handshake

The reference implementation of the OpenSSL command line tool **"openssl"** was tried first to get the required information for this step. In this context the fact that this tool implements everything for the complete secure communication process (it is the "reference implementation" for the whole OpenSSL suite) it was a big help for analyzing any details. But the timing behavior rendered this tool as not sufficient, even more because the interest on data is limited to the part only which is necessary to fulfill our service's needs. The tests proved that calling an extra process for a single address would create far too much overhead.

The three scanning steps which are connect to a TLS service, retrieve and parse the certificate and manage the found data must be done in short time.

Runtime constraints using interpreters for parsing data or starting the interpreter will always limit the throughput. But if the binary network communication part uses significantly more time than the data handling itself a script solution may be sufficient as "glue" technology for process controlling. In general the most sensible argument for using that type of work frame was that most part of the data management is done in the context of the management process (read/write files) and also the database communication (connect to database, insert, update, delete, disconnect) which both are implemented as compiled C-code[1].

5.2.2.1 Historical background

In summer 2014 a project at the FH-Joanneum was completed with the goal to verify the mathematical solution for D. J. Bernstein's coprime factorization (refer to section §2.4 for detail). In that project a script implementation using the scripting language TCL (https://www.tcl.tk/) was tested to verify the algorithm and to compare the runtime behavior with a native C-implementation. The runtime restrictions for the mathematical part have been proven as insufficient, but the fact that a binary TLS module exists convinced the author to use this multi-platform language as main controlling part.

The TLS implementation for TCL is a typical interface to the OpenSSL library for executing the handshake in a transparent way without disclosing the internal details of the SSL data. Basically it just confirms the successful authentication/identification of the connection setup, furthermore also the encryption and decryption steps are made without any feedback, etc. The web pages http://tls.sourceforge.net/-tls.htm describe the usage of the module.

[1]For the communication with the database a TCL-only PostgreSQL interface was added for portability and test reasons but for the productive work this module was not used.

5.2.3 Retrieve certificate data module

The RFC5246(Dierks and Rescorla, 2008) defines the technical background for the very complex SSL protocol over more than 100 pages, ca. ten pages are used to define the "Protocol Data Structures and Constant Values". But for the planned project only a few data objects are necessary to achieve the project's goal to identify broken key values.

By default only a small part of the SSL meta data is directly accessible to the user in the standard implementation of the TCL-TLS module (refer to table 5.3).

X509 Field	Description
issuer dn	The distinguished name (DN) of the certificate issuer
subject dn	The distinguished name (DN) of the certificate subject
notBefore date	The begin date for the validity of the certificate
notAfter date	The expiry date for the validity of the certificate
serial h	The serial number of the certificate in hex
cipher cipher	The current cipher in use between the client and server channels
sbits n	The number of bits used for the session key
sha1_hash hash	The certificate hash (not documented in the man page)

Table 5.3: Default provided meta data

Example Listing:

```
issuer    = CN=TERENA SSL CA 2,O=TERENA,L=Amsterdam,ST=Noord-Holland,C=NL
subject   = CN=www.fh-joanneum.at,OU=Domain Control Validated
notBefore = Apr 24 00:00:00 2015 GMT
notAfter  = May 21 23:59:59 2018 GMT
serial    = 35CEC0A2E87CB615F3B40441AE618809
cipher    = ECDHE-RSA-AES256-SHA384
sbits     = 256
sha1_hash = CD608B55BED51B6E3BDD3805F16E04A50B6F72F5
```

As the module covers the complete TLS-handshake procedure, the decryption and encryption routines, all required additional TLS related information must already present in the internal structures and can be requested by SSL library calls.

TCL provides a very powerful C-library interface (http://www.tcl.tk/man/tcl8.5/TclLib/contents.htm) which makes it possible to add functionality programmed in C/C++. With that background the decision to write a program extension to the existing Open Source module was made.

The existing TLS-module has been extended to provide the additional components from the TLS-handshake (see table 5.4) which are necessary for the "FJreSafe" project. The great book "Network Security with OpenSSL, Cryptography for Secure Communications"(John Viega, 2002) and the description "Parsing X.509 Certificates with OpenSSL and CCVE-2014-0160"(Durumeric, 2013) were a big help to add this extension in reasonable time.

Remark: The advantage of this strategy is that all certificate handling is processed in native compiled efficient and as secure known library code.

To implement this library part the source code for "tls1.6.4" of the TLS module has been downloaded from "http://tls.sourceforge.net" and all new functionality has been added to the file "tls.c" only.

Remark: In the development test phase a bug has been found and repaired in file "tlsX509.c". The existing implementation of ASN1_UTCTIME_tostr() did not implement the parsing of the OpenSSL type V_ASN1_GENERALIZEDTIME correct which failed in some situations with a "Bad time value" error. The code for repairing this bug was inspired by a discussion on the web portal "http://stackoverflow.com/questions/10975542/asn1-time-to-time-t-conversion", the required format is described in (X.690, 2015).

The new library add on

The result was the new TCL command "tls::certdata" implementing the following syntax for the TLS module:

```
% package require -exact tls 1.6.4.1
1.6.4.1
% tls::certdata
wrong # args: should be "tls::certdata  ?-cert? ?-pem? ?-der? ?-local? channel"
```

The command returns the additional required information as a list of key-value pairs describing the connected peer similarly to the already existing command "tls::status" (http://tls.sourceforge.net/tls.htm).

The new implemented SSL objects are described in table 5.4.

X509 Fields	Description
chain dn list	Certificate chain according (Cooper et al., 2008) as from-to DN list
e h	Exponent for the encryption in hex (refer to 1.2.1)
d h	Exponent for the decryption in hex (refer to 1.2.1)
l n	Size of modulus in bits (refer to section 8.8)
n h	Modulus of the public key in hex (refer to 1.2.1)
p h	First private prime factor in hex (not used in public keys)
q h	Second private prime factor in hex (not used in public keys)
typ text	Key type "rsa" if EVP_PKEY_RSA(John Viega, 2002, p.205) or omitted
version n	X509 certificate version as decimal number
cert	Complete X509-certificate in format PEM, DER or raw hex

Table 5.4: Extended certificate data

Especially for getting the complete certificate the following options can be used:

-cert this option retrieves complete certificate as raw hex bytes without format option

- **-pem** if specified as format option the certificate is retrieved in PEM format(Linn, 1993)
- **-der** if specified the certificate is retrieved in the binary DER format(X.690, 2015)

Modeled after the command "tls::status" the option **-local** can be used to retrieve the certificate information from the locally used one.

Public key example (status and certdata fields without "-cert" option):

```
cipher    = ECDHE-RSA-AES256-GCM-SHA384
issuer    = CN=RapidSSL SHA256 CA - G3,O=GeoTrust Inc.,C=US
notAfter  = Jun  4 07:26:21 2018 GMT
notBefore = Jul  7 00:14:47 2015 GMT
sbits     = 256
serial    = 058CAA
sha1_hash = C8E632259113CC94BC8F9720455243214FEC7467
subject   = CN=*.hybridserver.at,OU=Domain Control Validated - RapidSSL(R),
            OU=See www.rapidssl.com/resources/cps (c)14,OU=GT18085679
chain     = {/OU=GT18085679/OU=See www.rapidssl.com/resources/cps (c)14\
             /OU=Domain Control Validated - RapidSSL(R)/CN=*.hybridserver.at}
            {/C=US/O=GeoTrust Inc./CN=RapidSSL SHA256 CA - G3}
            {/C=US/O=GeoTrust Inc./CN=RapidSSL SHA256 CA - G3}
            {/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA}
e         = 10001
issuer    = CN=RapidSSL SHA256 CA - G3,O=GeoTrust Inc.,C=US
l         = 2048
n         = A962BE5A7C88D6DD7B7BE622FBE41786E6BB2F0E3205FFA29A6ED5C9EB24E40BE0C8A\
            91D21D9EB392F0662E4D89905B0B57BFB6DFAEFC9551624451C82B78C9D0D11A08736\
            9773AEA3C64F04D488C227E5C8E44D7B0CA55526978F4A6A36A03A0B5D515C51BB93E\
            389D9AFC9653B13A197BA1F22359807153507DF8740D43BD0038D105AF36A187F29F7\
            E00A19BF372110CB338C24D8C601AC31B06AC0C954BF39C652C55E352A2E9F25210B4\
            68E4AC0CDC5943E15EACCF02835439194C1758E71587B489C0A75A4CB7F3EF4A88945\
            F079E7FD6891B4B9A2461C93740ABF729EE3E79BF4378EFEDCFA64226F876C0E261DE\
            30E3BFDD99BEA522C718740258B9F
typ       = rsa
version   = 2
```

Known limitations: After calling the command "tls::handshake" the command "tls::status" can be used for getting the documented TLS status fields and some fields are filled in by the new tls::certdata command. To get all possible TLS related data objects it is necessary to call all three commands in order: handshake, status and certdata.

5.2.4 Additional reasons using TCL as "glue"

The multi platform programming language has a very good integration into the various operating systems which makes it easy to switch the project or parts of it to other platforms.

Furthermore the graphical **T**ool**K**it (TK, see https://www.tcl.tk/) is integrated part of typical TCL distributions and installations. It allows quick creation of GUI components, graphical visualization of data, etc. The combination of the two tools and the large publicly available libraries and modules makes it easy to create analysis data and charts from all sort of output, like log files, etc. Almost all evaluation charts in this work were created using the Plotchart library contained in the package "Tklib" (http://www.tcl.tk/software/tklib/).

5.3 Hardware environment and operating system

The limits defined in chapter 4 describe the lowest level for the hardware to be used. The hard disk, memory and CPU have been estimated. The fact that these machines must guarantee a specific performance and taking into account that it is rather likely that it becomes "blocked" (refer to section §8.4) induced the decision to use a root server exclusively for this project. A somewhere else used server could disturb the normal functionality if blocked.

Some preliminary tests have been made with a private workstation, a free Amazon micro instance (refer to section 5.1.4) and with an Amazon EC2 Linux m3.2xlarge instance (refer to section 6.2.4.1). For productive use a dedicated machine as root server has been hired, refer to appendix D on page 134 for technical details.

As Operating System Debian 7 (https://www.debian.org/) was installed in the first project phase and Debian 8 in the second one. By saving the old keyservice main directory tree and copying only this archive to the new machine confirmed the portability of the concept (see section §5.4 below) and made the seamless migration easy.

5.4 Flexibility and portability

No technical detailed description was found which describes a methodology how to scan the complete Internet, especially for the targeted solution. To get a method for collecting as much as possible RSA-based TLS certificates a complete step by step guide has to be developed. As such a work depends strongly on the used environment a basic portable structure was created to bring the setup easy into a new environment.

The basic idea was to define a few requirements for installation and all other dependencies (like software modules, used libraries, utilities, result data, etc.) shall be easy installable (see also section 5.2.4) or have to be delivered in one package. The final approach was to store all data specific information, i. e. the developed software modules and the prepared and collected data into single subdirectories. This main directory contains only subdirectories for dedicated tasks. By moving this top level directory to another hardware all self-created data is transferred and a few necessary common software package must be installed to get it run.

Of course when reusing the database content is planned the database content must be exported from the database in the source system and then imported into the new target database.

A short description of this structure and the program usage information of most components are available in appendix A.

Chapter 6

Final Scan Solution

This chapter describes the final used setup of the determined scanning environment and methodology. More specific tests led to a better solution and better understanding of the process and its influence factors. The optimization of the runtime behavior within the test environment should help to optimize the results and should finally ensure the quality of the planned service.

According to the basic definitions in chapter 2 on page 14 the conceptual considerations in chapter 4 on page 22 and the discussions of the test results in chapter 5 on page 28 the environment for the final solution has been fixed and implemented.

The final solution has to cover the following areas:

1. Data collection
   (a) Level I scanning method
   (b) Level II scanning method
2. Task improvement by using parallel processing
3. Process measurement and control methods
4. Prototype of the user interface for end users

6.1 Project timing constraints

The project was executed in two steps. The first one was finished in September 2015 and the second one which was active ca. from June until August 2016. Both phases had their own database areas and allow to compare the same processes in different time frames with a time offset of ca. one year. In the last phase both databases have been joined to get a noticeable larger database which provides the possibility to evaluate the data handling with a significant higher amount of data.

The part for the second phase was limited by time as the cancellation of the provider's server environment has been determined already at the begin of the part two.

6.2 Data collection

Due to the different handling of the IP addresses two different classes of scan jobs (Level I + II) have been defined (refer to section §4.2). Each scan type uses his own type of persistent storage mechanism:

1. The Full Scan Linear Table as complete set of IPV4 addresses: It "simulates" the intended database storage by a file (refer to section 4.4.1) and covers the complete IPV4 address space. The goal of this table is to separate the "available" from the "interesting" part of all IP addresses (i. e. addresses with possibly TLS related targets). According to the scan level they are also classified as Level I addresses.
2. The active monitored and managed TLS addresses for the keyservice project: As these addresses are a rather small part of the complete IPV4 address space only that part of the addresses is relevant to the project. The class of Level II addresses are stored in a database.

Nevertheless there are circumstances that imply the change of an IP address from one class to another. Examples are addresses which are not in use for a TLS service anymore or ones which have not been used before in a TLS context and that state has been changed or vice versa, etc. These transitions must be considered when managing the data.

6.2.1 Randomized linear IP-table

Even the first attempts to scan the Internet from a normal Office PC led to troubles with some companies which interpreted the scanning tests as malicious attacks (see also section 8.4 on page 84). The main reason was that a linear address range scan with a random start value was tried. A linear scan contacts some routers or access nodes very frequently and this is a typical pattern which triggers alarms of IDSs.

Many scanners allow to randomize addresses to avoid this situation. A full IPV4 scan accessing any address randomly would be possible in general in a relatively short time but if parts of the addresses are blocked or if the scan must be repeated for some reason it is not traceable anymore which addresses have been scanned and which failed. The idea to avoid this was to prepare a Randomized Linear Table which allows also the random pattern access but scanning can be repeated in guaranteed identical order.

6.2.2 Preparation of the Level I database

Any IP in the address range of IPV4 is handled as 4-byte integer in a linear table with an extra byte as possible flag value. It contains all IPV4 addresses and eight flag bits (or 256 flag values) to allow handling a possible scanning state for any address.[1] Technically seen a simple random access file from 0 ... 0xff ff ff ff with one extra byte data is written to a binary file. Thus the offset can be calculated and a series of bytes can be read as more or less "direct operating system action".

The first step of the scan process writes all defined IP addresses of the IPV4 address range (i. e. 2^{32} entries) and one byte extra information into a normal binary file on a hard disk. Each entry becomes initialized with the position in this range, i. e. 0 for 0.0.0.0 and 0xff ff ff ff for 255.255.255.255. In this table any possible IP address is present.

The used algorithm is simple and straight forward. One single pointer locates a specific address and by adding an address range the scan block is defined. The address space can be assigned block-wise to a specific scan job only the "end of the block" offset must be stored persistent to allow the "next block" assignment (see figure 6.1).

[1]The reserved additional flag byte could be used to store information which allows to handle the Level I scan according to discovered states like "address not reachable", "reachability state changed", address is "blocked", etc. but this feature was not used in the project.

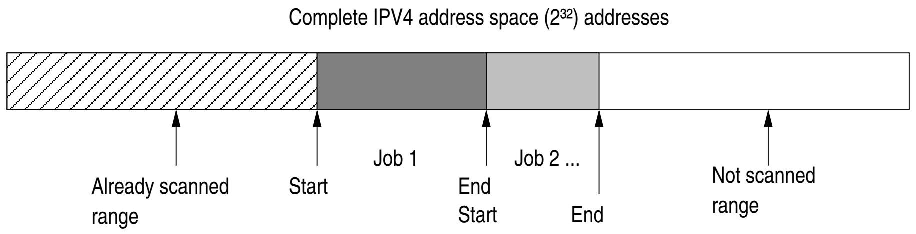


Figure 6.1: Full Scan Linear Address Table handling

This makes repeating while parallel scanning very easy.

6.2.2.1 IP-shuffling

A simple C-program called "ipshuffle" (for the usage refer to appendix A.2.4 on page 112) has been written to create an intended Randomized Linear IP-table. If the order of processing should be changed again this program must be called once again and the new result can be used for future scans.[2] The following functions have been implemented:

1. Create an array containing of all IPV4 addresses (0.0.0.0 ... ff.ff.ff.ff) or parts of it.
2. Optional shuffling of all IP-addresses (i. e. exchange any address with a randomly selected one), optional with a specific srand() seed value.[3]
3. Write the array to a binary file, optionally write all address entries in reversed byte order (e. g. 192.168.12.38 → 38.12.168.192).

Program runtime for full IP range on FH-Joanneum server "bogota" (Dell PowerEdge 2900, 8 core Intel(R) Xeon(R) E5410 @ 2.33GHz) and the file size:

```
[cluster@bogota l-c]$ time bin/ipshuffle -n 0 -f dat/ipv4shuffle.dat
Running loop from 0 to 4294967295
assigned 4294967295 adresses
Hello universe, pointer size of iplist=8
Beginning shuffling of 4294967295 addresses ...
 ...reordered 4294967295 adresses
writing 4294967295 records to file 'dat/ipv4shuffle.dat'
sizeof(b_iplist)=5, count=4294967295
21474836475 bytes written to file 'dat/ipv4shuffle.dat'

real    23m18.450s
user    20m6.572s
sys     1m28.532s
```

[2]As the program allocates the complete address space in one call it did not work on a machine without a dedicated swap area. It failed with a 'Cannot allocate memory' error message independently of the physical available memory. It was necessary to create and activate a swap space to get the program running.

[3]The standard C library function void srand(unsigned int seed) initializes the random number generator used by the pseudo-random number function rand(void).

To proof the quality of the distribution a copy of the the result file has been compressed using the tool “gzip”.

```
[cluster@bogota 1-c]$ cp dat/ipv4shuffle.dat dat/ipv4shuffle.cpy
[cluster@bogota 1-c]$ gzip dat/ipv4shuffle.cpy
[cluster@bogota 1-c]$ ls -l dat/ipv4shuffle.*
-rw-rw-r-- 1 cluster cluster 21474836475 May 16 16:30 dat/ipv4shuffle.dat
-rw-rw-r-- 1 cluster cluster 19266227740 May 16 16:41 dat/ipv4shuffle.cpy.gz
```

The resultant compression ratio in percent was:

$$100 - \frac{19266227740}{21474836475} \times 100 = 10.28\%$$

A compression rate of 10% shows a good binary data distribution.

Statement: This file is used in all following Level I scan actions as “scan order” basis. That means that each repeated scan is done in the exact same order as the prior ones, regardless if it is a full or partial scan. Nevertheless the block size of a scan is the minimum part for handling repetitions.

6.2.3 Runtime investigation

Based on this initial situation a few speed tests have been made on different hardware configurations to get a feeling about the time behavior of the scan process. A private SOHO Server (4 Core with 8GB RAM) and two Amazon cloud services a free T2.micro and a commercial M3.2xlarge have been compared. The tested aspects were the number of CPU cores, the installed RAM, the network bandwidth and the size of the IP packets to scan (see table 6.1).

Family	Type	vCPUs	Memory	Storage (HD/SSD)	Network performance
Office Server	AMD Athlon 64	4	8 GB	2TB	Low
Amazon Cloud	T2.micro	1	2 GB	8GB	High
Amazon Cloud	M3.2xlarge	8	30 GB	80GB	High

Table 6.1: Different hardware setups

6.2.4 Single process scan tests

The already existing tool **“scan_sequ.tcl”** (refer to appendix A.1.7 on page 108 for the program usage) was used to test the complete cycle of single block scan actions.

6.2.4.1 Some typical single scan results (Scanning 19 ports, one million address block)

Low End server with very slow up-link speed.

```
scan_sequ.tcl -n 1000000
[188538_1000000]: scan job start: 2015-06-07/20:02:42
dat/exclude.conf: excluding 122 ranges from file
Starting masscan 1.0.3 (http://bit.ly/14GZzcT) at 2015-06-07 18:51:06 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
```

```
Scanning 867738 hosts [19 ports/host]
[188538_1000000]: scan job finished: 2015-06-07/20:53:10
[188538_1000000]: 1000000 IPs scanned in 00:50:28 (330.3/sec), 155 SSL related
    IPs (0.02% - 0.05/sec), 20 certs
```

Tests with Amazon EC2 Linux free T2.micro instance.

```
[80096_1000000]: 1000000 IPs scanned in 00:26:58 (618.0/sec), 359 SSL related
    IPs (0.04% - 0.22/sec), 17 certs
```

Tests with Amazon EC2 Linux M3.2xlarge.

```
[80101_1000000]: 1000000 IPs scanned in 00:20:42 (805.2/sec), 331 SSL related
    IPs (0.03% - 0.27/sec), 16 certs
```

As result can be said that the network bandwidth has the major influence when calling a single scan process. But according to the reached values and taken the basic assumptions (see section §4.1) into account it was obvious that massive parallelizing will be necessary to get the required results.

6.2.4.2 Influence of the scan block size

While testing with a varying number of IP addresses (scan block sizes) significantly different runtime has been recognized. Therefore further tests to determine the optimal scan block size have been made.

Job/ID	IPs	Time	IP/sec	TLS	%TLS	TLS/sec	Certs
80206	100	00:00:14	7.1	0	0.00	0.00	0
80306	1000	00:00:14	71.4	3	0.30	0.30	0
1196306	5000	00:00:13	384.6	32	0.64	0.64	0
81306	10000	00:00:17	769.2	48	0.48	0.48	1
1201306	50000	00:00:17	2941.2	240	0.48	0.48	7
91306	100000	00:00:27	3703.7	208	0.21	0.21	7
1251306	500000	00:05:26	1533.7	317	0.06	0.06	14
191306	1000000	00:20:45	803.2	316	0.03	0.03	11

Table 6.2: Masscan tests with different package sizes

In this test series on the M3.2xlarge a maximum at a block size of 100.000 addresses became conspicuous (see table 6.2). To ensure that this does not apply only on the specific hardware another test was made with the other Amazon EC2 instance. The result shown in table 6.3 confirmed this behavior. In figure 6.2 on the next page this conspicuous peak is clearly visible in both tests.

Job/ID	IPs	Time	IP/sec	TLS	%TLS	TLS/sec	Certs	CPU
4294950000	17295	00:00:17	1017.4	7	0.04	0.41	7	54%
0001100000	100000	00:36.90	2702.7	697	0.70	18.84	14	64%
0001200000	150000	00:91:04	2348.8	1010	0.67	15.78	16	80%
0000900000	200000	01:45.52	1904.8	1354	0.68	12.90	30	87%

Table 6.3: Masscan tests on Amazon EC2

Statement: As the maximum at a packet size of 100.000 has been confirmed in several measurements this value was chosen as “standard packet size” for the productive scans.

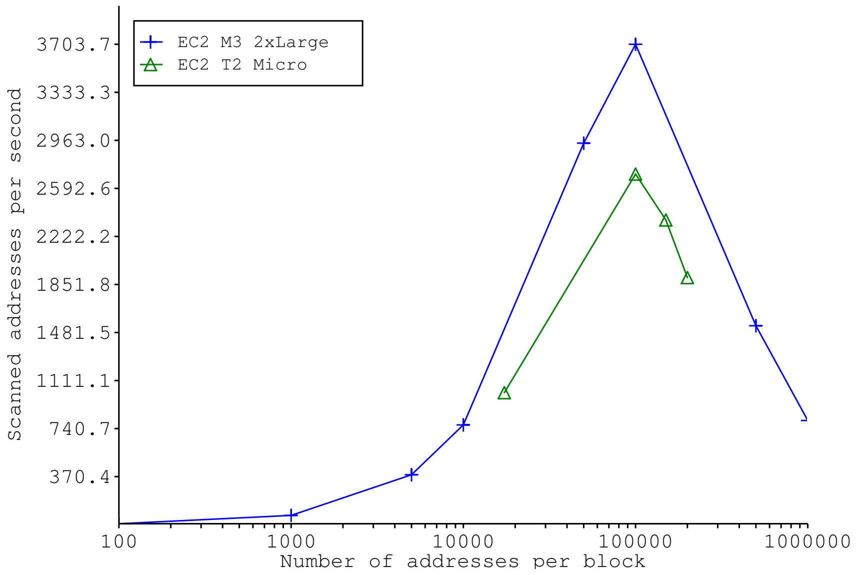


Figure 6.2: Scan rate depending on the packet size

6.2.5 Parallel scan tests

One important fact for massive scanning with controlling the input and post processing the retrieved data is the chance to scale the process. A simple setup by starting two processes nearly at the same time with a block size of 100.000 addresses confirmed that the process is well scalable.

```
Scan started: 0753700000_100000 2015-08-22/22:28:09
[0753700000_100000]: 100000 IPs scanned in 00:01:41 (990.1/sec), 834 SSL
    related IPs (0.83% - 8.26/sec), unknown=1 ssl=5 ssh=10 open=813
    db_update=2 db_insert=719 complete=834 X509=5
Scan finished: 0753700000_100000 2015-08-22/22:29:50

Scan started: 0753800000_100000 2015-08-22/22:28:10
[0753800000_100000]: 100000 IPs scanned in 00:01:42 (980.4/sec), 911 SSL
    related IPs (0.91% - 8.93/sec), unknown=1 ssl=4 ssh=4 open=899 db_update
    =2 db_insert=777 complete=911 X509=3
Scan finished: 0753800000_100000 2015-08-22/22:29:52
```

200.000 IPs has been scanned in 00:00:103 which is a scan rate of **1941.7** IPs per second. The sum of both scan results is with **1970.5** only a little bit higher.

Remark: Parallel processing cannot be judged only by starting several processes at the same time and adding the single results. The effective increase of the throughput is not identical to the number of processes multiplied with the runtime of a single scan block. With the number of processes the runtime for the management for each process also increases.

Other involved parts are different system components which influence the overall performance. Typical retarding process parts are database actions, device I/O, additional calculations while collecting, monitoring processes, etc.

6.2.5.1 Detailed log file and meta data analysis

A detailed look at all involved files and processes for one scan task is used to understand the behavior of this method.

In sequence the following steps have to be executed:

1. The wrapper and management script **"scan_sequ.tcl"** (for usage message see appendix A.1.7 on page 108) is started. The configuration parameters are read from the configuration file **"etc/-file_db.cfg"** (appendix A.8.3 on page 117) and **"etc/scan_db.cfg"** (section A.8.4).
2. This script reads the current index pointer from the "Randomized Linear IP-Table" (see also section 6.2.1). Then the number of IP addresses is read from the file and the index pointer is incremented by the blocksize value (the block size is defined in scan_db.cfg or specified on command line) and written back to the sequence file.
3. The address list is then written to a temporary input file. In this example it is **"tmp/0753800000-__100000.txt"**. File time and size information is available in appendix C.3.4 on page 133.
4. The external scanner executable **"masscan"** (for usage refer to appendix A.2.3 on page 111) is started via the plugin interface (see appendix A.6 on page 114 for details) using the temporary input file written before. The standard output of the scanning executable and other program part's output become written to the analyzed logfile **"log/scan_20150822_10000___1_7306.log**" (content shown in appendix C.3.3 on page 132).
5. The scanner executable writes the results to the temporary output file **"tmp/0753800000_-100000.csv"** (file time meta information in are available in appendix C.3.4 on page 133) and terminates.
6. The plugin module parses the temporary output file **"tmp/0753800000_100000.csv"** into a predefined interface format and the management script writes it to the SQL database table **"scan"** (for the database structure refer to appendix B.1.1 on page 126).

The single steps and their timing values are conflated in table 6.4.

Object	Time	Sec.	Remark
Scan job start	2015-08-22 22:28:10	2	Job ID 0753800000_100000
Temp input finished	2015-08-22 22:28:12	34	0753800000_100000.txt (100000 lines)
Masscan start	2015-08-22 22:28:46	105	reported 20:28:46, timezone corr.
Temp output finished	2015-08-22 22:29:51	1	0753800000_100000.csv (913 lines)
Log file closed	2015-08-22 22:29:52		scan_20150822_10000___1_7306.log
Scan job end	2015-08-22 22:29:52		overall runtime: 102 sec. (00:01:42)

Table 6.4: Scan job timing

A time chart created from table 6.4 discovers possibilities for optimization (see Figure 6.3 on the following page). The overhead of the startup and the post-processing steps are negligible but the preprocessing step uses 1/3 of the time from the complete scan process. This is most probably due to the file IO steps, i. e. reading ca. 1/2 MB of binary data (5 byte multiplied with the the block size), converting it to readable IP-addresses and writing the input file to the hard disk which again gives a file of an approximate size of 1.4 MB.

This is obviously an issue for future optimization.

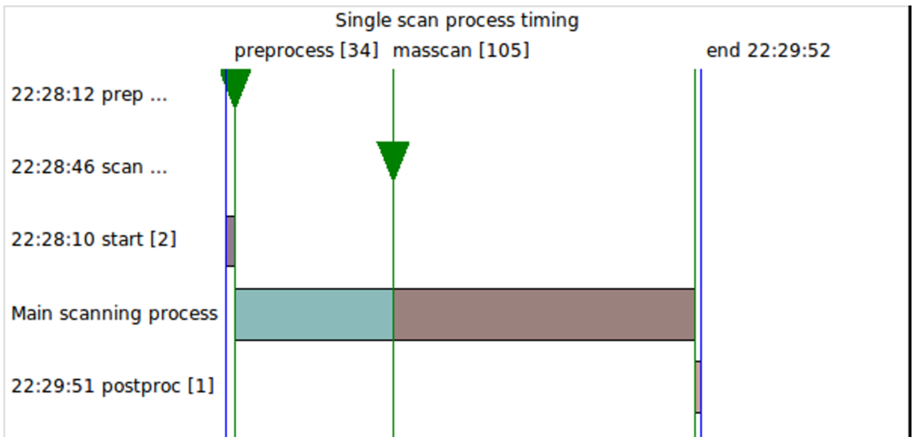


Figure 6.3: Scan job time chart

6.2.5.2 Parallel scan speed measurement

To measure the real scan speed a simple command line statement was used to check the overall result. It uses the sequence pointer and the block scan runtime values. Because the sequence pointer of the Linear Address Table (refer to section 6.2.1) is increased on any scan block request it can be used as rough value to estimate the "effective" scan rate.

```
SECS=60;START=$(cat dat/ipv4shuffle_20150624.seq);\
sleep $SECS; ENDE=$(cat dat/ipv4shuffle_20150624.seq); \
RES=$(echo "scale=3; ($ENDE-$START)/$SECS" | bc); echo $RES $ENDE-$START $RES
```

Applying this example to the values from table 6.5 results an average scan rate of **4188.8** addresses per second for this setup.

scans	secs	scans/sec
240000	60	4000.0
300000	60	5000.0
26666.6	60	4444.0
150000	60	2500.0
300000	60	5000.0

Table 6.5: Effective scan rate

This sequence file evaluation example measures the number of "tried" scans not taking into account how many successful attempts have been made i. e. how many IP addresses were found with possible TLS related services.

Sources for answering this question are the sequence values for the database tables "scan" and "rsa_certs" (i. e. the sequence names "next_scanval" and "next_rsa_val" in the table "seq_table") or the auto-generated max(ID) values of these two tables while importing.

A script code example for the first case using “next_scanval” could look similar to:

```
while true; do
  psql -h localhost -U keymaster -d keyservice -c \
        "select now(),value from seq_table where name='next_scanval';"|\
         sed -n '3,3p'|tee -a log/retrieve_seq_$(date +%Y-%m-%d).log
  sleep 300
done
```

This script creates in about every five minutes (wait period W is 300 seconds) a log entry with the latest “next_scanval” sequence value S from the table “seq_value” (refer to appendix B.1.3 on page 127 for the table definition and examples) in the log file “log/retrieve_seq_2015-08-25.log” using the following format:

```
 2015-08-25 23:15:17.987979+02 | 47640
 2015-08-25 23:20:18.526297+02 | 132140
 2015-08-25 23:25:19.062381+02 | 218840
 2015-08-25 23:30:19.289489+02 | 294940
 2015-08-25 23:35:19.51653+02 | 382940
 2015-08-25 23:40:19.650092+02 | 474440
 2015-08-25 23:45:19.88909+02 | 558640
 2015-08-25 23:50:20.219764+02 | 648040
 2015-08-25 23:55:20.41744+02 | 738340
```

The evaluating script has to calculate the sum of the different values of adjacent sequences per second ($\triangle S/W$) divided by the number of wait periods n.

$$\frac{\sum_{i=2}^{n} \frac{S_{i-1}-S_i}{W}}{n-1}$$

This can be achieved by a simple one-liner using AWK syntax and assuming the values are written with a constant time offset value of five minutes (i. e. 300 seconds):

```
awk '{if (NR==1) {old=$NF; next} new=$NF; dif=(new-old)/300.0; sum+=dif; old=
    new;} END{print "Average value: " sum/(NR-1) " svcs/sec."}' log/
    retrieve_seq_2015-08-25.log
Average value: 287.792 svcs/sec.
```

The average scan rate is in this case **287.8** found services per second which consist of an IP address and a service port counted in the table “scan”.

Remark: For later evaluation more sophisticated scripts have been created which extract the timing values from the log records and therefore it delivers more exact timing values and it can be used even with changing wait periods.

6.2.5.3 Scanning speed and values distribution

Another important performance indicator is the long-term behavior of the scan rate because the prior “sequence scan” evaluation provides just a single mean value over a fixed observation period. This information can also be found in the database results by grouping the inserted records into time slots.

The following SQL query was used to get the number of found service records per hour within the last two days by building hourly groups:

```
select count(*),to_char(seen_at, 'YYYY-MM-DD HH24') from scan
  where seen_at >= '2015-08-21 00:00:00'
  group by to_char(seen_at, 'YYYY-MM-DD HH24')
  order by to_char(seen_at, 'YYYY-MM-DD HH24');
```

The corresponding values can be used to present the results in a chart, see the red "stored" lines in figure 6.4.

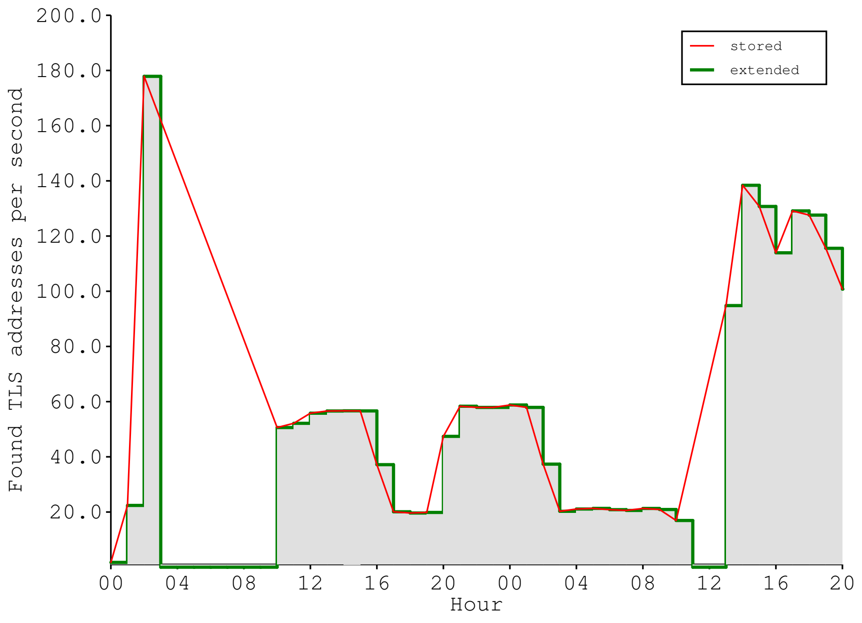


Figure 6.4: TLS scan rate over ca. two days

Explanation: This image may give a wrong impression about the scan progress due to the methodology of the data extraction in combination with the chart representation. In this case it is not visible if in a specific time range values have been collected or not. In this example from 3:00 . . . 10:00 and between 11:00 and 13:00 no value has been collected and the intrinsic curves would occlude this (the red "stored" chart lines). To get the definitive grouped values the green histogram chart (the "extended" view) has been calculated.

In appendix C.3.2 on page 132 the values for this chart can be found.

6.2.5.4 Memory usage

As another important criterion for long-term collection the memory consumption was observed using the tool Monitorix (http://www.monitorix.org/).

The figure 6.5 shows that the memory usage (the small red "Used" part on top of the green "Cached" memory in the graphic) was continuous. No "memory leak" could be found in the scanning part. The memory cached part is increased due to the memory organization of the operating system.

In other areas like the public key export function memory limits became obvious which will be discussed later.

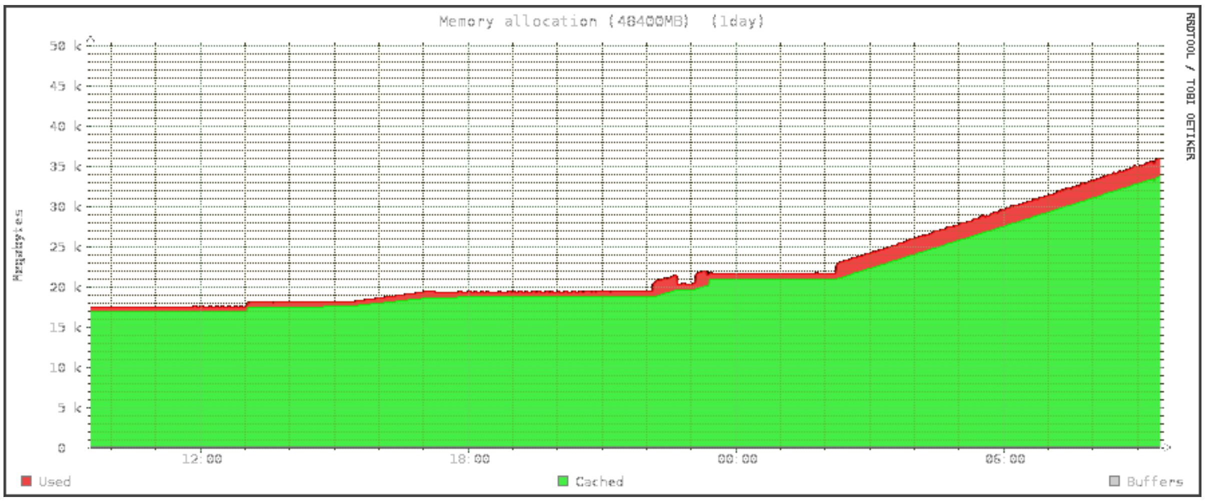


Figure 6.5: Memory usage

6.3 Parallel scanning

The investigation in section 6.2.5 confirmed that the start of more scan jobs increases the throughput significantly. A very simple method to start more jobs at almost the same time is to use the shell syntax for background processing.

```
./0.7/scan_sequ.tcl &
./0.7/scan_sequ.tcl &
./0.7/scan_sequ.tcl &
```

In this example the Level I scan script **"scan_sequ.tcl"** (refer to appendix A.1.7 on page 108 for the program usage) has been started three times. This was done by using the control character "&" at the end of the command. It executes the command as background task in Linux/Unix environments and the command line can be used immediately again after entering the command.

But there are several disadvantages using this method:

1. All jobs start at nearly the same time as background processes.
2. The jobs will probably run for different periods of time (i. e. they end after variable times).
3. New tasks can be started again when enough resources are available.
4. It is therefore not foreseeable when the next jobs shall be started.

Especially the last point is not easy to handle because if the following tasks are started after a too long delay, the machine is not utilized as it could be and if the delay is too short, it may overload the system with too much tasks.

6.3.1 Timed Loop approach

The first approach was to include some time to wait for the loop to start the next task. The idea was that a well selected process wait value will create a balance between the number of finished and the number of started jobs. A simplified script could look like:

```
while true
do sudo ./0.7/scan_sequ.tcl -n 10000 &
   sleep 6 # wait time in seconds
done
```

In general this simple mechanism was able to create a more or less constant process throughput but the system load situation was not as constant as expected and sometime it became overloaded and went into some sort of lock situation. A more detailed description can be found in the implementation description in section 7.2.2.

6.3.2 Controlled Loop approach

To circumvent this problem the following task control scheme has been applied (see figure 6.6). It can be described by the following pseudo syntax:

1. Check if a specific number of running tasks (**JOB_LIMIT**) has been reached
    (a) If yes, wait some delay defined in (**EXTRA_DELAY**) and continue with step 1.
2. Start a scan job as background task with a defined block size (**SCANJOB_SIZE**)
3. Wait a specific time in seconds (**JOBWAIT_TIME**)[4] and continue with step 1.

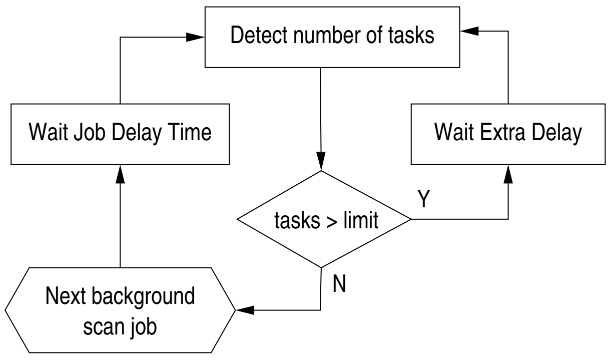


Figure 6.6: Job flow control

[4]The JOBWAIT_TIME delay ensures that not all tasks are started at nearly the same time. It portions the tasks to spreads them over a minimum time range.

This will converge in a defined process load which is approximately constant in some range depending on the parameters but the number of concurrent tasks will be limited. The loop counter (MAX_SCAN_SEQ) is used to avoid the endless loop situation which ensures that the script is ended after some maximum runs.

The variable parameters had to be determined to ensure the expected scan rate and a simplified shell code with typical values is shown here:

```
num_procs()
{ # $1 ..taskname
  ps -ef|grep "$1"|egrep -vw "grep|sudo|$0"|wc -l
}
SCANJOB_SIZE=100000 # number of scan records per job
MAX_SCAN_SEQ=10000  # number of scan jobs
JOBWAIT_TIME=0.7    # wait seconds between jobs
EXTRA_DELAY=10      # wait if task limit is reached
JOB_LIMIT=25        # maximum number of parallel jobs
JOB_NUMBER=1        # job loop counter
JOB_NAME="./0.8/scan_sequ.tcl"
while [ $JOB_NUMBER -lt $MAX_SCAN_SEQ ]
do RUN_JOBS=$(num_procs $JOB_NAME)
   if [ $RUN_JOBS -gt $JOB_LIMIT ]
   then sleep $EXTRA_DELAY; continue
   fi
   sudo $JOB_NAME -n $SCANJOB_SIZE & # must be run as root
   sleep $JOBWAIT_TIME
   JOB_NUMBER=$(($JOB_NUMBER+1))
done
```

Using this approach more investigations have been made to optimize the scan rate according to the used system environment. As described in section 6.2.5 a log file is written for each scan block and therefore this content can be used to analyze the behavior of the parallel scanning process.

The shell script **"sta/stats_parscan.sh"** (see appendix G.1.12 on page 150 for the call syntax) has been written which reads a complete directory of specific log files and creates an overview about the single processes and calculates a complete summary statistics. As example a small timespan of 2:30 minutes as start range has been selected to analyze the time behavior which can be seen in the time chart in figure 6.7 on the next page. The reduced output of this example is shown:

```
$ sta/stats_parscan.sh -b "2015-08-22 15:00:00" -e "2015-08-22 15:02:30"
==================================================================================================
      From               Until           BlkSize   Time    IPs/sec SSL  %SSL SSL/sec   Scan ID
--------------------------------------------------------------------------------------------------
2015-08-22 15:00:01  2015-08-22 15:02:00  100000  00:01:59  840.3  554  0.55  4.66  0143400000_100000
2015-08-22 15:00:11  2015-08-22 15:02:12  100000  00:02:01  826.4  528  0.53  4.36  0143500000_100000
2015-08-22 15:00:21  2015-08-22 15:02:28  100000  00:02:07  787.4  642  0.64  5.06  0143600000_100000
...
2015-08-22 15:02:12  2015-08-22 15:04:21  100000  00:02:09  775.2  540  0.54  4.19  0146300000_100000
2015-08-22 15:02:22  2015-08-22 15:04:32  100000  00:02:10  769.2  666  0.67  5.12  0146400000_100000
--------------------------------------------------------------------------------------------------
    Scan start            Scan end       Blocks  IPs tot.  SSL rel.  Scan time   IPs/sec  SSL/sec
2015-08-22 15:00:01  2015-08-22 15:04:32  36938   2800000   19873    0d 00:04:31  10332.1   73.3
==================================================================================================
```

Within an overall time range of 4 Minutes and 31 seconds 28 scan jobs with a block size of 100.000 IP addresses have been successfully finished. In this situation 25 tasks were executed in parallel (the green line in the middle of the time window of figure 6.7). The runtime summary shows a cleaned scan rate of **10332.1** IPs/sec for data of class A_1 according to section §6.4.

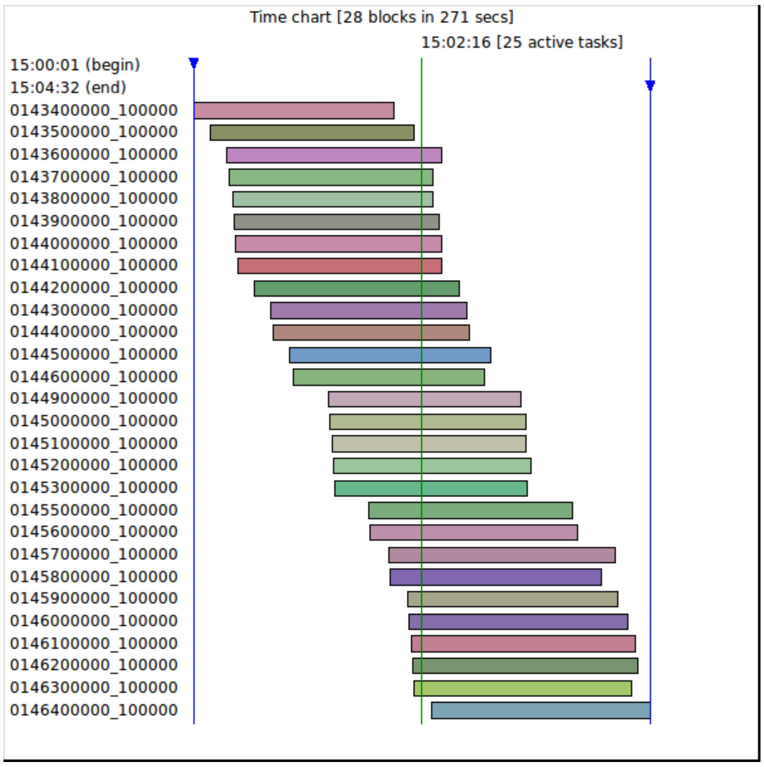


Figure 6.7: Parallel tasks time chart

6.3.3 Process controlling

In practice and even more in the development phase it was necessary to hold on the scan process e. g. for adapting the scan parameters or to repeat failed attempts. To achieve this goal a start script has been programmed which behaves like a conventional Unix system service.

The command must implement at least two command line options.

- This is the option “start” which starts one or more monitoring processes and the main background process loop (Controlled Loop Scan).
- The other one is the option “stop” which kills the background process loop (Controlled Loop Scan) and the monitoring processes. Then it waits until all running scan processes become finished.
- Optional the command line option “status” can be implemented which gives a short status how many scan jobs are running.

The control script **“scr/monitored_rsa.sh”** is used to explain the discussed mechanism. It initiates the following execution steps if the command line option “start” is used.

1. The script **“scr/par_scan_rsa.sh”** (see appendix A.10.3.2 on page 121) starts the Controlled Loop Scan process for certificate retrieval.
2. The script **“scr/monitor_procs.sh”** (see appendix A.10.2.1 on page 120) starts the process monitoring which writes every 60 seconds ($WAITTIME value) the number of processes with the name **“retrieve_cert.tcl”** ($REXECUTE script, see appendix A.1.4 on page 107) into the log file.

3. The script **"scr/log__dbinserts.sh"** starts the database monitoring which writes every 60 seconds ($WAITTIME value) two records into the database in "rsa" mode (the "next_scanval" sequence and the "max(rsa_certs.ID)" value). Refer to appendix A.10.2.2 on page 121 for the call syntax.

Refer to appendix A.10.3.4 on page 121 for the example call syntax and A.10.3.5 for the complete script code for the controlled Level II functionality.

6.3.4 Sequential scan order

This part of the document covers the parallelism of the scanning process. Therefore it is necessary to ensure that an input entity (e. g. an IP address) is only used once in the complete process. "Sequences" are used to address an entity in a unique way.

Sequences are common database features which ensure that a used ID value cannot be reused again. The database mechanism of a sequence works in a way that the "request" of a sequence number always increments the sequence counter even if this sequence value is not used further. But in case of the project's goal a gap-less number of sequence range values must be guaranteed because a "completely defined amount" of entities has to covererd, i. e. "all IPV4 addresses".

This requirement was the reason for implementing the special sequence mechanisms described in table 6.6.

Entity type	Implementation	Stored in
Level I scan address	Randomized Linear IP-Table	Plain file
Level I - TLS address	Sequence key "next_scanval"	Database
Level II - retrieve cert	Sequence key "next_rsa_val"	Database
Exported address	Sequence key "next_export"	Database
Joining RSA cert ID	Sequence key "next_joinval"	Database

Table 6.6: Sequence handling methods

A Level I sequence (section 6.2.1) is extracted by opening the text file which contains the current address offset. This value is read and incremented by a defined value (the scan block size) and finally this value is finally written back to the file. See also section 6.2.5.1. The read value is returned as "next_sequence_value" and a subsequent access will read the "current" value incremented by the requested block size, i. e. the start of the next block.

The other methods have a similar logic but the "distinct" values are stored as database record in the table "seq_value" (refer to appendix B.1.3 on page 127 for details). A sequence value can be read continually but the write access must appear always in conjunction with raising this value.[5]

Important in this context is that this implies that the "block size" is the minimum amount of entities which must be handled in one single step.

[5]Contrary to the database sequence functionality the current implementation does not guarantee a unique new (higher) value because concurrent access is not handled consistent. But on the other way easy handling is given which is convenient for manipulating the values during the development phase.

6.3.5 Measurement methods for evaluating the results

To compare the different results it is necessary to retrieve quantitative information for evaluating the used methods. In this project the following data sources have been used:

1. Temporary result files
2. Creation and evaluation of log files
3. Existing and new monitoring tools
4. Database queries

Remark: The evaluation tasks also led to an improvement of the scanning methods and the modified scan tasks got new demands to investigate specific features and impacts to get better results.

Temporary result files Many tasks are combined with program calls which produce a predefined result. This information is usually not directly usable within the process. The data is written into temporary files which will be processed by a later step.

The content may sometimes include time stamp information or quantities which are relevant for the evaluation and also for the measurement process. Even if no time information is available the file meta information (e. g. file date) can be used to extract time-line information. Examples can be seen in section 6.2.4 and 6.2.5.

Log file evaluations As in most unattended batch jobs it is common to write log file output for supervising passed off processes. Depending on that content the evaluations can be possible.

If the necessary data and the time information is present the basis for process evaluation is defined. A typical application is described at the end of section 6.3.2.

Monitoring tasks Many monitoring tools already exist for system survey and later error analyzing steps. These tools often provide ready-to-use output. Others allow the access to the collected data. Several special tasks have been programmed to store a basis of data for convenient later usage and post-processing.

Monitoring tools and tasks are used to create useful information, graphic charts, etc. Section 6.2.5.2 shows examples using sequence files and SQL queries and in section 6.2.5.4 the Open Source Tool “Monitorix” is used as example.

Database queries and interpretation Independent of the monitored system information which is vital for the systems optimization. The effective outcome of the scan process is the final result. Except the mentioned Level I methods for calculating the Level I results (refer to section §6.3). All other values are stored in the database.

Skilled database queries are a major source for getting quantified reports and results for almost all scanning situations. Section 6.2.5.3 shows an typical member of such an exponent.

6.4 Level I scan process

The scanning has been divided in two classes according to figure 2.2 on page 17, each of it has a defined input. Then this input is enriched by additional information and the result is saved. Mathematically seen is A_1 the usable range of basic data A with extra information.

$$A \to A_1$$

This extra information limits the amount of data A (in this case the complete IPV4 address range) to an amount of data A_1. For example by removing private IP range(Rekhter et al., 1996), broadcast/unspecified addresses(Mogul, 1984), excluded address ranges because a user or company does not want to become scanned, etc. This will finally be used for checking if there exists a related information for the project.[6]

The fact that a service consists of the information IP address and connection port (refer to section §6.2) the port number is added to an IP address (this information is called I_1) which is necessary to contact it over the Internet, to get the identifying information for this scan object called “service”.

$$A_1 \times I_1 \to S$$

The output of this step is S, the complete set of possibly related services. This information is stored in the database table “scan” (see section §B.1.1 for details).

Ascertainment: As the identification fields are “ip_address” and “service” each service must be unique.

There are several methods to check if a service is available and it depends on the scanning software how this is implemented and which possibilities are provided and used. The “masscan” program is highly compatible with the widely used “nmap” and configurable by command line options and configuration files. Many informations are available in IT Security literature and on the “nmap” home page (https://nmap.org/book/man-port-scanning-techniques.html). As masscan uses the TCP-SYN option “-sS” by default (also known as Stealth scan) most scans were done in that mode.

> SYN scan is the default and most popular scan option for good reasons. It can be performed quickly, scanning thousands of ports per second on a fast network not hampered by restrictive firewalls. It is also unobtrusive and stealthy since it never completes TCP connections.(Lyon, 2011, chap. 15)

In section §8.4 this project aspect will be discussed in more detail.

6.4.1 Level I service life cycle

The numeric database field “level” of the table “scan” is dedicated to manage the life cycle of a record in the database. In the first run of the scan a service may be reachable or not. If it is accessible, it is probably a service, but in this phase it is not checked in detail.

[6]The Private IP ranges and Broadcast addresses (all “1” and all “0”) are implemented in the masscan program logic, exclude lists are manually maintained and handled by masscan accordingly.

Several situations are possible:

1. The service is new and active. It is stored as a new possible TLS related service.
2. The service is already stored and active. The meta data is updated (e. g. field “seen_at”) and the scanning level is increased if it is beyond the configurable maximum value.
3. The service is not available. The fact that a service is not accessible anymore is an evidence for a missing service, but there could also be temporary problem. To cope this condition the “level” field is decreased. Only if a service is not available and the level is at 0 the record is deleted from the database.

Remark: To manage the state of a service by accessing it only in a manner as port scanners do, is no trivial task. Experience and observance of the communities reaction, communication with affected users and companies may lead to a change of this scanning philosophy in future.

6.5 Level II scan process

After running the steps described in section §6.4 a list of possibly TLS related IP addresses is available, stored in the table “scan” (see appendix B.1.1 on page 126 for technical details).

In the next step all these addresses are contacted and extract a valid X509 certificate if possible. If a valid certificate is found the information for factoring the RSA keys (see section §2.4) is extracted together with other information which is necessary to manage the key service. This “Scanning Level II” uses the mechanisms described in section §5.2. The results are stored in the table “rsa_certs” (see appendix B.1.2 on page 127 for details).

6.5.1 Certificate collection

Similar to the Level I scan process (see section §6.4) an input data amount of S (the complete list of possible related TLS services) exists, which maybe reduced to an amount of S_1 (e. g. by removing addresses from the list according to the exclude list logic or other manually managed services), this transformation step be formulated as:

$$S \rightarrow S_1$$

S_1 is the filtered list of expected services.

Remark: All entities which are stored in the table “rsa_certs” had at least once delivered information successfully in one certificate extract step.

I_2 is the “new” information in sense of certificate relevant information extracted in this scan step. This information can be added to existing records or entries can be removed from the existing set.

$$S_1 \pm I_2 \rightarrow C$$

The final set C is the basis for providing the information to the factorization step (refer to section §2.4).

Many certificates are used multiple times (wild card certificates for multiple hosts) but the factorization does only use single distinct public key values “**n**“ (see section 1.2.1) in the Copri-module.

$$C \to K$$

Retrieving only unique key values reveals K the unified set of the all available distinct values of C. This topic will be discussed more detailed in section 8.6.3.

6.5.2 The Copri-Interface

The set K of unique public keys is sent to the factorization module of the Copri service (see section §2.1) which are basically only large numeric values (limited by the key size).

If a calculation result will be transferred back the lookup in the RSA table will find a single or many different IP addresses and services. The detailed description about the searching back from a key value to the affected services is described in the post-processing part in section §8.2.

6.5.3 Level II service life cycle

Similarly to the Level I service life cycle in section 6.4.1 it is also necessary to take care of the time behavior and availability of the information. But unlike to the theoretical limited amount of information of Level I data (the complete IPV4 range multiplied by all available port numbers) the information in Level II is not limited.

Definition: The term “invalid key” will be used in this context as a synonym for
a n-factor which was retrieved from a revoked certificate or
which validation period is outdated (e. g. it was valid sometime ago) or
it comes from another source which has not been collected or created by this project.

The basic reason is that all certificates have a limited lifetime. If the current date is either later than the “notAfter” date of the certificate or earlier than the “notBefore” date it is treated as invalid and a certificate can also become “revoked” according to the CRL Profile(Housley et al., 2002, p.14). But additional certificates can be created either within the validation time range that more “valid” certificates according the PKI specification may exist at one time.

Another important aspect is that also “invalid keys” are proper input values for the factorization step. Due to Bernstein’s algorithm it turned out to be right, that if a factor in a coprime base has already been used once the other one is found in linear time whereas the calculation of prime factors by factoring it from the product directly requires exponential time.

These properties force the project to consider the lifetime of the already collected key data. Even more it must be considered how and with which methods the existing data will be managed.

This theoretical process has many influence factors and it cannot be determined in short time how to manage and to use these factors under all constraints.

Therefore only a few thoughts which have been discussed so far by this project are provided.

1. Also "invalid keys" are a valid input for the factorization.
2. The runtime for factorization increases linearly with the amount of keys, but also this time has to be considered for the complete process.[7]
3. The management of historical key data may also perturb the whole process (refer to section 9.2.1 for examples). In this situation it will be unavoidable to remove "invalid keys".

Under these conditions and from the experience in this project so far, the author proposes a similar process to that explained in section 6.4.1. The difference lies in the appreciation of the "non-accessibility" which is clear defined the Level I (an "accessible" port is active), in Level two different connection states are used. The related information is stored in the database field "status" of the table "rsa_certs" and also the field "level" is intended to manage the life cycle. Several states are possible but only the value "ok" can be treated as "accessible".

The full certificate collection process runs through several states:

1. The connection to the port itself. This step is identical to the connect process of Level I, but it uses in addition the SSL network stack and therefore it works only with SSL controlled end points. If that cannot be accomplished the service is flagged with the status value "sock" and the level is downgraded if the record already exists.
2. The connection itself does not ensure that there is a responding service behind (refer to section 8.4.4 and section 8.4.5 for possible reasons). If the connection step cannot be finished within a specific time the service is flagged with the status value "timeout" and the level is downgraded.
3. As next step a complete SSL handshake is initiated. Even if a SSL endpoint is found, it does not guarantee that any or a valid certificate is used to ensure the connection. With the handshake all data is retrieved to make it possible to validate the connection. If this step cannot be finished the flag value "handshake" is written to the status column and the level is downgraded.
4. As last step the extraction of the RSA relevant certificate data is called (refer to section 5.2.3). This step provides the necessary information for further processing and it writes the value "ok" to the status field of the corresponding service record on success and the scanning level is increased if it is beyond the configurable maximum value. If it fails for some reason the record is flagged with "certdata" and the level is downgraded (refer to appendix section §E.2.2 for a broken certificate).

In contrast to the Level I life cycle no final logic has been implemented for the Level II. In the current prototype only existing connections are handled, no "unaccessible" RSA-records are deleted and if the downgrade level "0" is reached the status of the record is set to "off". No "end" state is defined which triggers the deletion of this record.

Remark: A final life cycle logic for Level II has not been finally implemented so far. Manual care by deleting old records which do not have the "ok" status or which have the status "off" could be worth considering.

[7]The factorization runtime behavior is not in the scope of this work, but obviously finding broken factors from two "invalid keys" makes the result useless as the time for calculation is consumed but the result is not of interest for valid certificates.

Chapter 7

Implementation

This chapter describes necessities from an organizational point of view and which actions have been taken to get all necessary data which will be evaluated in chapter 8 and the detailed progress of the scan work.

In section §4.4 the technical specifications have been defined. Based on these requirements 15 providers in Austria, Germany and Switzerland have been contacted to request an offer for the following minimum requirements:

> A dedicated scalable server solution with a bandwidth of one GBit.
> The product performance should be in a medium power class.
> A possible scalability up to ten nodes should be considered.

Within two weeks seven detailed offers arrived. Only one supplier asked for technical background information for writing a customized offer. Other reactions were just references to their home pages and many did not react at all. The finally chosen hard- and software is described in section §5.3 and technical details can be found in appendix D on page 134.

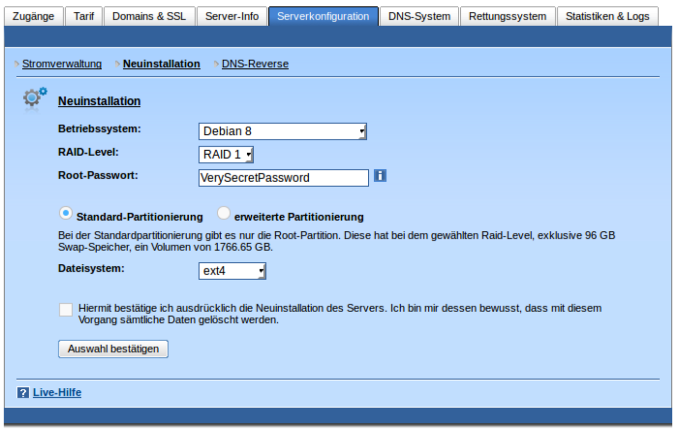
7.1 Server setup

The installation of the server was done via the Web-GUI of the provider (see figure 7.1 on the following page).[1] After a few minutes the server was installed and login via SSH was possible. A new application user (keymaster) has been created and the necessary program packages have been installed: "postgresql" (meta package for version 9.4), "tcl-8.6", "tk-8.6" and "libpgtcl" (the binary PostgreSQL database interface for TCL) by using the Debian package installer "apt-get".

```
$ apt-get install postgresql tcl-8.6 tk-8.6 libpgtcl
```

The monitoring tool "monitorix", the web server "apache2" with "php5" and "rsync" for the later described sync-mechanism has been installed in addition to allow monitoring the system from outside (web frontend) and for the service of the project's Web Information page.

[1]Unfortunately the input of the root user's password was visible in clear text inside of the user interface.



Neuinstallation

Betriebssystem: Debian 8
RAID-Level: RAID 1
Root-Passwort: VerySecretPassword

[✓ **Standard-Partitionierung**] **erweiterte Partitionierung**

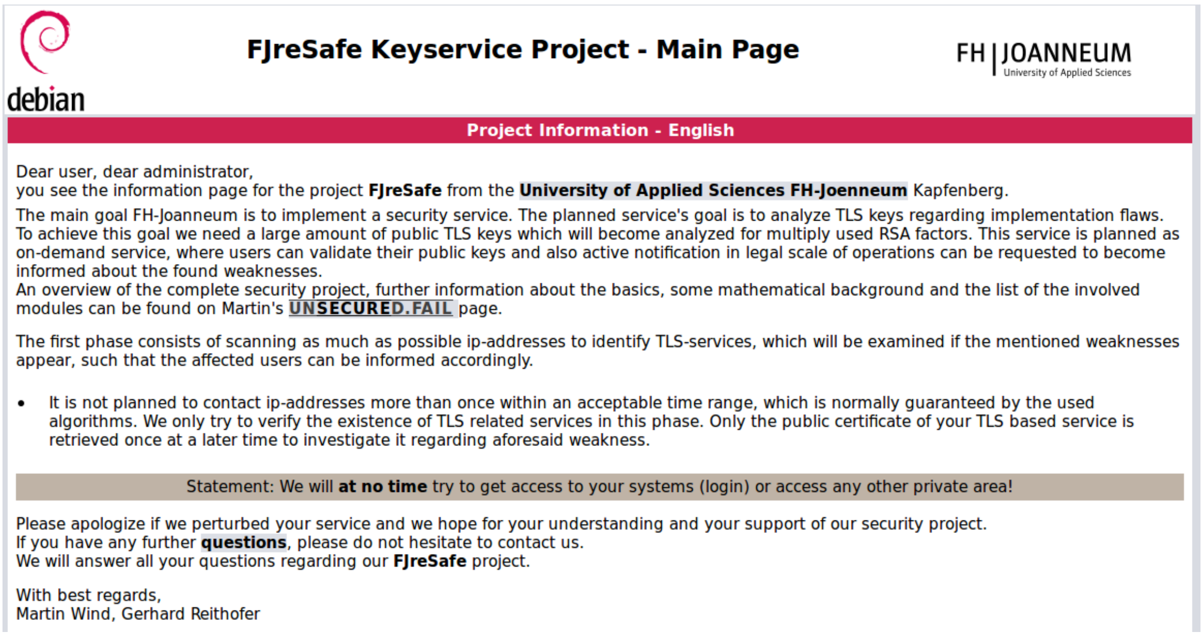
Bei der Standardpartitionierung gibt es nur die Root-Partition. Diese hat bei dem gewählten Raid-Level, exklusive 96 GB Swap-Speicher, ein Volumen von 1766.65 GB.

Dateisystem: ext4

☐ Hiermit bestätige ich ausdrücklich die Neuinstallation des Servers. Ich bin mir dessen bewusst, dass mit diesem Vorgang sämtliche Daten gelöscht werden.

Figure 7.1: Installation GUI

The Web Information page (figure 7.2) have been created after some project progress because complains of affected system administrators about supposed network attacks due to the Level I scanning occurred repeatedly (refer to section §8.4).



FJreSafe Keyservice Project - Main Page

Project Information - English

Dear user, dear administrator,
you see the information page for the project **FJreSafe** from the **University of Applied Sciences FH-Joenneum** Kapfenberg.

The main goal FH-Joanneum is to implement a security service. The planned service's goal is to analyze TLS keys regarding implementation flaws. To achieve this goal we need a large amount of public TLS keys which will become analyzed for multiply used RSA factors. This service is planned as on-demand service, where users can validate their public keys and also active notification in legal scale of operations can be requested to become informed about the found weaknesses.
An overview of the complete security project, further information about the basics, some mathematical background and the list of the involved modules can be found on Martin's **UNSECURED.FAIL** page.

The first phase consists of scanning as much as possible ip-addresses to identify TLS-services, which will be examined if the mentioned weaknesses appear, such that the affected users can be informed accordingly.

- It is not planned to contact ip-addresses more than once within an acceptable time range, which is normally guaranteed by the used algorithms. We only try to verify the existence of TLS related services in this phase. Only the public certificate of your TLS based service is retrieved once at a later time to investigate it regarding aforesaid weakness.

Statement: We will **at no time** try to get access to your systems (login) or access any other private area!

Please apologize if we perturbed your service and we hope for your understanding and your support of our security project.
If you have any further **questions**, please do not hesitate to contact us.
We will answer all your questions regarding our **FJreSafe** project.

With best regards,
Martin Wind, Gerhard Reithofer

Figure 7.2: Web Information page, English part

Together with the development of the scanning software also the Web front-end for the Copri user interface has been programmed and integrated into the Web Information page (refer to section §8.2 for details).

To ease the further installation the user keymaster was modified to allow "sudo" commands for simple system administration and SSH keys for unauthenticated access from defined machines have been created and implemented. This mechanism provides the basis to work on a foreign machine (a computer inside of the own office with all necessary development environment). In current Linux versions ICMP calls are also privileged commands and therefore some Level I scanning commands also required "sudo" for project tasks.

The final installation was done by synchronizing the main directory "keyservice" to the provider's server into the user area of the user "keymaster".

As last step the database had to be created as described in section §4.4.4.

Frequently used software management tools:

The sync-script **"utl/upsync.sh**" (see appendix A.12.4 on page 125) has been created to mirror all modified data within a set of configured subdirectories from the local computer up to the provider's server.

Two tools to copy single files or directory contents up and down without the necessity to enter a password by using SSH auto-login keys and respecting the defined directory structure, **"utl/usc.sh"** (section A.12.5) and **"utl/dsc.sh**" (section A.12.6) has been created in addition.

7.2 Level I scanning (discovering services)

As preparation for the scanning part a few configuration files had to be adapted.
The Randomized Linear Address Table (refer to section 6.2.1) with the file name defined in the configuration file **"etc/file_db.cfg"** was created (see appendix A.8.3 on page 117 for the program usage).

Immediately after this step the first scan sequence could be started. A typical output is shown here:

```
keymaster@d10162:~/keyservice$ sudo ./0.5/scan_sequ.tcl
Module masscan successfully loaded
Pre-Action: 2048 records written to input file /home/keymaster/keyservice/tmp/8194_2048.
    txt
dat/exclude.conf: excluding 123 ranges from file
Starting masscan 1.0.3 (http://bit.ly/14GZzcT) at 2015-06-23 20:38:18 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1655 hosts [19 ports/host]

Exec-Scan finished, 1 results written to /home/keymaster/keyservice/tmp/8194_2048.csv
Post-Action: 51 results found
[8194_2048]: 2048 IPs scanned in 00:00:14 (146.3/sec), 51 TLS related IPs (2.49% - 3.64/
    sec), ssl=1 open=49 X509=1
```

Documentation of the output of this program in detail:

- The start offset for the Linear Address Table was **8194**, the value is read from the sequence file - corresponding to the command line option "-s 8194".
- The address block size was **2048**, defined in the configuration file **"scan_db.cfg"** with the parameter name NUM_RECS (see appendix A.8.4 on page 118 for an example) - corresponding to the command line option "-n 2048".
- The temporary file **"8194_2048.txt"** contains the masscan input list and in the file **"8194_-2048.csv"** the masscan result is stored. Both files are located in the temporary subdirectory which is fixed to the subdiretory **"tmp"** in the installation directory.

- The used scanning module was **"masscan"** (refer to appendix A.6 on page 114). The default command line parameters are shown, the name of the exclude list and the amount of excluded ranges, the number of specified ports (also defined in "scan_db.cfg") and the final size of cleaned addresses are shown.
- The post action detected 51 results, i. e. 51 possibly TLS related services were found. Statistical runtime and quantity information are shown in the last output line.

Refer also to subsection 6.2.5.1 on page 45 where this process has been analyzed in detail.

7.2.1 Scan rate and performance

As described in section §6.3 the optimal parameters had to be found for the parallel scan performance. A balance between the number of processes and the system load for the existing environment was expected and it could also happen that some limitation of one component may overload and output will become decreased.

Within the next few tests a component limit was detected because the number of open database connection connections exceeded the configured limits and the following runtime errors occurred:

```
Database error: Connection to database failed
FATAL:  sorry, too many clients already
```

It was necessary to change the server parameter "max_connections" from 100 to 2000 in the file "/etc/-postgresql/9.4/main/postgresql.conf".

Knowing that changing of one single configuration value often do not reach general optimization success or sometimes it even may gain negative effects in other situations, a web based optimization tool could be found which supports this optimization. The source code of this freeware tool is available on GitHub (https://github.com/le0pard/pgtune). The database server was tuned by providing the actual data and selecting the application type "Mixed type of application".

The recommended parameters were applied to the server configuration file and the database was restarted:

```
max_connections = 2000
shared_buffers = 12GB
effective_cache_size = 36GB
work_mem = 3145kB
maintenance_work_mem = 2GB
checkpoint_segments = 32
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
```

7.2.2 Scanning process tuning

In section §6.3 methods for scaling the scan process by starting more processes in parallel have been described. In this section the optimization steps for the different loading conditions in the specific hardware configuration will be analyzed.

The graphic performance measurement tool XOSView (http://xosview.sourceforge.net/) shows various system states in real time. It has been used to monitor the current system's state. In case of the scan work the load of the 16 CPU cores is of interest to judge the allocation of the single cores on parallel tasks. As a typical X-Application this tool can be widely configured via an X-Resources configuration file (refer to the documentation in http://xosview.sourceforge.net/documentation.html#X%20RESOURCES).

In figure 7.3 a clipped part of XOSView window demonstrates the system load and the load of the first five CPU cores.[2]

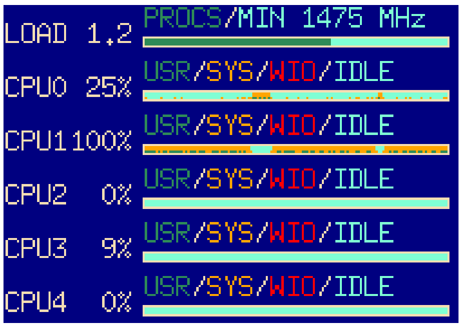


Figure 7.3: Single CPU load - clipped image

CPU1 is fully utilized for ca. 1/3 of the width of the observed window. Then the CPU is released and the process allocates CPU0 for a short time (less than 10% of the bar width) and after this period CPU1 is again (almost exclusively) used until the end of the observation period which covers a time of ca. twelve seconds.

In the used XOSView display settings different hardware states per CPU are shown. These are the User load (“USR”, non-privileged processes in green color), the System load (“SYS”, privileged OS processes in orange), the Wait state (“WIO”, if a process waits for a resource in red) and the CPU Idle state (in light blue color) when it waits for assignment of new tasks. The horizontal bar is actually a histogram view with stacked values for the two CPU states “USR” and “SYS”. The 100% usage of CPU1 is the sum of user and system load which encompasses the complete bar height in this case.

The small green part of the bar for CPU1 reflects a small user load (very small user process overhead) and the yellow part above the green one refers to the operating system’s created load, which is in this case mostly the network IO handling. The system load (“LOAD”, the top most bar) is with 1.2 rather small, it is the average number of processes waiting for allocation resources (e. g. waiting for disk or network IO) within a defined time period.

7.2.2.1 System load examples

Lets take a look at four different and symptomatically load situations in figure 7.4 on the following page. These have been monitored using the Timed Loop mode, described in section section 6.3.1 and in the Controlled Loop mode described in section 6.3.2.

1. The most left graphic shows the situation with a job wait time of four seconds, i. e. every four seconds a new job is started with a chosen block size of 1.000 addresses. This leads to an average scan rate of 250 addresses per second. The bars show a significant user load on ca. five CPUs with currently four cores at 100% and almost no system load. The interpretation of this information is that mainly the management scripts created the load by starting the interpreter, reading the configuration, preparing and calling the scan module and interpreting the output afterwards. No notable system load has been detected as it would be if the network processes would become stressed. The low network use (172 Kb/sec, the lowermost indicator) validates this because only the remote access and the transmission of the screen of the graphic tool needs ca. 100Kb/sec. Also the process

[2]This screen shot has been made when the masscan process was been tested as single scan process.

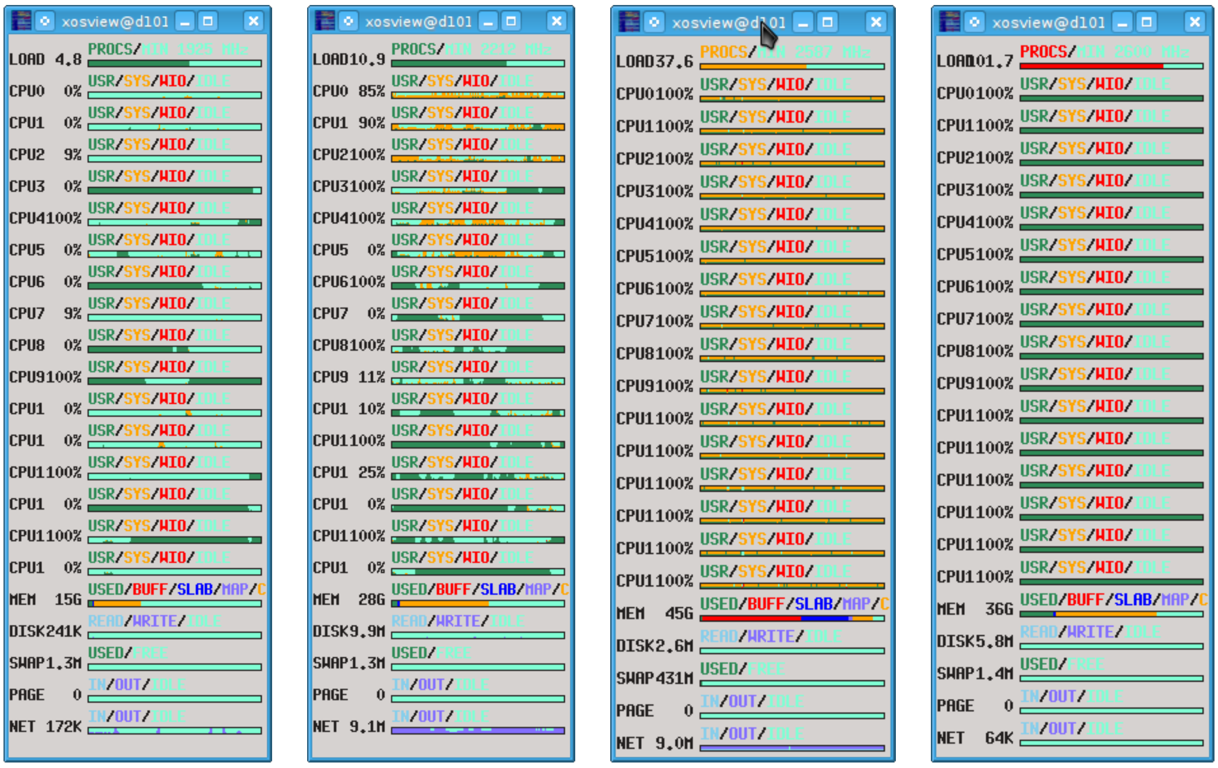


Figure 7.4: Parallel processing load

monitoring confirmed an amount ca. four permanent scan processes and also the load average of 4.0.

2. In the next graphic a clearly increased user load is visible, ca. 1/2 of the CPUs are fully utilized (7 times 100%) and five of them show mainly a higher system load (i. e. network activities in yellow color). The process monitoring confirmed a mean value of seven parallel scan processes. The network throughput of 9.1 Mb/sec. comes near to the predetermined limit of 100 MBit/sec (refer to section 4.1.1). The block size was in this case 10.000 addresses with the same wait time as before, this gives a mean scan rate of 2500 processes per second if balanced.

3. The Timed Loop mode has much uncertainties as it relies on a relatively constant time for scanning a single block. In some situations the system load was significantly increased and the danger of a system overload could not be eliminated. Also later investigations confirmed that the scan time for a single 100.000 address block varied from a low value of 00:00:40 up to the highest value of 00:04:44 which is a time factor of 7.1. This induced the idea to limit the number of processes using the “Controlled Loop” mode (refer to section 6.3.2). Within this setup the system load could be limited to a meaningful value - see the third graphic with a system load of 37.8 created by a wait period of 0.7 seconds.

4. The scan rate cannot be increased without limits, because too much system load decreases the effective throughput and if the system is too long used near the limits, resources may run out and the output becomes zero (right most image). In the rightmost graphic such a situation was reached and the scan process did not deliver anymore useful results. The user load of all processes was

100%, the load average was larger that 101.7 and the network shows only a transfer rate of 64 Kb/sec which can be qualified as “nothing”.

Remark: Especially on multi core systems a wait state of one core does not necessary thwart another core, as long as it doesn’t wait for the same resource. This topic is also discussed in section §9.2.

7.2.3 The process view and system load

The measurement until now only provided mainly chart overviews. Looking at the single CPU core and the memory usage per process gives information how utilized the machine is on process level. The Linux standard command “top” is a powerful tool to analyze the system load on process level and the utilization on CPU core level.

```
top - 12:51:38 up 10:34,  2 users,  load average: 12,13, 13,30, 13,32
Tasks: 267 total,   9 running, 258 sleeping,   0 stopped,   0 zombie
%Cpu(s): 40,2 us, 19,9 sy,  0,0 ni, 36,5 id,  0,6 wa,  0,0 hi,  2,8 si,  0,0 st
KiB Mem:  49561972 total, 26309436 used, 23252536 free,    94336 buffers
KiB Swap: 50331624 total,        0 used, 50331624 free. 24012900 cached Mem
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2042 root      20   0  340316 179164   4292 S 118,7  0,4   0:37.56 masscan64
 2098 root      20   0  340316 179156   4292 S 118,7  0,4   0:30.74 masscan64
 2087 root      20   0  340308 179156   4292 S 112,5  0,4   0:31.35 masscan64
 2121 root      20   0   17792   2860   1992 R 100,0  0,0   0:23.13 masscan64
 2128 root      20   0   17788   2824   1964 R 100,0  0,0   0:19.37 masscan64
 2156 root      20   0   17792   2860   1996 R 100,0  0,0   0:11.54 masscan64
 2165 root      20   0   17280   2600   1992 R 100,0  0,0   0:07.68 masscan64
 2182 root      20   0   17276   2604   1996 R 100,0  0,0   0:03.55 masscan64
 2191 root      20   0   52420  19776   3500 R 100,0  0,0   0:00.80 tclsh
 2138 root      20   0   17788   2868   1996 R  93,7  0,0   0:15.49 masscan64
 2035 root      20   0  340312 179148   4292 S  37,5  0,4   0:40.51 masscan64
 2049 root      20   0  340316 179152   4292 S  25,0  0,4   0:37.90 masscan64
 2108 root      20   0   20192   3708   2180 R  25,0  0,0   0:24.98 masscan64
    3 root      20   0       0      0      0 S  12,5  0,0  75:08.77 ksoftirqd/0
 2194 keymast+  20   0   23780   3012   2448 R   6,2  0,0   0:00.01 top
13399 keymast+  20   0   35212   4648   4092 S   6,2  0,0   0:38.22 xosview
    1 root      20   0   28768   4996   3112 S   0,0  0,0   0:02.07 systemd
```

This output shows that twelve masscan processes running at the same time and three of them show a CPU usage of $> 100\%$ which means that more than one core is involved by the process. seven processes have a CPU core usage of 100% or near this value which means that the CPU is full utilized.

The load average of 12-13 was still relatively low but went up to values near 40 (i. e. the system has 40 waiting processes) as figure 7.5 on the next page demonstrates.

The tool “top” also provides the CPU states as numeric values which already have been described in section 7.2.2. A typical load situation snapshot with 60 parallel processes and a system load near 40 can be seen here with a very good load distribution among the CPU cores.

```
top - 21:39:18 up 27 days,  2:57,  2 users,  load average: 36,72, 37,30, 37,63
Tasks: 300 total,  38 running, 262 sleeping,   0 stopped,   0 zombie
%Cpu0  : 45,4 us, 53,6 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  1,0 si,  0,0 st
%Cpu1  : 41,6 us, 58,4 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu2  : 41,1 us, 57,9 sy,  0,0 ni,  1,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu3  : 45,2 us, 53,8 sy,  0,0 ni,  0,7 id,  0,0 wa,  0,0 hi,  0,3 si,  0,0 st
%Cpu4  : 43,8 us, 55,6 sy,  0,0 ni,  0,7 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu5  : 44,1 us, 55,9 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu6  : 42,6 us, 54,5 sy,  0,0 ni,  2,6 id,  0,0 wa,  0,0 hi,  0,3 si,  0,0 st
%Cpu7  : 44,6 us, 55,1 sy,  0,0 ni,  0,3 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
```

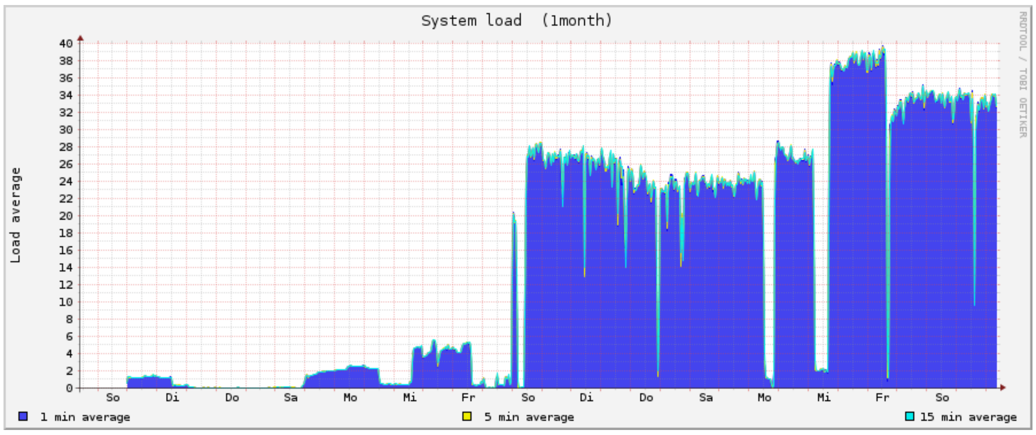


Figure 7.5: Very high system load situation

```
%Cpu8  : 43,0 us, 57,0 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu9  : 41,9 us, 58,1 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu10 : 43,9 us, 55,8 sy,  0,0 ni,  0,0 id,  0,3 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu11 : 40,5 us, 59,2 sy,  0,0 ni,  0,3 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu12 : 42,8 us, 57,2 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu13 : 57,6 us, 42,4 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu14 : 44,1 us, 55,9 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu15 : 43,9 us, 56,1 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
```

7.2.4 Scan Level I performance

The evaluation of a time range of 1-2 days in a relatively optimized state of Phase 1 of the project can be seen in table 7.1. The reached scan rate (IPs/sec) are class *A* addresses according section §6.4, the TLS/sec are class *S* addresses according section §6.5.

The first scan was for port 443 only and the second one scanned the nine ports 22, 465, 636, 989, 993, 994, 4031, 8140 and 8443.

case #	port 443 only #1	9 ports (not 443) #2	#2/#1	Remark
Scan start	2015-08-21 10:08:22	2015-08-22 13:03:55		
Scan end	2015-08-22 12:41:18	2015-08-24 13:19:22		
Blocks	47236	36938	0.782	
IPs tot.	4059017295	3270700000	0.806	
TLS rel.	3575192	22936589	6.415	Probably TLS
Scan time	1d 02:32:56	2d 00:15:27	1.811	
IPs/sec	42469.0	18826.7	0.443	Class A addresses
TLS/sec	37.4	132.0	3.529	Class S addresses
TLS/1M	880	7010	7.966	1M=1 million IPs
TLS/1M/port	880	778.8	0.885	Specific Rate "R"

Table 7.1: Scan rate Level I

By comparing both cases it becomes obvious that multi port scanning has a clearly a higher output of possible TLS related services (TLS/sec) because more ports become scanned per IP address. It was a mix of different expected services which are known that they use also TLS (except ssh[3]).

But taking into account the number of IPs and the number of services and the TLS hits (TLS/1M/port) the values do not differ much. This value is called the "Specific Rate" for class *A* addresses.

Also one more criterion must be considered, i. e. the amount of TLS approved services vs. the open ports without TLS services.

This aspect is analyzed in section 7.3.4.

7.2.4.1 Single port scan

The single port scan (figure 7.6) shows a very steady and continuous behavior in relation to the found TLS ports. This was expected as most services on the HTTPS ports are intended services which provide a HTTPS service.

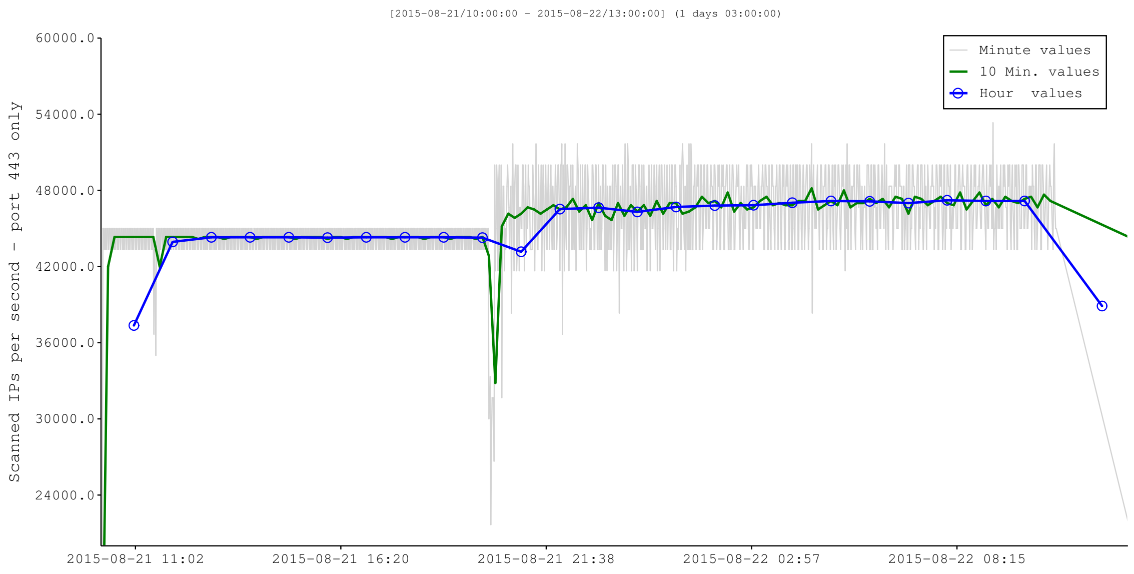


Figure 7.6: Scan rate Level I - port 443 only

An interesting effect can be seen looking at the chart before and after 2015-08-21 20:15.

In the first part of this graphic oscillates most time between two values of collected services. The evaluation using **"stats_parscan.sh"** (refer to appendix on page 150 for program usage) between 10:00 and 20:00 shows an average scan rate of **44137.1**.

Then the timing value JOB_WAIT_TIME of the "Timed Loop Mode" (see section 6.3.1) was reduced from 0.5 to 0.3. The average scan rate (2015-08-21 20:00 until 2015-08-22 10:00) went up to **46884.8** but also the variance of the scan rate profile became increased over proportional.

This is an evidence for increased crosstalk of the simultaneously running processes. By enhancing this trend the variance would become increased until the overload situation described in section 7.2.2 would appear.

[3]The SSH service was added as typical non-TLS service by intent to check if another TLS related services maybe hidden behind this port.

7.2.4.2 Multi port scan

The multi port scan (figure 7.7) shows a complete different scan rate profile.

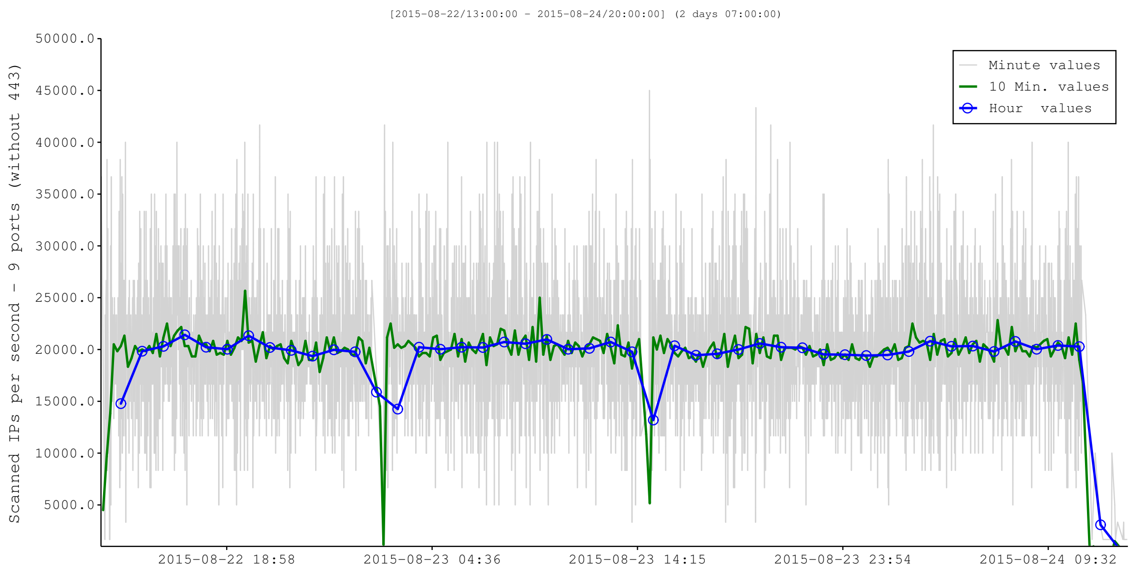


Figure 7.7: Scan rate Level I - multi ports

The constant scan rate from the single port scan does not appear, mainly because many ports either do not answer in a correct way and many connections became timed out. The result is a noticeable different runtime for the single scan processes. Depending on the timer values this situation leads to a lock situation when the process load becomes too high. This is more likely here than in the single port scan mode before.

The "clean average" scan rate was **19655.5** addresses per second, the values in table 7.1 represent the exact "raw" value, i. e. the mean value is calucated over the complete process.[4]

The interaction between the scan performance and the providers feedback will be discussed in section 8.4.6.

7.2.5 Phase 2 of Level I scanning

The project was executed in two parts (refer also to section §6.1) and the second part of Level I becomes investigated now.

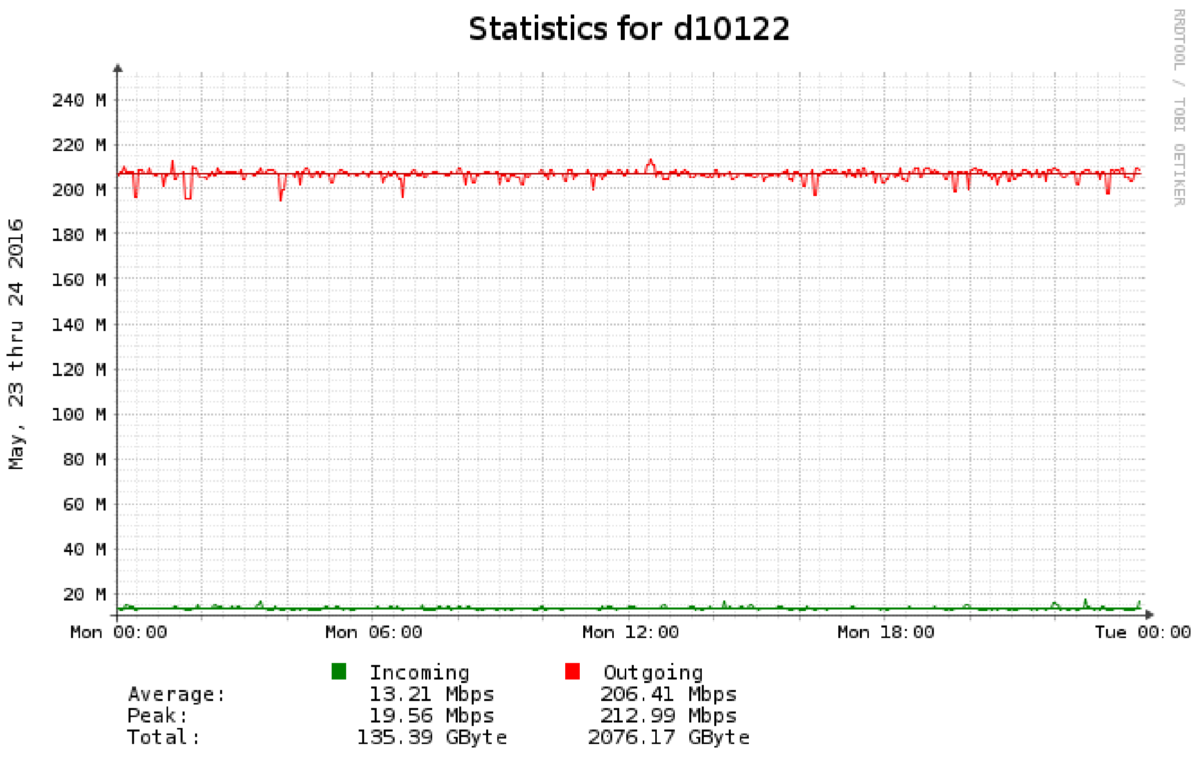
The main difference between the phases was that in this part the scan was executed as plain scanner program call without the described block based access method (refer to section 6.2.4 and section §6.3). To become a more clear idea about the port distribution a larger amount of ports to scan have been selected, the port list can be seen in appendix C.2 on page 131.

The last step (read the output files and write the results to the database) had to be executed nevertheless, because the scanner only delivers output files which must be read and processed in the mode as it has been scheduled for the wrapper script **"sequ_scan.tcl"**. The output was a large single text file containing more than 100 million result records (see section 7.2.6).

[4]"Clean average" means cutting off the startup value and outflow phase. This was done actually by adapting the start and at the end time of the evaluated time range.

The used masscan parameters are shown in the configuration file **"etc/masscan.cfg"** in appendix A.8.2 on page 117 and the starting program called **"scr/basescan_all.sh"** in appendix A.10.4.1 on page 123. Due to the command line output of masscan 32 address ranges have been excluded and **3.970.008.572** hosts (class A_1 addresses) with **19** ports per IP had to be scanned (refer to appendix A.5.1 on page 114).

The bandwidth limitation became clearly evident. Investigating the output and monitoring data proved that the scan rate of this method was solely limited by the network bandwidth. In figure 7.8 the 200 MBit bandwidth limit of the provider is obvious.



```
              🟩 Incoming       🟥 Outgoing
Average:         13.21 Mbps       206.41 Mbps
Peak:            19.56 Mbps       212.99 Mbps
Total:          135.39 GByte     2076.17 GByte
```

Figure 7.8: Full seed scanning Level I

Unfortunately there are no results how many IP addresses have been scanned because after ca. 35 hours the experiment has been stopped. The cancel criterion was that 100 million result records had been reached, mainly class S addresses according section §6.4.

The final saved IP addresses were **92.797.617** which could be taken as input for the calculation of the used address space according table 7.1 on page 66, but as the environment was rather different no ensured results can be expected.

Remark: The theoretically bandwidth limit due to the provider contract in Phase 2 and the technical possibilities was 500 MBit (compared to 100 MBit in Phase 1). But the provider lowered the allowed value because of the high amount of complains of network responsible persons. This is also one factor to be considered when selecting a provider and the network infrastructure for the project.

The temporary output file delivered result records in the following format:

```
open tcp 8443 104.16.75.86 1463952428
```

The five component fields of this line are:

1. The state of the connection (only “open” is used for scanning and “banner” for statistics only)
2. The protocol, fixed to “tcp” (refer to section §2.3)
3. The connection port number of the service
4. The IP address where the service is located
5. The Unix timestamp (i. e. seconds since 1970-01-01 00:00:00)

The timestamp information can be used to calculate the scan speed result in services per second.

7.2.6 Hi speed scan import

The created masscan result list had to be imported. This was a specific situation which cannot be compared to the steps before as the temporary result output file of the high speed scanning was one single file with a final size of 4.4GB with more than 100 million lines.

```
$ ls -lh tmp/basescan_all.out
-rw-r--r-- 1 gerhard users 4.4G May 24 11:29 tmp/basescan_all.out
$ wc tmp/basescan_all.out
 100027755  510113202 4636356608 tmp/basescan_all.out
$ grep -v "^open" tmp/basescan_all.out | wc -l
2959060
$ echo 100027755-2959060|bc
97068695
$ echo "scale=5;97068695/100027755"|bc
.97041
```

The exact open port count was **97.068.695** because there have been written additional lines to the output file, mainly “banner” entries which should not appear with the used options. The resulting error is ca. **3%**.
For parallelizing reasons the big output file was split into single files with each one containing a maximum of one million lines using the Unix standard command “split”:

```
split --lines 1000000 --numeric-suffixes  --suffix-length=3 \
      --additional-suffix=.csv basescan_all.out basescan_all_
$ ls -l tmp/basescan_all*.csv
-rw-r--r-- 1 gerhard users 45795896 Mai 25 00:56 tmp/basescan_all_000.csv
-rw-r--r-- 1 gerhard users 45652295 Mai 25 00:56 tmp/basescan_all_001.csv
-rw-r--r-- 1 gerhard users 46946136 Mai 25 00:56 tmp/basescan_all_002.csv
...
-rw-r--r-- 1 gerhard users 45414386 Mai 25 00:57 tmp/basescan_all_099.csv
-rw-r--r-- 1 gerhard users  1316204 Mai 25 00:57 tmp/basescan_all_100.csv
```

To importing temporary result files the new import script **“save_scan.tcl”** has been programmed which contains the logic for importing a single temporary output file (refer to appendix A.1.6 on page 108 for the usage message). Furthermore an additional process management script **“scr/par_import.sh”** has been created which manages the parallel processing in Controlled Loop mode and the monitor program **“scr/monitor_base.sh”** which monitors the increase of found IP addresses and calculates the services per second.

7.2.7 Effective scan rate

The evaluated values from table 7.1 on page 66 reflect the unfiltered input of addresses of class A (refer to section §6.4) from the Randomized Linear Address Table (see section 6.2.1) displayed in the rows "IPs tot." and "IPs/sec".

Because of the lack of time dependent information the addresses of class A or (A_1 which is with 92.43% a little bit smaller due to address filtering) cannot be evaluated in this situation. Only class S data (possibly TLS related open ports) can be evaluated which differs significantly depending on the size of the multiplier I_1 and this is the reason why the value "TLS/sec" is remarkable higher on the multi-port scans. In this step 19 ports have been used (refer to appendix C.2 on page 131 for the list of scanned ports).

The three curves in figure 7.9 are the gray "Scan rate", i. e. the evaluated values from the database, "Mov_Avg_60" is the green smoothed curve calculated as moving average over 60 values and the "Trend line" in blue.

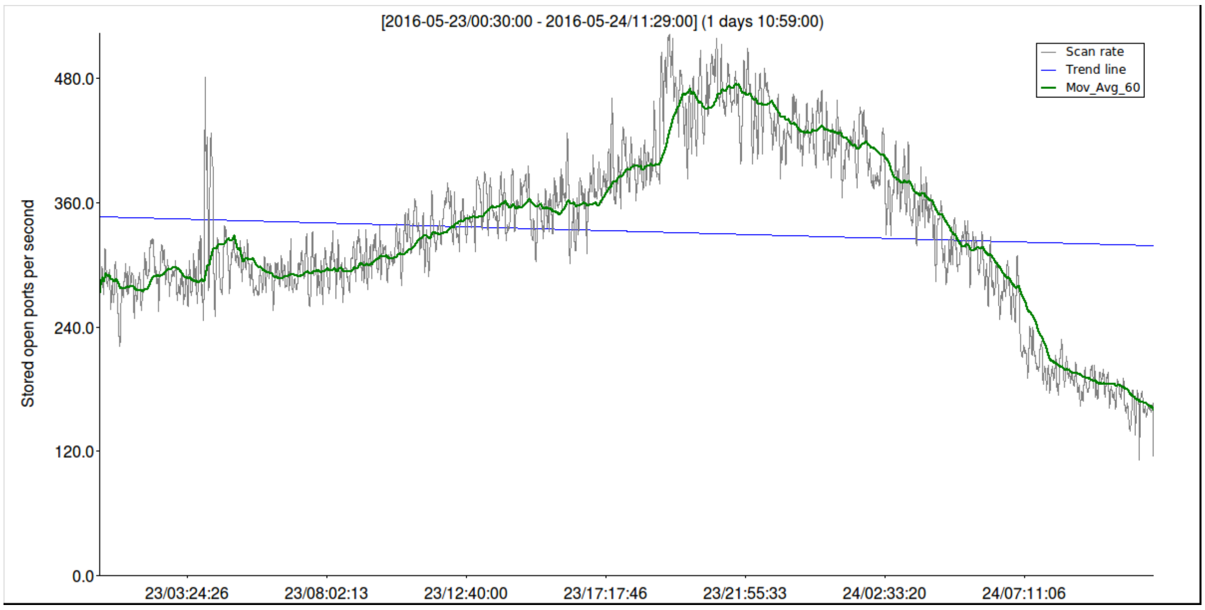


Figure 7.9: Effective scan rate

The stored open ports trend line starts with 345 services per second and it decreases slightly until to a value of ca. 320 services per second. The exact curve varies in a larger range but the downward trend is obvious. This is an expected behavior because the system stress is increased due to a growing database, file system, etc. and one reason may also be caused by the daily fluctuation of the providers load.

This information is stored in the database and has been created by querying the database. Therefore it is the "Effective scan rate".

This result can be retrieved by using the database evaluation tool using the following command:

```
$ utl/select2CSV.sh -o sta/data/seq_mins_group_save.csv "select
  to_char(seen_at,'YYYY-MM-DD HH24:MI')||':00',count(*)/60.0
  from scan where seen_at<='2016-05-24 11:30:00'
  group by to_char(seen_at,"'YYYY-MM-DD HH24:MI')
  order by to_char(seen_at,'YYYY-MM-DD HH24:MI')"
```

Remark: **select2CSV.sh** is a script which allows the execution of SQL commands and the results can be written to a CSV file, refer to appendix A.12.3 on page 124 for the command line syntax.

7.2.8 Port number distribution

The Phase 2 was a multi port scan. Therefore it is a good opportunity to analyze the distribution of the ports,figure 7.10 shows the result:

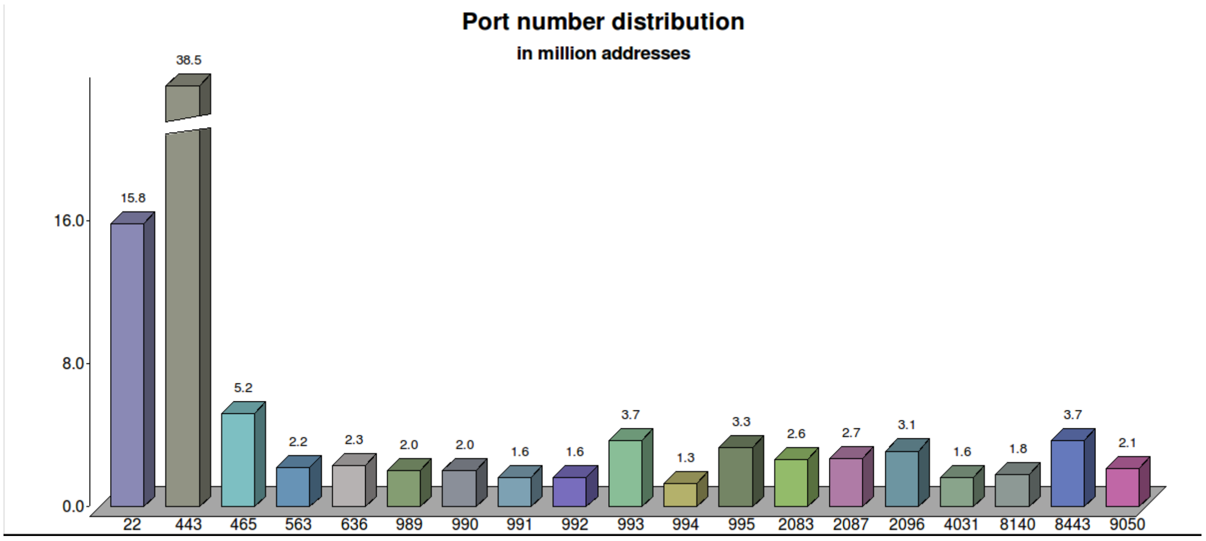


Figure 7.10: Distribution of the service ports

Remarks: The collected set of addresses contains 97.068.695 entries (i. e. ca. 100M) therefore the values in the graphic corresponds roughly with the percentage values. The highest bar (port 443) is displayed in reduced size. Its original height is more than twice the value of the second largest value for port 22.

The top three port ranking including SSH (97.068.695 addresses)

Port 443	38529358	39.7%	HTTPS most used TLS service
Port 22	15753958	16.2%	usually not a TLS service
Port 465	5223677	5.4%	
other 16	42785379	31.1%	max. 3.4%, min. 1.3%

When removing the port 22 which can be treated as “experimental” entry in sense of checking for a possible TLS service behind the SSH port, the drop of relevant TLS ports is significant and the HTTPS port percentage becomes increased up to 47.4%.

The top three port ranking excluding SSH (81.314.737 addresses)

Port 443	38529358	47.4%	HTTPS most used TLS service
Port 465	5223677	6.4%	
Port 8443	3669432	4.5%	
other 15	33892270	41.7%	max. 4.5%, min. 1.6%

7.3 Level II scanning (collecting certificates)

This scanning mode shows a complete different picture compared to the Level I. As in Phase 1 of the project Level I mainly was limited to the number of processes because the network transfer per scan entity is almost negligible (ca. 40 Byte per access). Here the database load becomes relatively high because of many read/write operations for the data management[5]. Level II produces a noticeable amount of traffic and the tested average value was ca. 3.6 Kb per certificate (refer to the section 4.1.1). On the other side the encryption and decryption procedures are very CPU intensive activities because they require very time consuming calculations with very large integer numbers.

Therefore it can be expected that the main limitation will be the network throughput on one and the CPU usage but on the other side but also high loads will be caused by database activities.

7.3.1 The Level II scan process

According to figure 2.1 on page 15 the certificate collection part is based on the following steps:

1. Read IP/Service range from table “scan”.
2. Trying to collect certificates for all service records.
3. On success insert/update table “rsa_certs” and save it as file (refer to section §2.6).
4. Update “status” field in table “scan” according to the results in step 2. and 3.

7.3.1.1 Preparing the table “scan”

One important point for the planned service is the intensive access of the port scan of Level I. This leads always to feedback from network providers and administrators which usually interpret this step as malicious access.

It is necessary to react on feedback of the network responsible persons as they signal to block access from the project’s IP addresses.

The project’s reaction was that the corresponding IP addresses and ranges became black listed if they do not want to become scanned anymore and they provide the address range information. This was done by adding the information to a file which contains addresses and network ranges (the “exclude list”) which should be ignored and their addresses won’t be accessed anymore.

At this time already scans have been made and corresponding addresses are most probaly already present in the table “scan”. To take this into account the scan table must be manipulated in a way that all addresses from the exclude list become marked as “blocked”. The small shell script **“util/mark_blocked.sh”** (refer to appendix A.12.2 on page 124) has been written which reads the file **“etc/exclude.conf”** (refer to appendix A.8.1 on page 115) and creates the corresponding SQL statements.

[5]This was also one of the main reasons why the Level I input storage was organized as flat file because the throughput is much higher - refer to section 6.2.1.

A reduced typical output may look like:

```
$ utl/mark_blocked.sh
--
-- Pipe the following command into PSQL-client
--
update scan set status='blocked' where ip_address << '0.0.0.0/8';
update scan set status='blocked' where ip_address << '10.0.0.0/8';
update scan set status='blocked' where ip_address << '100.64.0.0/10';
 ...
update scan set status='blocked' where ip_address << '66.148.70.0/18';
update scan set status='blocked' where ip_address  = '203.9.184.0';
update scan set status='blocked' where ip_address  = '131.242.0.0';
update scan set status='blocked' where ip_address << '144.39.0.0/16';
```

The command output can be used directly as pipe input for the PostgreSQL database client:

```
$ utl/mark_blocked.sh | psql -d keyservice -U keymaster -w
UPDATE 0
UPDATE 0
UPDATE 2119
 ...
UPDATE 2443
UPDATE 0
UPDATE 409
UPDATE 415
UPDATE 0
```

This topic will be covered in more detail in section §8.4.

7.3.2 Performance issues

The steps 1., 3. and 4. from section 7.3.1 are "only" database actions and as time consuming part step 2. was supposed which was also confirmed after the first tests. With an increasing number of address entries the time behavior of this step became increased lucidly. The reason was that many database access steps were called and inefficient access due to a bad index management on the table "scan". The import time for a single certificate went up to more than 20 seconds and after creation of new indexes the time was reduced to less than one second.

In that reorganization phase another database limitation appeared. PostgreSQL does not allow to create indexes on too large rows, the following error appeared when trying to index the byte array field "rsa_n" which is vital to do a certificate lookup for a specified RSA n-value (see also section §8.2):

```
keyservice=# create index on rsa_certs(rsa_n);
ERROR:  index row size 4112 exceeds maximum 2712 for index "rsa_certs_rsa_n_idx"
TIP:  Values larger than 1/3 of a buffer page cannot be indexed.
Consider a function index of an MD5 hash of the value, or use full text indexing.
```

As alternate index creation command in this situation a hash index was necessary and all concerned queries had do be adapted:

```
create index rsa_n_md5_index on rsa_certs(md5(rsa_n));
```

7.3.3 Scan Level II tuning

The techniques explained in section 7.2.2 can also be applied to the Level II scanning. All results are directly stored in the database and no "external" tools are used within this scanning step. Therefore the database can be used as main source for evaluations but some information (e. g. the number of parallel processes) can be retrieved only from log files.

The complete functionality is covered by the script **"retrieve_cert.tcl"** (refer to appendix A.1.4 on page 107). No other external program like the executable scanning module in Level I is involved.

Remark: In contrast to the Level I scanning in this section we will concentrate on the second project phase because this process is identically with Phase 1 and this is the final software state and most optimized part. If "older" information is referred it will be stated explicitly.

The tasks view shows a very constant task load caused by the Controlled Loop Method described in section 6.3.2.

According to figure 7.11 the task value begins with a maximum of 36 allowed parallel tasks at 2016-06-04 18:34, after 15.5 hours the number of tasks became increased to 50 and this configuration was running for eight days and 17 hours. After a pause of 8.5 hours the last phase was scanned with a task limit of 60 parallel tasks for ca. one day and 9 hours at 2016-06-15 14:53. The complete process ran ca. three days taking the various pauses into account.

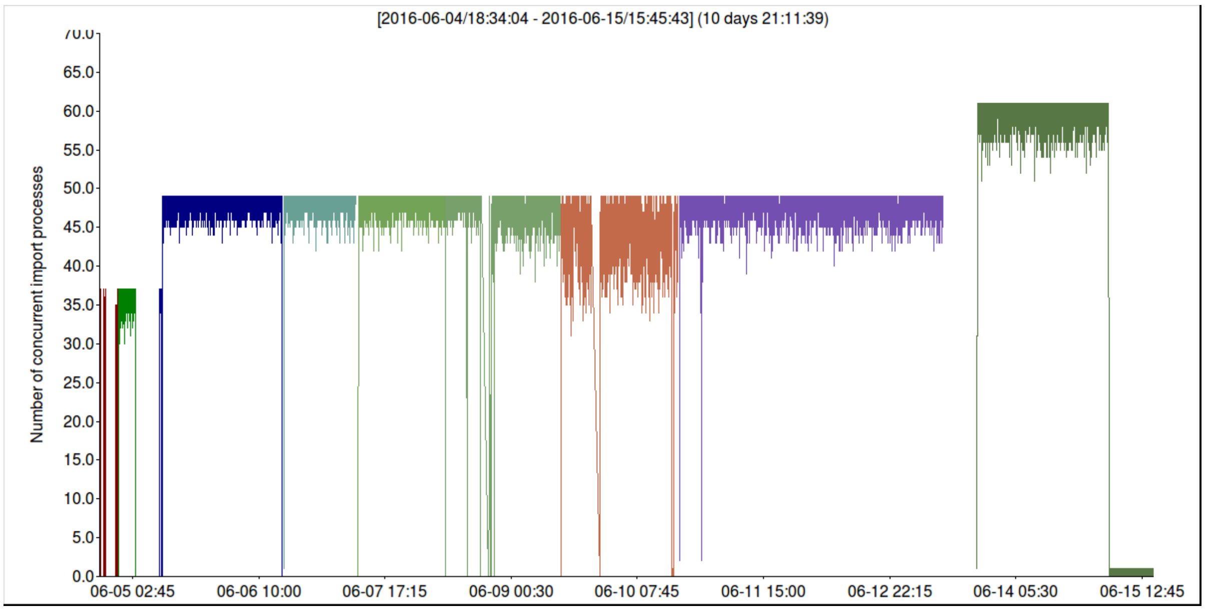


Figure 7.11: Parallel retrieve certificate tasks

Remark: In this scanning mode the runtime parameters can be changed at any time without stopping the whole process. This is simply done by changing the values (like the variable $JOB_LIMIT) in the Controlled Loop mode script (in this case "scr/par_scan_rsa.sh"). The values will then be used in all further scan blocks (refer to section 6.3.2 and section 6.3.3).

The “Effective scan rate” of this step which is retrieved from the database using SQL statements as described in section 6.2.5.2 will be analyzed now. For figure 7.12 the scan rate over the same time range as in figure 7.11 is evaluated grouped in ten-minute blocks and visualized as line chart.

the data has been calculated by using the following command:

```
$ utl/select2CSV.sh -o sta/data/rsa_mi10_group_cert.csv "select
  substring(to_char(last_mod,'YYYY-MM-DD HH24:MI') from 1 for 15)||'0:00',count(*)/600.0
  from rsa_certs
  where last_mod>='2016-06-04 18:00:00' and last_mod<='2016-06-15 04:00:00'
  group by substring(to_char(last_mod,'YYYY-MM-DD HH24:MI') from 1 for 15)
  order by substring(to_char(last_mod,'YYYY-MM-DD HH24:MI') from 1 for 15)"
```

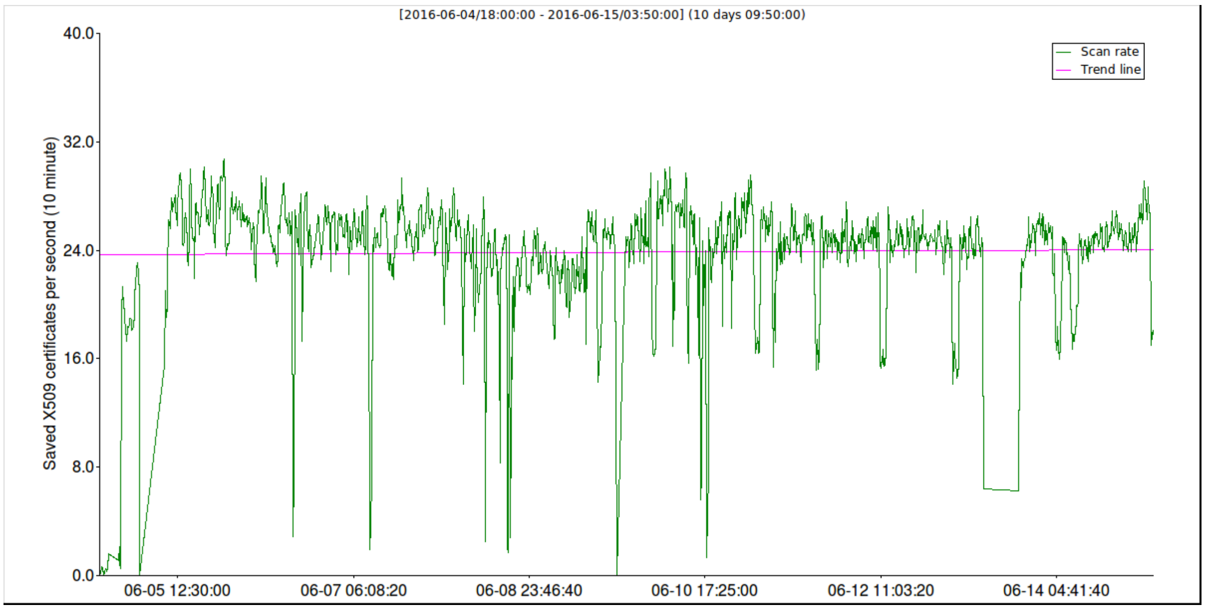


Figure 7.12: RSA certificate collection rate

An interesting finding is that the first change of the number of concurrent processes is clearly visible (from 36 parallel processes to 50 at ca. 2016-06-05 10:30) wheres the the second increase (from 50 to 60 at ca. 2016-06-13 20:00) near the end of the time scale does not increase the scan rate noticeable.

A similar behavior has already been realized in other measurements of another environment like the “last-hour” rate evaluation with a JOB_LIMIT of 32 and 64 processes.

32 parallel tasks

```
keyservice=# select count(*) from rsa_certs where last_mod>now()-interval '1 hour';
 116814
```

64 parallel tasks

```
keyservice=# select count(*) from rsa_certs where last_mod>now()-interval '1 hour';
 119029
```

The increase of the task limit by 100% (from 32 to 64) increased the effective scan rate only by ca. 2% (from **32.5** to **33.1** certificates per second) which confirms this effect.

7.3.4 Comparison of open ports and retrieved certificates

The values used in section 7.2.8 have been extracted from the Level I scan process which just tests if a port is “reachable” (see also section 6.4.1). If a service finally contains an active TLS service is another interesting aspect.

As the Level II scan feeds the result back to the Level I data into the “status” field it allows the comparison of the “open” non-TLS ports with the “found” TLS certificates.

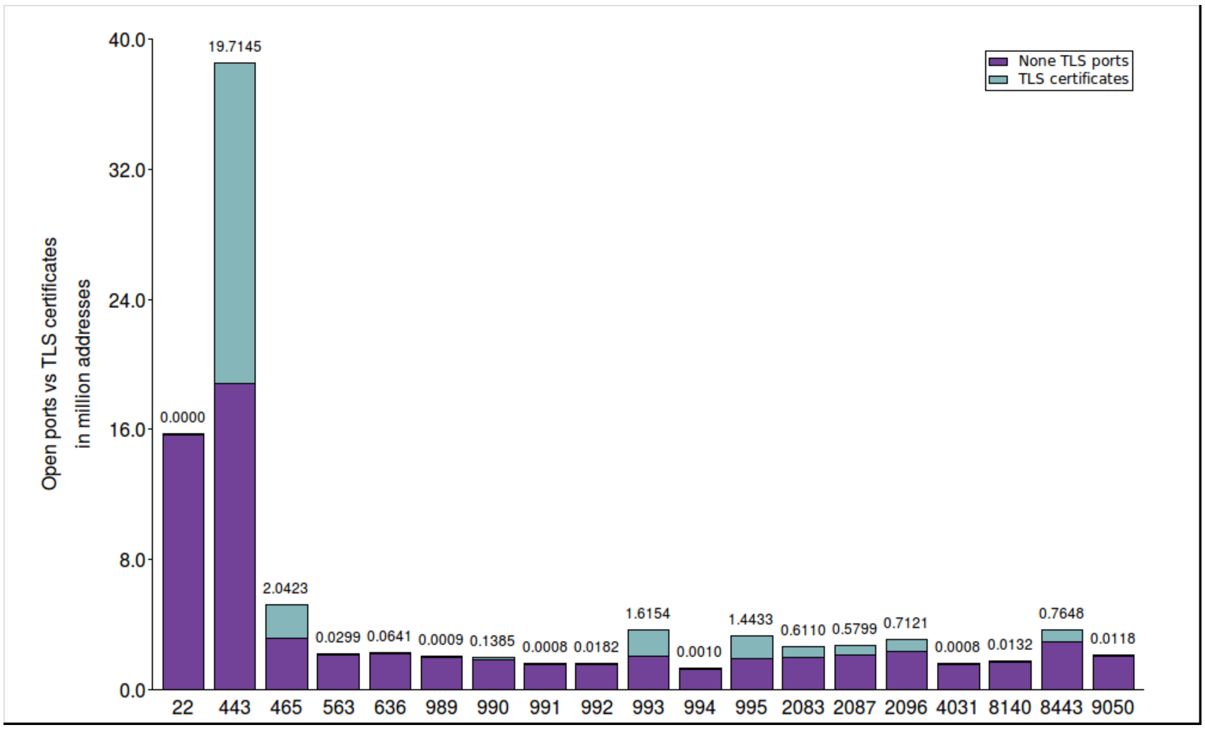


Figure 7.13: TLS approved service ports

The chart in figure 7.13 demonstrates this situation and it becomes obvious that less than 30 percent of the addresses with open ports really hold a valid TLS service which is publicly accessible.

The number of final found certificates out of **92.797.617** records was **27.762.509** (**28.60%**). The HTTPS port only covers more available TLS addresses (**51.17%**) than the sum of all other open TLS ports.

The relative distribution of the ports of collected services compared to the found services in percent can be seen in figure 7.14 on the following page. It shows that on several other ports the found certificates in comparison to their amount of open ports are also notedly higher. But due to the small absolute values this will not influence the statistics very much.

The complete list can be found in table C.2 on page 131.

Remark: This leads to the ascertainment that this large discrepancy should be investigated in more detail in later project steps.

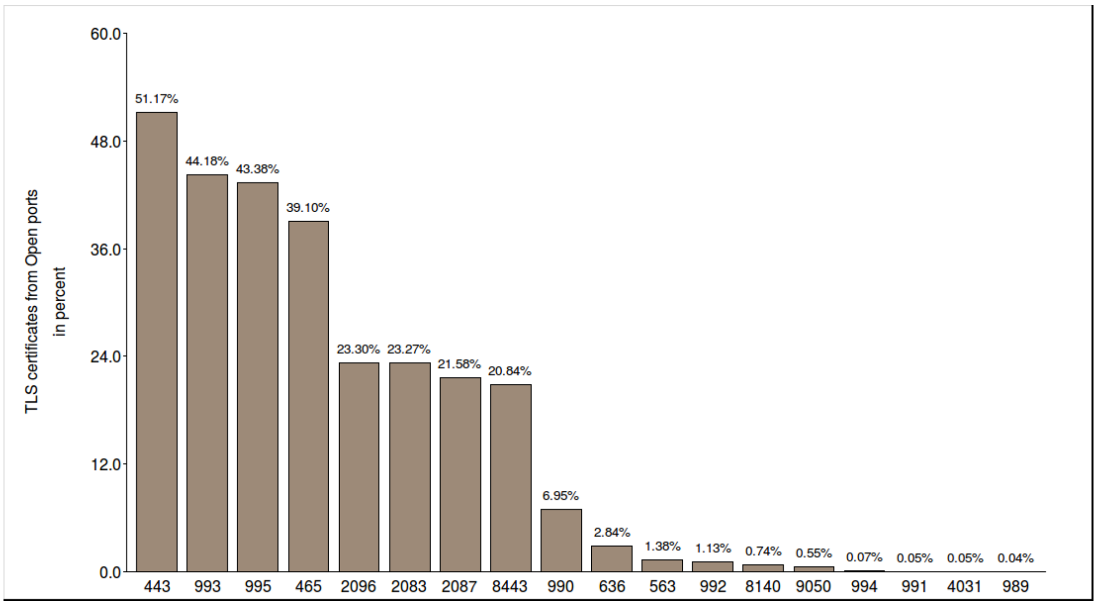


Figure 7.14: TLS certificates from open ports

7.4 Joining the databases

Until now two project phases have been distinguished, one from May until August 2015 and the second one from May until August 2016 and both parts have been treated separately and each part had its own database (see table 7.2). To work with the same database name in Phase 2 the existing "old" database was renamed to "oldservice" and a new one was created again with the name "keyservice".

Databases		oldservice	keyservice	Remarks
Scan IPs	records	63205106	92797617	not joined
	oldest	2015-06-23 22:38:19	2016-05-22 23:27:08	
	newest	2016-05-24 01:18:25	2016-07-19 16:13:10	
RSA certs	records	34882381	29476410/62678753	before/after join
	oldest	2015-11-10 21:31:03	2016-06-03 15:49:38	
	newest	2015-11-19 11:00:23	2016-07-28 10:21:06	

Table 7.2: Project databases

When analyzing the two areas it became obvious that parts of the data appeared in both databases and other parts contained distinguishable data.

According to the life cycle consideration in section 6.5.3 it has been decided to join the two database contents of the RSA certificates table to get a larger base for the factoring step and to analyze the distinctly enlarged system load. The not common data have been added to the new database to increase the amount of possible and proved TLS related information for the factoring step.

The arguments not to join the "scan" records were that this table has to be more topically than the "rsa_certs" which can still provide keys even if the certificates are outdated (refer to section 6.4.1 for the related aspects).

An additional TCL script has been programmed to join the tables "rsa_certs" from the two databases. The environment and the implemented database logic was:

1. The identification fields are the IP address, the service and certificate hash as it is possible to have different certificates for the same service.
2. Some fields have been removed as they were not necessary for the process (pubkey_algo, signature_algo) and one field has been added ("level" for managing the life cycle, refer to section 6.5.3).
3. Only non-existing records in the new database have been added and existing records have been ignored.

The program syntax of the tool **"utl/join_rsa.tcl"** can be seen in appendix A.12.1 on page 124. It also uses the block execution logic for the parallel processing mode therefore the new sequence entry "next_joinval" has been created in the sequence table (refer to section 6.3.4 for details).

The evaluation result from the log file shows:

```
begin: 2016-07-03/20:31:50, end: 2016-07-05/19:51:25, s_scanned=34882381,
    s_errors=0, s_success=33202343, s_existing=1680038, s_runtime=1d
    23:19:35.000(170375 sec.)
```

This represents a processing rate of **204.7** records per second and an effective import rate of **194.9** RSA records into the database. Existing PEM certificate files have not been imported into the new directory structure (refer to section §2.6).

A surprising result of that import was that only **1.680.038** imports (4,8%) has been rejected due to existing records by 47% of new imported records compared to the already existing database records.

This may have two diverging reasons:

1. Many certificates have been changed within one year
2. The new set of certificate records is only a small part of the complete Internet

The detailed investigation of that open issue is also left to the future of project.

7.5 Factorization

Calculation of the coprime base is part of the work "Copri - Factoring RSA Keys" of Martin Wind at the FH-Joanneum Kapfenberg.

7.6 Recalculation

Recalculation after changed records is part of the work"Copri - Factoring RSA Keys" of Martin Wind at the FH-Joanneum Kapfenberg.

Chapter 8

Data post-processing

This chapter explains the intended usage of the collected data as described before and it touches also parts of the planned service which are not covered by this work in detail.

The main goal is to provide many RSA public keys (more exactly the RSA n-factors only) which will be used to find eventually broken keys by factoring a coprime base. The results from the factoring process will be used to inform the user about a broken key.

An important part for a trustworthy service is the communication with the involved persons and organizations which is in this case the “whole world” because the requirement “Scanning the complete Internet” affects every person which has a public available IP address with a service behind.

Due to the fact that many administrators will grade the port access to their public IPs as malicious activity, it is important to communicate with the responsible persons to get the admission to continue the scan process or to take actions that these service ports will not be accessed anymore in future.

The management of this area is another important part of this chapter.

Finally statistical information about the scan process and the results will be given to create a deeper understanding of the planned service and it may help to improve or adapt it if the environment changes.

8.1 Transferring data to the factoring process

As described in section §2.5 only the public keys are forwarded to the factoring module. A program which was already programmed 2014 (see also section 4.5) has been extended to work with the database (refer to appendix A.1.2 on page 107 for the usage message).

A public key export is basically a database select query and the results are written to a file in the MPZ format specified by (Granlund and GMP development team, 2014). A typical call may look like:

```
$ ./0.8/manage_mpz.tcl uniq -c "key_size>=(1024-512) and key_size<(1024+512)
    " -f dat/uniq_1024d512.mpz
Multi precision MPZ modulus file management module 0.8
Start: 2016-07-27/16:08:11
Write a unique key list to file, scanning database ...
Selecting: distinct encode(rsa_n,'hex') from rsa_certs where key_size
    >=(1024-512) and key_size<(1024+512) ...
Database time: 01:58:44.730
Found 1832924 distinct public keys values ...
 1832924 records written to file dat/uniq_20160727_1024d512.mpz
```

```
Finished: 2016-07-27/18:07:23
Complete runtime: 01:59:12.596
```

This program call exports all distinct public keys with a size between 512 (1024-512) and 1535 (1024+512-1) bits. The key size value is retrieved from the RSA record and written to the result file "dat/uniq_1024d512.mpz" which is a suitable input for the factoring step.

8.2 Reverse lookup - back from calculation

The factoring process delivers the results as text file. The broken keys are written as JSON records with the keys "key" and the list of the two factors using the key "factors" along with some additional internally used key-value pairs. To get the corresponding RSA records the "key" record is used to search in the database whereas the DB design already contains the necessary search index (see also section 7.3.2). The SQL queries are defined accordingly in the tools.

As implementation the script **"search_cert.tcl"** has been created, the usage can be seen in appendix A.1.8 on page 108. This program is able to retrieve the information from a list of public keys or from a JSON file which comes from the factorization module. The public key values must be specified as decimal integers. For each database record the result contains a record name and the value in one line separated by a TAB character, the RSA-n value is provided as decimal and as hex value.

```
$ 0.8/search_cert.tcl 28497964500720314315253852748641377239911027643203566
          93242386469259040543377427409691335750465436363055325436063558485
          62745378707354311729884722624381507389954762098287179276194649927
          26555556494130303202660162282998414929930218687192328771695920905
          11102985901444956062194680115411530825391497498396487460211950816
          39756905982108593507486678488570780829236102380673969494392240307
          17828723542246638003195344902425463589842288245012196306296586333
          05232164500254195749787858857187143040937645294034999331872787151
          22792428387184320781937518438974253160530847664566652049663130784
          50740022030058381658990290797322362334492551

id      64522
ip_address      212.158.128.68
service 443
last_mod        2016-06-04 23:57:19.171095
status  ok
subject emailAddress=info@czware.cz,CN=212.158.128.68,OU=QIS,O=CZWARE s.r.o
    .,L=Prague,ST=Czech Republic,C=CZ
issuer  emailAddress=info@czware.cz,CN=212.158.128.68,OU=QIS,O=CZWARE s.r.o
    .,L=Prague,ST=Czech Republic,C=CZ
chain   /C=CZ/ST=Czech Republic/L=Prague/O=CZWARE s.r.o./OU=QIS/CN
    =212.158.128.68/emailAddress=info@czware.cz | /C=CZ/ST=Czech Republic/L=
    Prague/O=CZWARE s.r.o./OU=QIS/CN=212.158.128.68/emailAddress=info@czware
    .cz
rsa_n(hex)      e1bf533477b790d87976a759b7f1aae12b5eb6876070e7127ca1dcf1c
        e6ef55644bfd4f0585883328a220aa9e6d0477ccf1465cd8f0dbc9aa2e2a5aced
        2ec8700415017f250557639c005d7cd61e4053c7f01095570f0beb1ce77b1bdc4
        b47b4bc97e80952eabbbd2fd371c1c6792d26492e99b245498652efdc244482f2
        5ebb1441898b527ca65fa0e0303b839c6bfea0a62673a5adbef2530e17a0a9f74
        83dd14d21a7e05b7f6b6a7df547f08013ce41500d6a6db2a280e7b4f625b92938
        3017828bfc6dd3adb5a03d49c1b98dc559ee6c18465fc8cd1c5b6a18d8138e40f
        3436f3d330fb927963d692b2987395ab08b5d48998f5d98f0cd740838fa743787
rsa_n(dec)      284979645007203143152538527486413772399110276432035669324
        23864692590405433774274096913357504654363630553254360635584856274
        53787073543117298847226243815073899547620982871792761946499272655
```

```
                55564941303032026601622829984149299302186871923287716959209051110
                29859014449560621946801154115308253914974983964874602119508163975
                69059820859350748667848857078082923610238067396949439224030717828
                72354224663800319534490242546358984228824501219630629658633305232
                16450025419574978785885718714304093764529403499933187278715122792
                42838718432078193751843897425316053084766456665204966313078450740
                022030058381658990290797322362334492551
        rsa_e   65537
        key_type        rsa
        key_size        2048
        not_valid_before        2014-11-04 04:41:15
        not_valid_after 2024-11-01 04:41:15
        link:   ./crt/212/158/128/68/p443.pem
```

The web tool (refer also to section §7.1) allows to do the Reverse Lookup on the server by calling this program. Subsequently the results are sent back to the user.

In the input fields either a decimal key value (Check public key) or a JSON file from the factoring results as described before can defined (Evaluate JSon file). The key value is evaluated by pressing the button “Send Value” and the JSON file is uploaded to the server if “Upload File” is clicked. The JSON file is handled like multiple key entries which are processed in one step.

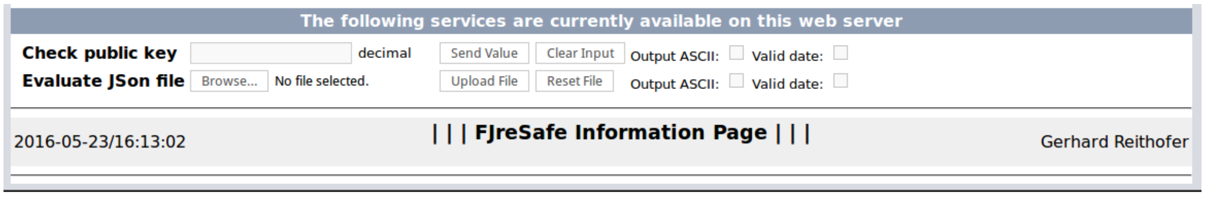


Figure 8.1: FJreSafe Info Page - Settings

The browser renders the result records according the selected output settings (see figure 8.1) either as HTML table on the web site (as shown in figure 8.2 on the following page) or it can be downloaded as CSV file if the option “Output ASCII” is activated.

If the option “Valid date” is activated the certificates are also checked for the valid period content and are only returned if it is valid at evaluation time.

The output may also contain the link ”Click for download” (figure 8.2) which allows the download of a saved PEM certificate (refer also to section §2.6) if it exists in the file system.

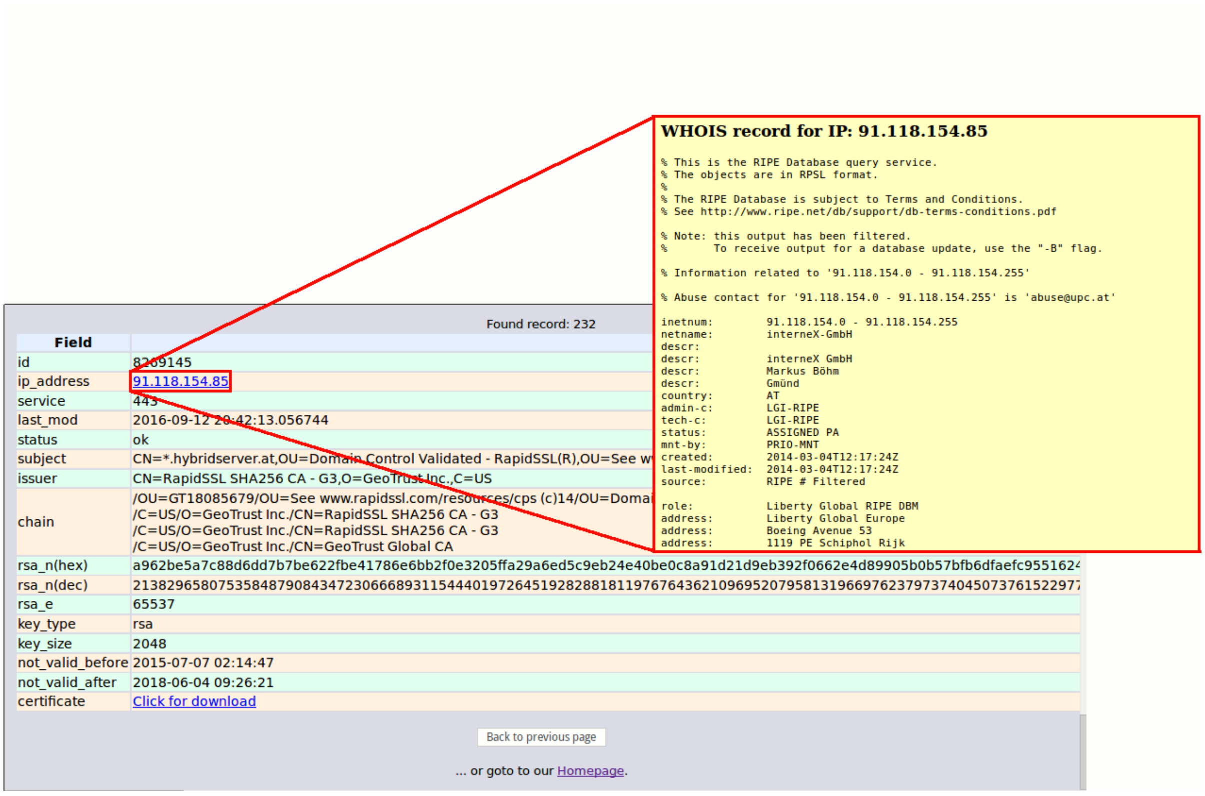
Additionally to the IP address a web link is provided which executes a WHOIS(Daigle, 2004) request in real time if clicked (also shown in figure 8.2 on the next page).

8.2.1 Personal user related data

A very important part for the planned service is the communication with users to provide automatic or on demand feedback if broken keys have been found and the possibility to communicate other information to registered users.

The attempt to get personal contact information from WHOIS records as first level user data source did not turn out as successful approach for several reasons.

The WHOIS protocol is only a simple clear text protocol which maybe used to get the necessary personalized data contained in the common fields. But in the RFC3912 in section 5 the following information can be read:



Found record: 232

Field	
id	8269145
ip_address	91.118.154.85
service	443
last_mod	2016-09-12 20:42:13.056744
status	ok
subject	CN=*.hybridserver.at,OU=Domain Control Validated - RapidSSL(R),OU=See w
issuer	CN=RapidSSL SHA256 CA - G3,O=GeoTrust Inc.,C=US
chain	/OU=GT18085679/OU=See www.rapidssl.com/resources/cps (c)14/OU=Domai /C=US/O=GeoTrust Inc./CN=RapidSSL SHA256 CA - G3 /C=US/O=GeoTrust Inc./CN=RapidSSL SHA256 CA - G3 /C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
rsa_n(hex)	a962be5a7c88d6dd7b7be622fbe41786e6bb2f0e3205ffa29a6ed5c9eb24e40be0c8a91d21d9eb392f0662e4d89905b0b57bfb6dfaefc9551624
rsa_n(dec)	2138296580753584879084347230666893115444019726451928288181197676436210969520795813196697623797374045073761522977
rsa_e	65537
key_type	rsa
key_size	2048
not_valid_before	2015-07-07 02:14:47
not_valid_after	2018-06-04 09:26:21
certificate	Click for download

... or goto to our Homepage.

WHOIS record for IP: 91.118.154.85

```
% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf

% Note: this output has been filtered.
%       To receive output for a database update, use the "-B" flag.

% Information related to '91.118.154.0 - 91.118.154.255'

% Abuse contact for '91.118.154.0 - 91.118.154.255' is 'abuse@upc.at'

inetnum:        91.118.154.0 - 91.118.154.255
netname:        interneX-GmbH
descr:
descr:          interneX GmbH
descr:          Markus Böhm
descr:          Gmünd
country:        AT
admin-c:        LGI-RIPE
tech-c:         LGI-RIPE
status:         ASSIGNED PA
mnt-by:         PRIO-MNT
created:        2014-03-04T12:17:24Z
last-modified:  2014-03-04T12:17:24Z
source:         RIPE # Filtered

role:           Liberty Global RIPE DBM
address:        Liberty Global Europe
address:        Boeing Avenue 53
address:        1119 PE Schiphol Rijk
```

Figure 8.2: FJreSafe Info Page - Certificate

> The WHOIS protocol has no provisions for strong security. WHOIS lacks mechanisms for access control, integrity, and confidentiality. Accordingly, WHOIS-based services should only be used for information which is non-sensitive and intended to be accessible to everyone.(Daigle, 2004)

Also the decision not to use SNI (refer to section §4.3) reduced the amount of user related data because only the server's certificate is found with the IP based approach.

Statement: The domain of the person related data should not be based on automatic retrievable public information. A trustworthy service should be published and managed through common used marketing mechanisms in a way as any other respectable service would do. This will increase the effort for the project apparently but this is the obligation to convince the customers that their private information is handled in a regardful and secure way.

In Austria person related data is protected by law(BKA.Österreich, 1999). For this reason and for the credibility of the service highest security demands have to be applied to this area.

Remark: The part "Personal user related data" will not be covered further in this work.

8.3 User feedback

Intended contact with users on security issues and automatic feedback (newsletter, informations, notification on key problems, . . .) is one basic target of the planned service.

Remark: Until now the management of the user related data is not defined (see also subsection before) therefore this part is also left out in this work.

8.4 Scanner obstacles

Massive scanning is a basic requirement for this project but this may also be an indication of malicious activities which are very common in public networks. Therefore many tools and even specific hardware has been developed to guard the own network infrastructure.

Even the first attempts of testing network scanners in this project created feedback from responsible administration staff where the scanned host addresses were located.

```
Von: infosec-noreply@caltech.edu [mailto:infosec-noreply@caltech.edu]
Gesendet: Donnerstag, 25. Dezember 2014 22:46
An: abuse@a1telekom.at
Betreff: 178.189.100.130 blocked at caltech.edu

178.189.100.130 was observed probing caltech.edu for security holes. It has been blocked
    at our border routers. It may be compromised.
 ...
```

Refer to appendix F.1 on page 143 for the complete message.

The type of reactions were very divergent. Some providers sent detailed connection information which allowed the calculation of the connect frequency (see the full message above) and others provided no useful or even senseless information:

```
Keep in mind that the source IP of our client has been sanitized for anonymity.
185.3.232.72:52427 > 10.10.10.40:22
```

Sometimes partially anonymized information was given:

```
May 22 19:25:10.392348 185.3.232.72.52427 > xxx.xxx.xxx.181.9050: S (src OS: unknown)
    518513609:51 8513609(0) win 1024
```

Others list the “connection rate” which makes it possible to estimate a maximum value which may not be classified as malicious:

```
/var/log/messages:May 23 01:10:15 darknet.superb.net Darknet: 185.3.232.72 exceeded
    connection attempt threshold to tcp:4031 43 times in a 30 minute period
```

From some detailed listings it have been calculated which time period of contact is treated as “probing”. In these cases only values less than 120 probes per minute (or less than two per second) have been detected.

8.4.1 How to reduce complaints

In the document “Analysis of the HTTPS Certificate Ecosystem” in the section “Reducing Scan Impact” the authors propose to use only single probe packets and to limit the scanning rate:

> When we perform a host discovery scan, an individual destination address receives at most one probe packet. At this scan rate, a /24-sized network receives a probe packet every 195 s, a /16 block every 0.76 s, and a /8 network block every 3 ms on average.(Durumeric, Kasten, et al., 2013, p. 3)

Statement: This is one of the main reasons why high speed scanning method cannot be recommended. An efficient collection methodology has to be developed and adapted if edge conditions vary. It can be expected that thorough planning and partitioning IP ranges will result in more reliable information compared to forcing high scanning throughput.

8.4.2 Proposed reactions

In this project each feedback message has been reacted on by mail with a small explanation about the project. A link to an explanatory information web page (see figure 7.2 on page 60) helped to make it easier to understand the project’s goals. The text explained the situation and it was also promised to take out the affected IP addresses or ranges from the scanning activities if the necessary information was provided in the feedback mails (refer to appendix F.4 on page 146 for an example).

By analyzing the feedback a rate of two probes in one minute seems to be an accepted rate per service but this still will probably vary in a wide range depending on several edge condition. As example the author also got a notion that SSH port is guarded much better than most other ports as some complaints were restricted to port 22 only.

Finding missing information (like to find the affected address range from the complaint) sometimes needed careful investigation, e. g.:

- The concerned domain can be determined from the email address context.
- An IP address can be found by doing a “nslookup” to the email server.
- The assigned IP address range can be retrieved by a WHOIS lookup.
- Web servers with the same domain can be used to find another point of contact.
- etc.

If several information artifacts do not contradict each other it can be taken as plausible. In such a case the addresses were added to the scanners exclude list and the table “scan“ had to be updated (refer also to section 7.3.1.1).

Feedback from the reaction mails was rare but mostly positive. Some blocked scan ranges could be removed again from the exclude list.

Statement: Finally the service acceptance will depend strongly on the integrity of the communication with the affected users.

8.4.3 Repercussion of address blocking

Inclusions to the scan exclude list in the range from a single IP address up to large address ranges were caused by reactions. To estimate the effect on the result a blocked IP range from the German provider “hetzner.de” will be investigated.

The feedback from Hetzner (refer to appendix F.2 on page 144 for details) allows to analyze several scanning aspects.

1. A method how to get the necessary address range.
2. The amount of the affected IP addresses if blocked.
3. The amount of already collected IP addresses.

The IP network range comes from the WHOIS record whereas one address from the provided list was taken as input:

```
$ whois 78.46.176.242
...
inetnum:        78.46.0.0 - 78.47.255.255
netname:        DE-HETZNER-20070416
org:            ORG-HOA1-RIPE
descr:          Hetzner Online AG
country:        DE
...
% Information related to '78.46.0.0/15AS24940'
route:          78.46.0.0/15
descr:          HETZNER-RZ-NBG-BLK5
...
```

The address range is the 15 bit network 78.46.0.0/15. A 15-bit net mask consists of $2^{17} = 131072$ IP addresses which covers $\frac{2^{17}}{2^{32}} \times 100 \sim 0.00305175\%$ of the complete address defined range, which can deliberately be ignored.

The range became included to the masscan exclude list by adding the following lines (see appendix A.8.1 on page 115).

```
# Abuse Message 1C4CAC21.txt
78.46.0.0/15
```

For calculation the already scanned IP addresses within of this range a simple SQL statement can be used:

```
keyservice=# select count(*) from scan where ip_address << '78.46.0.0/15';
 count
-------
 15585
```

To remove these addresses from the scan process the method described in section 7.3.1.1 can be used.

8.4.4 RBL/CBL blocking

Other obstacles appeared by activating automated RBL and CBL blocking from "honeypots" (http://-honeynet.org/) or from "conficker sinkholes" (http://net.cs.uni-bonn.de/wg/cs/applications/containing-conficker/), etc.

Access to these addresses are interpreted as virus or spambot activity. A typical feedback text from a conficker sinkhole is shown partly here (refer to appendix F.3 on page 144 for the complete text).

```
This IP is infected (or NATting for a computer that is infected) with the Conficker botnet.
More information about Conficker can be obtained from Wikipedia
Remember:  Conficker is not a spam sending botnet.  It does not send email or spam.  It does not
use port 25.
Please follow these instructions.
Dshield ...
```

These mechanisms are based on IP addresses and DNS entries and sites which use these block lists will block any access from the listed entries.

8.4.5 IDS, IPS, firewalls etc.

As many IDSs and IPSs allow to define rules which warn about stealth scans it is nowadays more probable to get blocked than it was a few years ago. But it is also common to use this as only one criterion from a list for unsolicited accesses in conjunction with others (like access frequency, etc.). Therefore the method how to probe services shall be limited to one or a few triggering methods to reduce access blocking.

Statement: It is advised to observe the community and the development of the security tools to react when the behavior or default handling in firewalls is changed.

8.4.6 Feedback vs. scanning speed

The mentioned examples demonstrate that there is a clear relation between the port probe frequency and negative reactions. The contradictory trends have a different influence factor on the scanning result.

- The faster the scan result the more negative reactions will appear because the high access rate will be interpreted as malicious attacks.
- If the scan frequency is lower less negative reactions will occur but the scan process will take more time to provide a useful information to the end user.

Another important effect must be preconceived. Even if a service owner accepts a specific access rate it will not create much influence on the data quantity directly. But the long term effect cannot be estimated and the acceptance may become lower.

Statement: The author of this work strongly recommends not to work against the community and network responsible people. The chance to loose the confidence of a public service must be rated higher than the "fastest possible" result for one specific user. Nevertheless exactly this aspect should be controlled and optimized continuously to get a maximum success.

8.5 Scan error management on Level II

Documented errors which will be created by the Level II scan process.

The error message texts are filled into the "status" field of the table "scan" while managing the certificate data. This controls the transition between Level I and Level II as described in section 6.5.3 on page 57.

The top 15 are:

```
  count   |    %    |                statustext
----------+---------+-------------------------------------------------------
 40503588 |  43.647 | timeout
 27762510 |  29.917 | ok
 15195367 |  16.375 | blocked
  3026784 |   3.262 | handshake failed: 5 attempts
  2983096 |   3.215 | handshake failed: unknown state
  1354009 |   1.459 | handshake failed: unknown protocol
  1197553 |   1.290 | handshake failed: socket is not connected
   267247 |    .288 | wrong algorithm type
   251642 |    .271 | handshake failed: unsupported protocol
   219239 |    .236 |
    12515 |    .013 | handshake failed: sslv3 alert bad record mac
     9190 |    .010 | handshake failed: wrong version number
     6523 |    .007 | handshake failed: unable to find public key parameters
     3071 |    .003 | handshake failed: wrong cipher returned
     2328 |    .003 | handshake failed: dh key too small
...
```

The complete list from this query can be found in appendix E.3 on page 139.

The "timeout" flagged services are mainly incorrectly configured servers (a very few may be honeypots, etc.). It shows that many servers (**43.64%**) do have open ports but no real service running on it and only **29.2%** are "ok" (i. e. they provided a certificate - which can also be outdated or revoked) and **16.5%** have been blocked by configuration.

All "handshake" errors and "wrong algorithm type" error are messages from the OpenSSL library which will not be analyzed more in detail here and one certificate ("data_insert" error) could not be parsed at all (refer to appendix E.2.2 on page 137 for the PEM certificate).

```
$ cmd/collect_pubkeys.sh 146.247.90.133 443
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 18441688780213926796 (0xffee0a3cc2f53b8c)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=FR, ST=France, L=Malataverne, O=NOVARC, OU=SI, CN=ocsinventory-ng/
            emailAddress=laurent.pons@novarc.com
        Validity
            Not Before: Mar  6 14:22:23 2013 GMT
            Not After : Bad time value
```

The **219.239 (0.236%)** services without an error message can easily be rescanned by extracting the IP/port data from the database and using the file input option of the certificate retrieve program.

```
$ utl/select2CSV.sh "select ip_address,service from scan where status=''" -o
    rescan_20160910.csv
Start: 2016-09-16 14:33:04
219239 rescan_20160910.csv
Finish: 2016-09-16 14:33:45
$ 0.8/retrieve_cert.tcl -i rescan_20160910.csv
```

8.5.1 Scan data - Level II

The separation of the two worlds (phase 1+2, refer to section §6.1 for details) was done by using the "status" field of the table which now becomes counted:

```
keyservice=# select rowcount('rsa_certs');
 rowcount
----------
 62678753
(1 row)
keyservice=# select count(*),status from rsa_certs group by status;
    count | status
----------+--------
        4 | sock
 27940718 | ok
        6 | off
 34738026 | imp
(4 rows)
```

The "new" collected dataset contains **62.678.753** records but only **27.940.718** certificates have been successfully collected in the second project phase (status='ok'). Only this set has been taken into account as "correct" collected certificate records and the keys from these records have been used for the factoring process.

8.6 RSA certificates and public keys

Public keys are stored per certificate identified by IP address, service port and the certificate's SHA1 hash value (refer also to section 7.4 on page 78).

8.6.1 Available records, new available and distinct certificates

To separate missing "old" and maybe faulty content only new data will be taken into consideration for comparing the complete amount of certificates with the "new" ones (status='ok ').

```
keyservice=# select count(distinct sha1_hash) from rsa_certs;
  count
---------
 8755651

keyservice=# select count(distinct sha1_hash) as distinct, count(*) as complete from
    rsa_certs where status='ok';
distinct | complete
---------+----------
 7380876 | 27940718
```

The difference of all distinct and the distinct amount of the newly imported certificates shows that there is a large overlapping of RSA records because from **34.738.032** imported records are only **1.374.775** (**3.96%**) common with the old database.

This reinforces the findings at the end of section §7.4.

8.6.2 Keydata from key export files

Another source for getting the amount of unique keys are existing key export files. This can be achieved with the tool **"manage_mpz.tcl"** as described in 8.1. By using the "count" option the number of keys can be extracted from a key export file. To get the number of 1024-bit keys from the export file "uniq_2016084_1024d512.mpz" (keys with 1024±512 bit) the following command can be used:

```
$ 0.8/manage_mpz.tcl count -f dat/uniq_20160804_1024d512.mpz
Multi precision MPZ modulus file management module 0.8
1832923 records found in file dat/uniq_20160804_1024d512.mpz, size=287190636.
```

8.6.3 Multiple use of certificates

Many providers use server certificates for many users because not every web page is hosted on its own physical server. These "virtual servers" often use a single certificate, which means that the same public key is used for many services which typically is called a "wild card certificate"[1].

From the **7.380.876** keys **5.906.613** (80%) are used for only a single service, all other **1.474.263** (20%) certificates are used repeatedly.

Multi usage	SHA1 certificate hash vaue
1137249	D91044D7BAAC4B1695B463D615D60DF796AB6CD7
909722	81D075440534DA06C1ED5AC48351A9A4978648AA
802591	455DBB43E5B704272E873AA0623F8DFE1EB399A5
362813	8A4F195A0E66D8AD9A56F6D810C3A0CC1C276932
256518	16D797F4BB60D1C29B43664A50DD54E58B00DEAF
189818	160ED722129930BE4A4B452E03CF215E1949DE2E
186700	D70EB9E71F1EF7BB11EE5DB659A8BB4C2E3ADD0A
148428	6B1EDFA2ED15058CA8F8CD41561C3B20E1498C0C
130008	9141B50CEE603888DC45F4F285C1C8844FE12221
127942	50433D50E27D25B7C365BAF5896635F0E7CA6D78

Table 8.1: Multiple certificate usage - top ten

The **Top ten** contain **4.251.789** certificates which is **15%** of all collected certificates (**27.940.71**).

The winner's subject: "O=Mini Webservice Ltd,ST=Some-State,C=PL", this 1024-bit self signed certificate is used over one million times, the NET and ORG sections of the WHOIS query for IP address 99.46.252.127 are shown:

```
 ...
NetRange:       99.0.0.0 - 99.127.255.255
CIDR:           99.0.0.0/9
NetName:        SBCIS-SBIS
NetHandle:      NET-99-0-0-0-1
Parent:         NET99 (NET-99-0-0-0-0)
NetType:        Direct Allocation
OriginAS:       AS7132
Organization:   AT&T Internet Services (SIS-80)
RegDate:        2008-02-25
Updated:        2012-03-02
 ...
OrgName:        AT&T Internet Services
OrgId:          SIS-80
Address:        3300 E Renner Rd
Address:        Mailroom B2139
Address:        Attn:IP Management
City:           Richardson
```

[1]Wild card certificates must be distinguished from SNI, see also section §4.3

```
StateProv:       TX
PostalCode:      75082
Country:         US
RegDate:         2000-06-20
Updated:         2016-06-17
 ...
```

Using the Censys repository (refer to section 3.1 on page 20) the details overview with further detail information links can be retrieved by using the following URL:

https://censys.io/ipv4/99.46.252.127

8.7 Valid certificates according the certificate lifetime

As already discussed in section 6.5.3 certificates have a time limited validity. In this context the change of this time valid state will be investigated[2].

Typical questions are: How many certificates were too old at the beginning of Phase 2 (2016-06-01), how many are outdated today (2016-09-18) and how many are valid in future?

```
keyservice=# select count(*) from rsa_certs where status='ok' and not_valid_after<'2016-06-01';
 20621461
keyservice=# select count(*) from rsa_certs where status='ok' and not_valid_after<now();
 26958220
keyservice=# select count(*) from rsa_certs where status='ok' and not_valid_after>=now();
 35741295
```

Within the $3\frac{1}{2}$ months **6.336.759** (10.1%) certificates ran out of their valid period, **20.621.461** (32.9%) were already outdated on 2016-06-01 and **35.741.295** (57.0%) from **62.678.754** are still valid after 2016-09-18.

8.7.1 Valid certification periods

First of all a complete overview about the validation periods has been extracted.

beg_min	beg_max	end_min	end_max
1902-11-01 13:06:05	10000-01-01 00:59:59	1901-12-16 02:14:02	10000-01-01 00:59:59

Table 8.2: Complete validation time range

A validation date range from 1901 until year 10000 was surprising and needed further investigation plus a check for certificates whose begin date is after the end date had to be made.

```
select count(*) from rsa_certs where not_valid_before>not_valid_after and status='ok';
 14758
```

In **14.758** certificates the "not_valid_after" date is earlier than the "not_valid_before" date.

Remark: All these certificates have never been valid.

[2]We did not check the revoke state of a certificate nor did we consider the trustworthy of the cert certification chain or rely on a specific trusted root certification authority. But as the chain data is saved in the database this should be possible if needed.

In further investigations it was found that there are only a few certificates with very unexpected validation ranges though some specific dates showed also increased values.

```
not_valid_before<='2002-12-31'
        1902 |      2
        1903 |      1
        1905 |      1
   ...
        1969 |      2
        1970 | 274579
        1971 |     27
        1972 |      4
   ...
        1997 |     61
        1998 |   3375
        1999 |  10964
        2000 |  49142
        2001 |  15985
        2002 |  21486
```

The peak in 1970 is most probably due to implementation errors as the date “1970-01-01 00:00:00” is the Unix time stamp start value 0, i. e. if no correct value is specified this date may be created and in year 2000 the amount of certificates again jumped up (new Y2K certificates).

```
not_valid_before>='2017-01-01'
      2017    35
      2018    32
...
      2164    12
      2165   104
      2558     1
     10000     4
```

The extreme values (from year 1901 to 10000) are curious but not relevant as these values were desultory and even if there were larger peaks they were too small to influence the statistics significantly.

Remark: Nevertheless this may be worth to investigate it for security reasons in future.

8.7.2 Certificate lifetime investigations

Valid periods from -132 years has been found (3 certificates, the “not_valid_after” date is 132 years before “not_valid_before” date) and valid periods up to 8003 years with the end date 10000-01-01 00:59:59.

The found validation dates were in many cases incorrect and may have been created a longer time ago.

More detailed investigations have been made in a time which is nearer to the current date. Between 2010-01-01 and 2025-01-01 which is an investigation range of 15 years. All ranges with negative periods have been eliminated.

The “current” statistics overview can be seen in figure 8.3 on the next page. The chart shows the start dates (“not_valid_before” in red, marked with '+') and end dates (“not_valid_after” in blue, marked with 'x') of all found certificates from the begin of 2010 until the end of 2024.

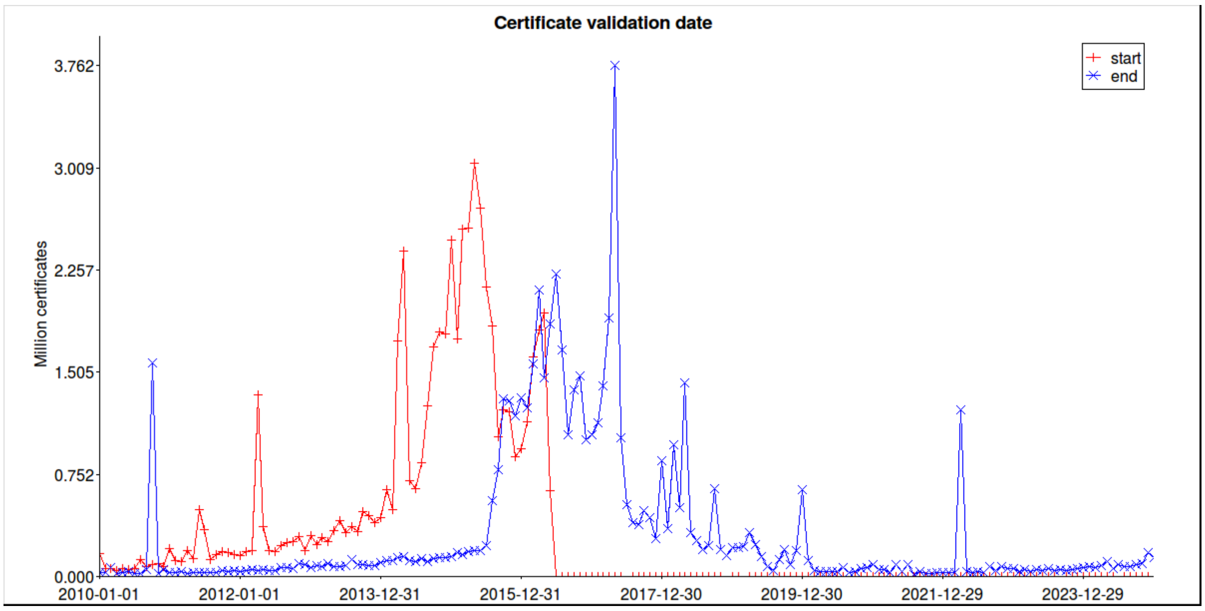


Figure 8.3: Certificate validation times

The five "smallest" cert lifetime values from 2010-01-01 . . . 2025-01-01

```
   ip_address     | srvc|  not_valid_before    |   not_valid_after    | lifetime
-----------------+-----+---------------------+---------------------+----------
 54.207.51.132    | 443 | 2016-05-07 00:53:11 | 2016-05-07 00:54:51 | 00:01:40
 54.72.26.74      | 443 | 2016-06-08 02:48:45 | 2016-06-08 02:50:25 | 00:01:40
 54.213.152.140   | 443 | 2016-04-18 00:09:32 | 2016-04-18 00:11:12 | 00:01:40
 64.69.83.250     | 443 | 2016-04-18 00:11:32 | 2016-04-18 00:13:12 | 00:01:40
 212.36.73.146    | 993 | 2016-03-28 16:57:41 | 2016-03-28 17:07:47 | 00:10:06
```

The five "largest" cert lifetime values from 2010-01-01 . . . 2025-01-01

```
  ip_address      | srvc|  not_valid_before    |   not_valid_after    |       lifetime
-----------------+-----+---------------------+---------------------+--------------------
 45.123.1.69      | 443 | 2010-01-04 20:13:59 | 2024-12-31 20:13:59 | 5475 days
 66.235.154.1     | 443 | 2010-04-08 14:05:12 | 2024-12-31 01:00:00 | 5380 days 10:54:48
 66.235.154.4     | 443 | 2010-04-15 18:48:53 | 2024-12-31 01:00:00 | 5373 days 06:11:07
 13.91.3.17       | 443 | 2010-05-10 08:00:00 | 2024-12-22 07:00:00 | 5339 days 23:00:00
 104.40.17.48     | 443 | 2010-05-10 08:00:00 | 2024-12-22 07:00:00 | 5339 days 23:00:00
```

The five "largest" cert lifetime values from 2010-01-01 . . .

```
   ip_address     | srvc|  not_valid_before    |  not_valid_after     |        lifetime
-----------------+--- -+---------------------+---------------------+---------------------
 167.121.11.36    | 443 | 2010-04-01 20:39:03 | 9999-12-30 19:39:03 |2918194 days 23:00:00
 167.121.111.201  | 443 | 2010-04-01 20:39:03 | 9999-12-30 19:39:03 |2918194 days 23:00:00
 203.81.25.236    | 443 | 2010-09-09 12:14:50 | 9999-12-31 16:59:59 |2918035 days 04:45:09
 50.240.187.85    | 993 | 2010-12-22 06:45:44 |10000-01-01 00:59:59 |2917931 days 18:14:15
 75.149.217.51    | 443 | 2011-01-17 01:29:33 |10000-01-01 00:59:59 |2917905 days 23:30:26
```

8.7.2.1 Lifetime periods

The lifetime periods vary in a wide range. The most used periods in days over the complete set (62M certificates) are shown in table 8.3.

#	days	used count	%
1	365	20095475	32.06
2	3650	4691419	7.49
3	1096	3779880	6.03
4	1095	3093500	4.94
5	730	2446184	3.90
6	366	1859280	2.97
7	364	1363494	2.18
8	1825	971335	1.55
9	7300	965219	1.54
10	731	923888	1.47

Table 8.3: Lifetime periods - top ten

A few certificates with striking time validation entries

Very long running certificate 13.93.212.202:8443 (7985 years), refer to appendix E.2.3 on page 137.

```
Sstat(notAfter)  = Dec 31 23:59:59 9999 GMT
Sstat(notBefore) = Jan  4 23:44:33 2015 GMT
```

One second time validation period. Not Before date is identical to Not After date, refer to appendix E.2.4 on page 138.

```
Sstat(notAfter)  = Jan 27 18:48:24 2011 GMT
Sstat(notBefore) = Jan 27 18:48:24 2011 GMT
```

Minus one day validation period. Not Before = May 25 10:09:43 2016 and Not After = May 24 10:09:43 2016, refer to appendix E.2.5 on page 138.

```
Sstat(notAfter)  = May 24 10:09:43 2016 GMT
Sstat(notBefore) = May 25 10:09:43 2016 GMT
```

Invalid validation period of -26,6 years. Not Before is Sep 23 16:49:29 2007 and Not After is Feb 21 10:21:13 1981,refer to appendix E.2.6 on page 138.

```
Sstat(notAfter)  = Feb 21 10:21:13 1981 GMT
Sstat(notBefore) = Sep 23 16:49:29 2007 GMT
```

8.7.3 Used stream cipher suites

The cipher suite is not a necessary part of the security project itself but as the information is present the cipher usage overview is evaluated.

The Top ten of the used Cipher Suites is show in table 8.4, the complete list can be found in appendix E.4.1 on page 140.

#	used count	cipher	%
1	17392469	ECDHE-RSA-AES256-GCM-SHA384	27.74
2	9969981	AES256-SHA	15.90
3	8752833	DHE-RSA-AES256-GCM-SHA384	13.96
4	7686206	DHE-RSA-AES256-SHA	12.26
5	4405751	ECDHE-RSA-AES128-GCM-SHA256	7.02
6	3113879	RC4-SHA	4.96
7	2587127	AES128-SHA	4.12
8	1916487	AES256-GCM-SHA384	3.05
9	1650812	ECDHE-RSA-AES256-SHA384	2.63
10	1446109	ECDHE-RSA-AES256-SHA	2.30

Table 8.4: The most use Cipher Suites - top ten

8.8 Key size investigations

The key size is one important factor for the security of the RSA crypto method. As computing power increases constantly the increase of the key size must go along with that development to keep the security level.

At current state the BSI(BSI, 2016) recommends for the modulus a minimum of 2000 bits.

The “bit size” is stored in the column “key_size” of the table “rsa_certs” as number of significant bits and the value is extracted with the OpenSSL standard function:

```
int BN_num_bits(const BIGNUM *a);
```

8.8.1 Top ten used key sizes

The following list shows the ten top most used bit sizes from 191 values to get an impression how the distribution may be. The use of the 2048 bit keys decreases drastically compared with the next lower bit class (1024 bit) and also the higher one (4096 bit).

Apparent is the concentration of key bits which are near main sizes (4096, 1024, 512, etc).

```
keyservice=# select row_number() over() as "#",* from (select key_size,count
    (*) from rsa_certs where status='ok' and key_size is not null group by
    key_size order by count desc) as "#";
  #  | key_size |  count
-----+----------+----------
   1 |     2048 | 21381173
   2 |     1024 |  5751378
   3 |     4096 |   654857
   4 |     1040 |    60708
   5 |      512 |    37252
   6 |     1039 |    15061
   7 |     3072 |     8141
   8 |     2047 |     6684
   9 |      768 |     5143
  10 |     1280 |     3723
 ...
```

8.8.2 Highest and lowest values

The same evaluation as before but this time sorted by the key size.

```
  #   | key_size  |  count
-----+----------+----------
   1  |    16384  |       28
   2  |    16383  |        1
   3  |    15424  |        1
...
 189  |      512  |    37252
 190  |      511  |       32
 191  |      384  |        1
```

The complete result of this query is available in appendix E.4.2 on page 141.

8.8.3 Values grouped into nearest discrete values

To get an overview about the key size distribution the chart in figure 8.4 has been created where all key sizes were grouped together with similar sized values.

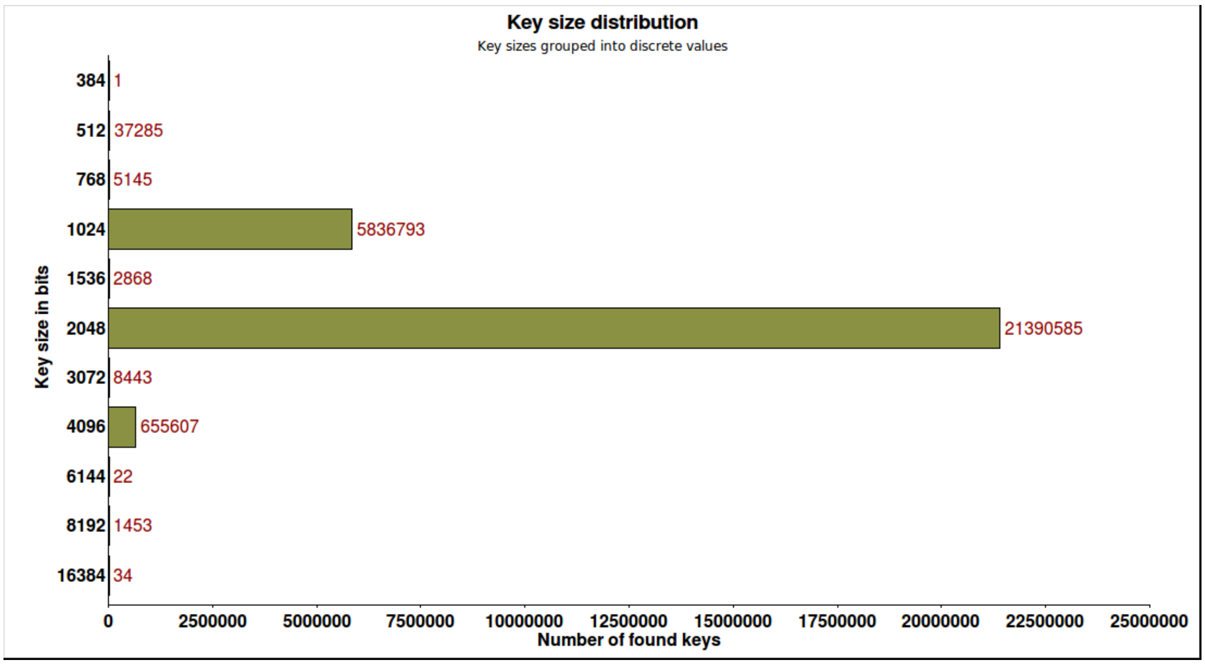


Figure 8.4: RSA public key size distribution in groups

This well demonstrates the usage of the main key sizes and shows the lowest value with **384 bits** (one single certificate) and **34** certificates with or near of **16384 bits** as highest bit size.

The most used key size is currently **2048-bit** with **21.390.585** (76.56%) keys, followed by the **1024-bit** keys with **5.836.793** (20.9%) and **655.607** (2.3%) **4096-bit** keys.

Chapter 9

Discussion

This chapter will give an overview of still open issues, proposals how to handle some specific situations and how to improve the existing methods, etc.

Another important aspect for the project is that the use of single system components are not comparable with common use of the typical IT tasks. Special hardware and special software may be necessary to provide the service in the intended quality.

Finally future ideas will be discussed, like currently postponed features or changes in the environment or requirements and their impact.

9.1 Improvement aspects

The project has shown that it is possible to scan the complete IPV4 address range within a reasonable time but the methods use existing tools which do not fulfill the requirements in any point.

As an example some "masscan" features have been found which influence the scan process negative and some appreciated options are implemented incomplete.

9.1.1 Port scanner "masscan"

Into the first category falls the behavior that the option "–randomize-hosts" is ignored and the randomization is always done. This is counterproductive because the input is already "randomized" (refer to 6.2.1). Another unwanted feature is that the scan process tries a handshake to get the "banner" information even if the option "–banner" is deactivated. The tool **"sta/show_bannerinfo.sh"** (refer to appendix G.1.5 on page 149) has been created to extract this information. Table 9.1 on the next page shows the "banner" statistics.

The "X509" entry is planned to retrieve the TLS certificate but this service is in development phase only and the few tests that have been made did not provide a valid certificate reliably .

Statement: Further development and optimization based on the existing "masscan" code may improve the overall scan performance noticeable but the effort for the special development must not be underestimated.

Amount	Banner
38	pop
64	imap
70	vnc
3085	ftp
6135	vuln
8492	title
14338	http
350369	unknown
484506	ssh
1044640	ssl
1047322	X509

Table 9.1: Scanning banner information

9.1.2 The Completeness pretension

The requirement to scan the “the complete Internet” can only be achieved in context with the IPV4 address space. The name based certificates (refer to 4.3 and 8.2.1 for the SNI related topic) could get additional information but the source will not be definable as exact amount of services. Typical Examples are:

- Several web sites offer “Reverse IP Domain Check” server lists (e. g. http://www.yougetsignal.com/-tools/web-sites-on-web-server/) and a search example for “www.tech-edv.co.at” lists 238 domains hosted on the same server.
- Commercial available server lists offer 62 Million+ Domain Names (e. g. https://www.youget-signal.com/list-of-domain-names/) but this cannot not be a “calculated maximum available” list as it can be done with the IPV4 address list.

Also very interesting could be the change from IPV4 to IP6 addressing. As IPV6 does not support SNI the statement regarding name based services can be ignored in that context. Additionally the address space is extended to 128 bit which leads to the following consideration:

$$\text{IPV4 } 2^{32} \approx 4.3 \cdot 10^{9} \qquad \text{IPV6 } 2^{128} \approx 3.4 \cdot 10^{38} \qquad \text{i. e. a factor } 2^{128-32} = 2^{96} \approx 7.9 \cdot 10^{28}$$

Considering the fact that actual hardware can be fully utilized with the IPV4 scanning the same approach will not be meaningful within the next few years. A name based approach based on DNS databases and on available collections will be more promisingly.

This requirement will also increase the demand to self manage the relevant service database.

9.1.3 Scan Level performance

Some aspects from scanning speed in Level I have already been discussed above and the “single step scan” (i. e. directly “access and retrieve” in one task) would also be an interesting approach. The disadvantage is that the very different data views (low vs. high network data load) must be handled within the same process concurrently. This would make it more complex and the error-prone therefore.

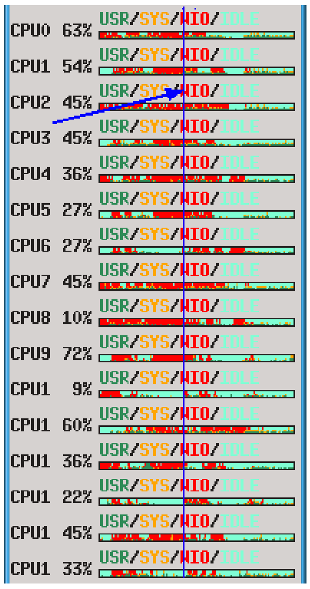
Statement: The investigated approach based on the division of labor is better scalable and if the division of the tasks is accepted the parallelizing is more easy implementable by externalizing the tasks to different servers.

9.2 Performance aspects

In section §4.4 some aspects have been defined, let's make a short subsumption here.

- Many CPU cores for parallel processing.
- Much memory for in-memory processing tasks (esp. for the database).
- High network performance Level I (low network load, high access frequency).
- High network performance Level II (high data flow-rate and computing power).
- Fast disk access (e. g. SSDs) for massive File IO.

For most of these requirements examples have been shown in the various chapters which reached the systems limits like the IO bottleneck shown in figure 9.1 near the blue marker line. In this situation almost no IO intensive task can be executed with an expected performance. This was a typical situation in many later database evaluations.



```
CPU0 63% USR/SYS/WIO/IDLE
CPU1 54% USR/SYS/WIO/IDLE
CPU2 45% USR/SYS/WIO/IDLE
CPU3 45% USR/SYS/WIO/IDLE
CPU4 36% USR/SYS/WIO/IDLE
CPU5 27% USR/SYS/WIO/IDLE
CPU6 27% USR/SYS/WIO/IDLE
CPU7 45% USR/SYS/WIO/IDLE
CPU8 10% USR/SYS/WIO/IDLE
CPU9 72% USR/SYS/WIO/IDLE
CPU1  9% USR/SYS/WIO/IDLE
CPU1 60% USR/SYS/WIO/IDLE
CPU1 36% USR/SYS/WIO/IDLE
CPU1 22% USR/SYS/WIO/IDLE
CPU1 45% USR/SYS/WIO/IDLE
CPU1 33% USR/SYS/WIO/IDLE
```

Figure 9.1: Disk IO bottleneck

There are two approaches to improve this situation:

1. Use more different IO components and partition the different data content accordingly (e. g. database table spaces on other disks than log files, temporary files on separate SSDs, etc.). This is a complex task and needs a well prepared organization.
2. Use more Servers in parallel. In this case the partitioning is solved automatically but the network needs extreme high performance to become comparable to powerful hard disk setup, esp. if they are organized as fast raids.

The aspect-list from the begin of this section can be simplified to:

- Use as much as possible CPU cores
- Use as much as possible disk spindles
- Use as much as possible memory
- Use high performing networks in multi-server environments

9.2.1 Experienced system restrictions

Despite the performance aspects discussed before a few examples where the high system load made it hard to work will be discussed. This also shows typical pitfalls which are caused by the extreme situations of the server's environment compared to a typical office workstation.

9.2.1.1 Interactive back draws

It is common and comfortable to use command line completion in nowadays systems. If a command is entered partly on the keyboard and the TAB key is pressed this mechanisms expands the command line to all matching commands.

Example: After entering "ls" and pressing the TAB key the command line is changed to the matching command if one exists. Otherwise it shows all possible matching commands:

```
$ ls
ls          lsblk       lscpu       lslocks     lsof        lspgpot
lsattr      lsb_release lsinitramfs lsmod       lspci       lsusb
```

The mechanism also work for command options and arguments. If after entering the command "ls " and a list of possible files is requested by pressing TAB twice, it may last ten seconds or more if the current directory contains a lot of files and the terminal is blocked while waiting for the result.

This situation is even worse if the working directory is on a remote file system because directory lookups on mounted directories are very inefficient.

9.2.1.2 Command line overrun

Some calculation scripts run over a large number of data files. To allow multiple files on the command line wild-cards can be used which are "evaluated" by the interactive shell. As the command line size of the shells is limited a maximum number of parameters in combination with the path name lengths of the files may be reached.

Processing a directory containing all port 443 temporary files shows this dilemma.

```
$ echo tmp/port_443/* | wc -c
2390056
$ cd tmp/port_443;echo * | wc -c
1755188
$ getconf ARG_MAX
2097152
```

As practical example the script **"stats_parscan.sh"** (refer to appendix G.1.12 on page 150 for the script usage) can be used.

```
$ sta/stats_parscan.sh -d -s -t tmp/port_443
sta/stats_parscan.sh: 81: sta/stats_parscan.sh: cat: Argument list too long

cd tmp/port443
$ ../../sta/stats_parscan.sh -d -t .
2015-08-21 01:00:47      2015-08-22 12:41:18       49568  4292117295
    5386322      1d 11:40:31      33419.6           41.9
```

In the first case the command line size is overridden which will lead to an error because the command line size is 2390056 (path+filename) which is larger than the limit 2097152.

The same call in the corresponding subdirectory will succeed bcause the command line size (all filenames only) is 1755188 which is smaller 2097152 .

9.2.1.3 File system limitations

The retrieved certificates are stored in PEM format (see also section §2.6). At evaluation time **27.940.010** certificates are stored in the file system for direct access and to provide a download possibility in the Web Interface (refer to 2.6). This information can also be used to re-import the data into the database in case of emergency.

Processing a large amount of files will need long run times, e. g. if the processing of a file needs 0.1 seconds one million files use over 27 hours).

This large amount of data is organized in a directory structure where each octet of the IP address represents a directory and in the last directory level a file named like "p443.pem" is stored. The number "443" represents the service port. This structure uses **51.981.544** inode entries (the native file size of all PEM files is 44 GB and the used disk size is of course higher) which is very efficient for direct access but working on "all certificates" is extremely time consuming. Typical examples:

- The creation of a compressed tar file of the complete certificate directory structure needed 2 days 19:27:07, the "zip rate" was **115,1** files per second, the amount of directories and files read from disk and written compressed to the tar file was **214,1** entries per second.
- Unzipping the database dump ("gunzip keyservice_20160813.dump.gz") which is a single file needed **01:02** hours, the uncompressed file size is **83.7** GB and therefore an unzip rate of **23.1** MB per second has been achieved.

9.2.1.4 Database limitations

Many database queries are only used from time to time. Evaluation which are used very frequently are optimized by creating the necessary indexes but too much index files also weigh down the database because on each insert and probably update the indexes must be actualized too.

An example for reaching inhibitory database loads is the export of the complete unique public key list **"uniq_20160804_all_keys.mpz"** which took **06:44:58** hours.

Some SQL queries had to be canceled because the memory usage exceeded the installed 48 GB, the tasks had to be divided to multiple smaller parts.

9.2.1.5 Memory consumption

The current implementation of the 2-step export of the tool **“manage_mpz.tcl”** (refer to appendix A.1.2 on page 107) has weak points. The first problem is that in all exports where the key size is a selection criterion a full can must be done, because this field is in no index present. The database should be designed in the context of the planned “select criteria”, but this can also run into a limitation (see the paragraph before).

The second problem is that the “database select” and “write to file” actions are currently implemented as sequential tasks, i. e. the complete result will be read into the memory (the client memory), then it is copied into the TCL’s object memory and after that it is written to the file. The installed 48 GB of RAM were not sufficient for the actual project’s database in some cases.

As countermeasure the interface design could be redesigned to direct read-write operation while retrieving the data from database.

Statement: Working in an environment necessary to fulfill the project’s requirement are in some situations very claiming. The large amount of objects make corrections or reevaluation long running tasks. Backups and data management require large disk spaces and long save and restore times. This must be taken into account or it must be aligned with other data intensive tasks. The working environments should be optimized to reduce wait time or managed accordingly.

9.2.2 Database optimization

The “productive” database functions can be seen as relatively optimized as all collection, store, retrieve functions work with good performance. But while working on this thesis many database evaluations had to be made for statistics reasons or just for checking the effect of a tested optimization step.

The evaluation performance were in many cases insufficient as many steps were running for hours and depending on the IO load the performance became retarded accordingly. This does not necessarily disturb the functionality in general but it is predictable that frequent analyzing becomes a central requirement for optimizing the service and therefore the amount of requested database queries will increase, even if it may not be necessary for the scanning work itself.

In context of the described limitations in section 9.2.1 the optimization issues will become essential in the database area. One already discussed basic method to distribute the resources can be applied. In this situation it would be necessary to have a “copy of the database” to allow resource consuming activities without disturbing the productive area and there are various implementations conceivable:

1. First of all the database model has to be analyzed in general and redundant information is one aspect for increasing file space (e. g. the multiple used certificate data). Normalizing the data model could improve this situation.
   Cons: A normalized data model must be re-optimized again and that does not necessary lead to optimal performance.

2. A simple idea is to create all sorts of indexes which are typically used.
   Cons: Indexes increase the database load as the record management affects many index tables.

3. The use of database partitioning. By defining tablespace on different physical hard disks the load on individual disks can be reduced and also other similar methods exist.
   Cons: The variance of the combination of tables, indexes and other data objects becomes increased rapidly and it is not easy to anticipate the consequences.

4. A simple way is making regularly exports (which is also necessary for data backups) and to import the into another “working database”.
   Cons: This approach requires large data transfer areas and exports and imports are heavyset operation and the necessary time must be guaranteed.

5. Another possibility is to develop database delta-sync mechanism which could reduce the amount of data transfer compared to the prior method.
   Cons: A delta-sync mechanism is usually complex and the final database load strongly depends on the implementation.

6. A database internal possibility is to create the corresponding views for the frequently used queries. But conventional views are typically just some sort of “stored SQL queries” and do not increase the performance. Special database objects like “materialized views” could solve this problem.
   Cons: A materialized view’s database load is also heavyset on data management and the storage use is increased significantly which also increases the file system’s requirement.

7. A high end solution may be using database clustering with the corresponding “intelligent” separation of the individual database management steps.
   Cons: Clustering and adequate use of the database is a complex task. If not planned thoroughly it will not be manageable with success.

Statement: Beside all pros and cons it must be considered that also each of these methods will have its limitations and it will require much investigation to optimize these limits. Nonetheless it will become necessary.

9.3 Service management issues

Within the project many tools have been developed to gather data and to store it in a way which allows efficient use. With increasing lifetime of this project additional and other demands will appear and the methods to react on these have to be developed.

9.3.1 Handling dynamic network changes

Until now no methods been developed to handle dynamically changing Internet data (refer to section 6.5.3). As it is not foreseeable which amount of “old” data can be handled no final decision can be made. The evaluations in section §8.7 indicates a monthly change rate of more than two million certificates which must be handled in short time and also the public key recalculation should be possible in short time.

Statement: Handling of the dynamic changes in the PKI ecosystem will be required to react in short time. Otherwise the quality of such a service will become rated low.

Chapter 10

Conclusion

In this chapter findings of the planned keyservice project are summarized and also issues which have been omitted so far are addressed.

This work is part of a larger project with the target to collect and investigate RSA public keys for finding multiple used prime factors. If factors are used more that once in a set of keys the factors can be calculated in near linear time by using DJB's coprime algorithm(Bernstein, 2005).

The work covers only the RSA key collection and management part and the calculation part is described by Martin Wind in his Master Thesis "Copri - Factoring RSA Keys".

10.1 The project's basics

The basic idea to partition the work into two parts, called Level I and Level II, has been confirmed in several situations. Single independent and repeatable steps ensure the traceability and guarantee the status of a specific task. Reaction on deviations from the standard case or faulty part steps can be repeated and repaired in short time.

10.2 Data management and interfaces

The data is stored in files and databases and the interface to the factoring module is a data export from a RSA key database. In Level I a finite number of objects can be stored, i. e. all IPV4 addresses (the "Randomized Linear IP-table" see 6.2.1) and a configured list of services is added for the next step. With this as input all IP addresses with possible related services can be found and are stored in the database table "scan". Level II searches for usable TLS certificates from the prior scan activity and stores the result in the table "rsa_certs". For one "scan" entity multiple "rsa_certs" records can exist (in fact the amount is theoretically unlimited).

From this table a list of public keys (RSA-n values) is exported into the file system using the MPZ-format(Granlund and GMP development team, 2014). The calculation step itself is is not part of this work.

From the factoring module a list of broken keys is returned as text file or in JSON-format (see 8.2 for details). With that information the affected certificates can be retrieved from the database.

Open issue: A web interface has been programmed which is able to provide the corresponding certificate information but currently no user management and only public access is implemented. The service user related part, the management of the users data and the communication flow is not defined until now. This is left open for the further project development and also the complete logistic and the legal part.

10.3 Scan blocking and service acceptance

The first part of the scan process is handled in a way as many illegal activities do. To take care of this information a list of servers and addresses must be managed which do not want to become scanned by the project. A broad acceptance will only take place if an open and transparent communication between the responsible of the security service and the managers of the concerned servers. Easy accessable documentation and explanation of the project's goals will be necessary to become accepted. Esteeming and open communication is very important and first successful feedback has been experienced.

Open issue: A simple prototype of an information site has been created (refer to figure 7.2 on page 60). In future this part has to be organized in a marketing like manner as any other confidential service should do.

10.4 Performance issues

In the section 9.2 on page 99 the topic "system overloads" have been discussed. Especial in case of interactive use a responsive feedback is expected. Therefore prioritizing the performance aspect in these cases should be rated very highly. The interactive parts should be designed with respect of strong independence of the others systems components and good responsivity.

10.5 Final statement

A security service as it has been designed is possible within the tested and evaluated range. But only a longterm project will confirm or rebut the success of the basic idea:

"Make the Internet to a more secure place now!"

Appendix A

Directory structure

```
Dir                 Description
--- ----------------------------------------------------------
0.8 Main TCL modules for scanning and collecting - see A.1
bak Various project data backups, not basically project relevant
bin Used binary scanner programs - see A.2
cmd Command scripts for manual calls - see section §A.3
crt Directory for collected certificates (very large!) - see section §A.4
dat Data directory for input or some investigations - see section §A.7
etc Global configuration directory - see section §A.8
img Icons, Images, screen-shots, etc.
lib TCL packages, user libraries, etc. - see section §A.9
log Log files for evaluations, error analyzing - see section §A.5
mod Scan module plugins for scanner - see section §A.6
scr Shell scripts, utilities for scanning, etc. - see  A.10 on page 120
sql SQL install and structure dumps - see A.11 on page 123
sta Statistics and evaluation routines (for Thesis only) - see G on page 148
tmp Temporary scanner output (also used for monitoring purposes)
utl Utilities - "other" stuff, tools, aux tools, ... - see A.12 on page 124
web Web site scripts and HTML files (Web service) - see A.13 on page 125
```

A.1 Main TCL modules

```
0.8/db_sim2.tcl ........ FileDB management module (A.1.1)
0.8/manage_mpz.tcl ..... Multi precision MPZ module (A.1.2)
0.8/parse_pkcs1.tcl .... Script based limited parser (A.1.3)
0.8/retrieve_cert.tcl .. Retrieve certificate tool (A.1.4)
0.8/rsa_fromfile.tcl ... Get RSA info from certfile (A.1.5)
0.8/save_scan.tcl ...... Scan result postprocessor (A.1.6)
0.8/scan_sequ.tcl ...... Level I scanning module (A.1.7)
0.8/search_cert.tcl .... RSA key reverse lookup tool (A.1.8)
```

A.1.1 db_sim2.tcl

```
$ 0.8/db_sim2.tcl -h
FileDB management module  0.8
 -h .. display this help text
 -i .. show current sequence (start) and count information
 -d .. [start [count]] - read count ip-addr flag pairs from start
 -a .. [start [count]] - read count ip-addresses from start
 -v .. [start [count]] - read count flag values from start
 -o .. [start [count]] - direct output of ip addresses from start
 -l .. [start [count]] list - search for ip range from list
 -r .. range [start [count]] - search ips in range (format IP/mask)
 -u .. write start ip flag [..] - write ip flag pairs from start
 -w .. write start flag [..] - write flag values from start
 -f .. specify an alternate flatfile db - def. ipv4shuffle
 -s .. specify an alternate sequence db - def. ipv4shuffle
 -n .. insert record numbers before each data line
```

A.1.2 manage_mpz.tcl

```
$ 0.8/manage_mpz.tcl -h
Multi precision MPZ modulus file management module 0.8
Usage: manage_mpz.tcl [-h] mode [-s start] [-n numrec] [-c condition] [-a] [-f filename]
  -h displays this help information
  The possible modes are:
    new .... a new file is created and the records are written to it
    add .... all found records data is appended to file
    read ... the requested data is read from MPZ file to display data
    uniq ... select unique keys and write MPZ file in mode 'new'
    write .. read binary data (.mpz) and write keys to text file (.dec)
    count .. read data and return only the number of values
    set .... do not read data, just set a new start value
  Option -c defines a specific database condition (SQL where clause)
  Option -s defines the start value for read records (def. 0)
  Option -n defines the number of records to return (def. 100000)
  Option -a for mode 'uniq' the filemode is set to 'append' (def. is 'new')
  Option -f defines a specific input/output mpz-file
  If the number of records is < 1 all selected records are returned
  The 'set' mode requires a numerical value with option -s
  Defaults: mode='', condition='id > 829', start=0, numrec=100000, file=./dat/export.mpz,
       append=0
```

A.1.3 parse_pkcs1.tcl

```
Script based limited certificate parser 0.8
Usage: parse_pkcs1.tcl [-k] [-c] pkcs1data ..
  -k .. following input is key data
  -c .. following input is a cert file
Default mode is -c
```

A.1.4 retrieve_cert.tcl

```
Retrieve certificate data from scan-list 0.8
 The adresses and ports are taken either from the global scanning
 table 'scan' or from the already collected rsa table 'rsa_certs'
Usage: retrieve_cert.tcl [-h] [-s start] [-n numrec] [-e unused] [-r rsalist] [-d savedb]
     [-c writecrt] [-w async] [-t timeout] [-f filter] [-a addr] [-p port] [-i csvfile]
  -h displays this help information
  If -a is specified no value is read from the database, -p defaults to 443
  If a csv file is specified the address/port combination is read from the file
  If Option -e is 1 only 'empty' values are selected (only new ones)
  If Option -d is set to 1 data is written to 'rsa_certs', else nothing is saved
  If -t is set to 0 a sync tcp socket communication is used (no timeout)
  If -r is set to 0 the table 'scan' is used as scanning list (sequence 'next_scanval'),
  if set to 1 the table 'rsa_certs' is used as scanning list (sequence 'next_rsa_val')
  With Option '-c' set to 1 the PEM certificate is stored locally as file
  Start value (-s) may be prefixed with + or - to calculate offsets to current value
  The -w requests sync tcp socket communication (no timeout)
  -f allows to specify a list filter (additional sql where clause to the select statement
     )
  Option -r 0 -e 1 corresponds to table scan with: status = ''
  Option -r 0 -e 0 corresponds to table scan with: status <> 'blocked'
  Option -r 1 -e 1 corresponds to table rsa_certs with: key_type is null
  Option -r 1 -e 0 corresponds to table rsa_certs without any filter
  Defaults: start=17, numrec=100000, empty=0, savedb=1, timeout=1000, async=1, rsalist=0,
      writecrt=1, filter=
```

A.1.5 rsa_fromfile.tcl

```
$ 0.8/rsa_fromfile.tcl -h
Retrieve or update RSA data 0.8
 from saved cert files using x509::pki TCL package
Usage: rsa_fromfile.tcl [-h] [-s start_id] [-n numrec] [-e unused] [-d savedb] [-a addr]
    [-p port] [-i certfile] [-f filter]
  -h displays this help information
  If -a is specified no value is read from the database, the file is retrieved from cert
     area
  Option -i defines the path to an existing PEM certificate file (disables savedb)
  Option -e selects the 'empty' values, which don't have a flag value
  Start value (-s) may be prefixed with + or - to calculate offsets
  -f allows to specify a list filter (additional sql where clause to the select statement
     )
  Defaults: start=0, numrec=100000, empty=1, savedb=0, addr=, port=443, filter=
```

A.1.6 save_scan.tcl

```
$ 0.8/save_scan.tcl -h
Hispeed scan result postprocessor 0.8
Usage: save_scan.tcl [-h] [-s start] [-n numrec] job_name
  -h shows this usage information
  -s .. start offset (def. 0)
  -n .. number of records, 0 for all (def. 0)
 job_name is a scan output file without directory and extension
```

A.1.7 scan_sequ.tcl

```
$ 0.8/scan_sequ.tcl -h
Sequential highspeed Level I scanning module for possible TLS services 0.8
Usage: scan_sequ.tcl [-h] [-d netdev] [-s start] [-p port] [-n numrec] [-m module]
  Defaults: netdev=, start=0, port=443,993, numrec=100000, module=masscan
```

A.1.8 search_cert.tcl

```
$ 0.8/search_cert.tcl -h
Search service of broken keys 0.8
Usage: search_cert.tcl [-h] [-v] [-a addr -p port] [-f jsonfile] [-d delim] [-l lo] [-h
    hi] [key ..]
  -h displays this help information
  If -v is specified only valid from/to dates are retrieved
  If record delimiter -d is specified, a CSV output is created
  Options -a and -p retrieve specific stored certificate
  Option -f defines a json input file
  Otherwise all entries are interpreted as n-factors in decimal
  -l and -h define factor limits, -l means (p/q <= lo), -h means (p/q => hi) and (p/q >
     lo && p/q < hi)
```

A.2 Binary scanner programs (bin)

The following executables have been tested for the Level I scan, "ipshuffle" (for usage see A.2.4) is necessary for creating the Linear Address Table (refer to section 6.2.1 on page 40). The C-programs have been newly compiled on the final platform (masscan64, zmap64), the Java scanner has been rebuilt from the source code (ipscan-linux-3.3.2.jar, ipscan-linux64-3.3.2.jar).

A.2.1 zmap

```
$ bin/zmap64 -h
zmap 2.0.0-RC1
A fast Internet-wide scanner.
Usage: zmap [OPTIONS]... [SUBNETS]...
Basic arguments:
  -p, --target-port=port        port number to scan (for TCP and UDP scans)
  -o, --output-file=name        Output file
  -b, --blacklist-file=path     File of subnets to exclude, in CIDR notation,
                                  e.g. 192.168.0.0/16
  -w, --whitelist-file=path     File of subnets to constrain scan to, in CIDR
                                  notation, e.g. 192.168.0.0/16
Scan options:
  -r, --rate=pps                Set send rate in packets/sec
  -B, --bandwidth=bps           Set send rate in bits/second (supports suffixes
                                  G, M and K)
  -n, --max-targets=n           Cap number of targets to probe (as a number or
                                  a percentage of the address space)
  -t, --max-runtime=ses         Cap length of time for sending packets
  -N, --max-results=n           Cap number of results to return
  -P, --probes=n                Number of probes to send to each IP
                                  (default=‘1’)
  -c, --cooldown-time=secs      How long to continue receiving after sending
                                  last probe  (default=‘8’)
  -e, --seed=n                  Seed used to select address permutation
      --retries=n               Max number of times to try to send packet if
                                  send fails  (default=‘10’)
  -d, --dryrun                  Don't actually send packets
      --shards=N                Set the total number of shards  (default=‘1’)
      --shard=n                 Set which shard this scan is (0 indexed)
                                  (default=‘0’)
Network options:
  -s, --source-port=port|range  Source port(s) for scan packets
  -S, --source-ip=ip|range      Source address(es) for scan packets
  -G, --gateway-mac=addr        Specify gateway MAC address
      --source-mac=addr         Source MAC address
  -i, --interface=name          Specify network interface to use
  -X, --vpn                     Sends IP packets instead of Ethernet (for VPNs)
Probe Modules:
  -M, --probe-module=name       Select probe module  (default=‘tcp_synscan’)
      --probe-args=args         Arguments to pass to probe module
      --list-probe-modules      List available probe modules
Data Output:
  -f, --output-fields=fields    Fields that should be output in result set
  -O, --output-module=name      Select output module  (default=‘default’)
      --output-args=args        Arguments to pass to output module
      --output-filter=filter    Specify a filter over the response fields to
                                  limit what responses get sent to the output
                                  module
      --list-output-modules     List available output modules
      --list-output-fields      List all fields that can be output by selected
```

```
                                 probe module
Logging and Metadata:
  -v, --verbosity=n             Level of log detail (0-5)  (default='3')
  -l, --log-file=name           Write log entries to file
  -L, --log-directory=directory Write log entries to a timestamped file in this
                                  directory
  -m, --metadata-file=name      Output file for scan metadata (JSON)
  -u, --status-updates-file=name
                                Write scan progress updates to CSV file
  -q, --quiet                   Do not print status updates
  -g, --summary                 Print configuration and summary at end of scan
      --disable-syslog          Disables logging messages to syslog
      --notes=notes             Inject user-specified notes into scan metadata
      --user-metadata=json      Inject user-specified JSON metadata into scan
                                  metadata
Additional options:
  -C, --config=filename         Read a configuration file, which can specify
                                  any of these options
                                  (default='/etc/zmap/zmap.conf')
  -T, --sender-threads=n        Threads used to send packets  (default='1')
      --cores=STRING            Comma-separated list of cores to pin to
      --ignore-invalid-hosts    Ignore invalid hosts in whitelist/blacklist
                                  file
  -h, --help                    Print help and exit
  -V, --version                 Print version and exit
Examples:
    zmap -p 80 -o - (scan the Internet for hosts on port 80 and output to
stdout)
    zmap -N 5 -B 10M -p 80 -o -  (find 5 HTTP servers, scanning at 10 Mb/s)
    zmap -p 80 10.0.0.0/8 192.168.0.0/16 -o (scan 10./8, 192.168./16 on port 80)
    zmap -p 80 192.168.1.2 192.168.1.3 (scan 192.168.1.2, 192.168.1.3 on port 80)
Probe-module (tcp_synscan) Help:
Probe module that sends a TCP SYN packet to a specific port. Possible
classifications are: synack and rst. A SYN-ACK packet is considered a success
and a reset packet is considered a failed response.
Output-module (csv) Help:
no help text available
```

A.2.2 ipscan

```
$ java -jar bin/ipscan-linux64-3.3.2.jar -h
Unknown option: h

Pass the following arguments:
[options] <feeder> <exporter>
Where <feeder> is one of:
-f:range <Start IP> <End IP>
-f:random <Base IP> <IP Mask> <Count>
-f:file <File>
<exporter> is one of:
-o filename.txt         Text file (txt)
-o filename.csv         Comma-separated file (csv)
-o filename.xml         XML file (xml)
-o filename.lst         IP:Port list (lst)
And possible [options] are (grouping allowed):
-s      start scanning automatically
-q      quit after exporting the results
-a      append to the file, do not overwrite
```

A.2.3 masscan

```
$ bin/masscan64
usage:
masscan -p80,8000-8100 10.0.0.0/8 --rate=10000
 scan some web ports on 10.x.x.x at 10kpps
masscan --nmap
 list those options that are compatible with nmap
masscan -p80 10.0.0.0/8 --banners -oB <filename>
 save results of scan in binary format to <filename>
masscan --open --banners --readscan <filename> -oX <savefile>
 read binary scan results in <filename> and save them as xml in <savefile>

$ bin/masscan64 -h
usage:
masscan -p80,8000-8100 10.0.0.0/8 --rate=10000
 scan some web ports on 10.x.x.x at 10kpps
masscan --nmap
 list those options that are compatible with nmap
masscan -p80 10.0.0.0/8 --banners -oB <filename>
 save results of scan in binary format to <filename>
masscan --open --banners --readscan <filename> -oX <savefile>
 read binary scan results in <filename> and save them as xml in <savefile>
gerhard@nbl350:/net/pub/ASE/2015SS/MasterThesis/Software/keyservice$ bin/masscan64 --help
MASSCAN is a fast port scanner. The primary input parameters are the
IP addresses/ranges you want to scan, and the port numbers. An example
is the following, which scans the 10.x.x.x network for web servers:
 masscan 10.0.0.0/8 -p80
The program auto-detects network interface/adapter settings. If this
fails, you'll have to set these manually. The following is an
example of all the parameters that are needed:
 --adapter-ip 192.168.10.123
 --adapter-mac 00-11-22-33-44-55
 --router-mac 66-55-44-33-22-11
Parameters can be set either via the command-line or config-file. The
names are the same for both. Thus, the above adapter settings would
appear as follows in a configuration file:
 adapter-ip = 192.168.10.123
 adapter-mac = 00-11-22-33-44-55
 router-mac = 66-55-44-33-22-11
All single-dash parameters have a spelled out double-dash equivalent,
so '-p80' is the same as '--ports 80' (or 'ports = 80' in config file).
To use the config file, type:
 masscan -c <filename>
To generate a config-file from the current settings, use the --echo
option. This stops the program from actually running, and just echoes
the current configuration instead. This is a useful way to generate
your first config file, or see a list of parameters you didn't know
about. I suggest you try it now:
 masscan -p1234 --echo

$ bin/masscan64 --nmap
Masscan (https://github.com/robertdavidgraham/masscan)
Usage: masscan [Options] -p{Target-Ports} {Target-IP-Ranges}
TARGET SPECIFICATION:
  Can pass only IPv4 address, CIDR networks, or ranges (non-nmap style)
  Ex: 10.0.0.0/8, 192.168.0.1, 10.0.0.1-10.0.0.254
  -iL <inputfilename>: Input from list of hosts/networks
  --exclude <host1[,host2][,host3],...>: Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list from file
  --randomize-hosts: Randomize order of hosts (default)
HOST DISCOVERY:
  -Pn: Treat all hosts as online (default)
  -n: Never do DNS resolution (default)
SCAN TECHNIQUES:
  -sS: TCP SYN (always on, default)
SERVICE/VERSION DETECTION:
  --banners: get the banners of the listening service if available. The
    default timeout for waiting to recieve data is 30 seconds.
PORT SPECIFICATION AND SCAN ORDER:
  -p <port ranges>: Only scan specified ports
    Ex: -p22; -p1-65535; -p 111,137,80,139,8080
TIMING AND PERFORMANCE:
  --max-rate <number>: Send packets no faster than <number> per second
  --connection-timeout <number>: time in seconds a TCP connection will
    timeout while waiting for banner data from a port.
FIREWALL/IDS EVASION AND SPOOFING:
  -S/--source-ip <IP_Address>: Spoof source address
  -e <iface>: Use specified interface
  -g/--source-port <portnum>: Use given port number
  --ttl <val>: Set IP time-to-live field
  --spoof-mac <mac address/prefix/vendor name>: Spoof your MAC address
OUTPUT:
  --output-format <format>: Sets output to binary/list/unicornscan/json/grepable/xml
  --output-file <file>: Write scan results to file. If --output-format is
     not given default is xml
  -oL/-oJ/-oG/-oB/-oX/-oU <file>: Output scan in List/JSON/Grepable/Binary/XML/
      Unicornscan format,
     respectively, to the given filename. Shortcut for
     --output-format <format> --output-file <file>
  -v: Increase verbosity level (use -vv or more for greater effect)
  -d: Increase debugging level (use -dd or more for greater effect)
  --open: Only show open (or possibly open) ports
  --packet-trace: Show all packets sent and received
  --iflist: Print host interfaces and routes (for debugging)
  --append-output: Append to rather than clobber specified output files
  --resume <filename>: Resume an aborted scan
MISC:
  --send-eth: Send using raw ethernet frames (default)
  -V: Print version number
  -h: Print this help summary page.
EXAMPLES:
  masscan -v -sS 192.168.0.0/16 10.0.0.0/8 -p 80
  masscan 23.0.0.0/0 -p80 --banners -output-format binary --output-filename internet.scan
  masscan --open --banners --readscan internet.scan -oG internet_scan.grepable
SEE (https://github.com/robertdavidgraham/masscan) FOR MORE HELP
```

A.2.4 ipshuffle

```
$ bin/ipshuffle64 -h
IP Address Shuffle program 0.5
  FH-Joanneum Keyservice Project
  (c) Gerhard Reithofer, Jun 24 2015, 21:10:26
Usage: bin/ipshuffle64 [-h] [-l] [-r] [-d level] [-i seed] [-s start] [-n numrecs] [-f
    outfile]
  numrecs .. num of ip recs in hex (def. 40000000)
  start .... start index in hex (def. 0)
  outfile .. output file name (def. ipv4shuffle.dat)
If numrecs is set to 0 the complete v4 range is used
If option '-l' (linear table) is used, shuffling is ommited
If option '-r' (reversed) is used, the ip byte range is reversed
The option '-d' defines the level of debugging output
The option '-i' allows the definition of specific srand seed value
```

A.3 Command scripts (cmd)

These scripts allow to call some key management commands using “openssl” for testing and comparing the results of the automated execution (*.sh) and scanning steps for Level I scanning using the same interface for the tested scanners “ipscan”, “masscan” and “nmap”.

A.3.1 collect_pubkeys.sh

Extract the certificate data for a specific IP address and port in text format or as PEM file if option “pem” is specified.

```
$ cmd/collect_pubkeys.sh
Missing hostname or ssl port
Usage: collect_pubkeys.sh host port [pem]
```

A.3.2 extract_pubkey.sh

Extract the public key (in PEM format and openssl text) from an existing PKCS1 (refer to Jonsson and Kaliski (2003)) certificate.

```
$ cmd/extract_pubkey.sh
Missing cert file
Usage: extract_pubkey.sh pkcs1-file
```

A.3.3 ipscan, masscan and nmap

This scanner command line interface calls are identical for all scanners - masscan example:

```
$ cmd/masscan
Error: Missing input file name
Usage: masscan input_file output_file
```

The “input_file” contains the list of addresses and the scanning results are written to the “output_file”.

A.4 Certificate archive (crt)

The files are store in the format

```
crt/$1/$2/$3/$4/p$port.pem
```

whereas “$1. . . 4” are the octets of the IP address and $port specifies the port number of the found service.
Examples:

```
crt/107/170/182/95/p993.pem    IMAPS (port 993) on IP address 107.170.182.95
crt/63/88/74/193/p443.pem      HTTPS on 63.88.74.193
crt/74/204/174/114/p22.pem     SSH on 74.204.174.114
```

A.5 Log files (log)

A.5.1 basescan_all.log

```
etc/exclude.conf: excluding 32 ranges from file

Starting masscan 1.0.3 (http://bit.ly/14GZzcT) at 2016-05-22 21:27:08 GMT
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 3970008572 hosts [19 ports/host]
```

A.6 Plugin modules (mod)

The scanning plugin modules were created to have a common call interface to make it possible to use different scanner programs with the same scanner framework.
A scanner module must include at least the following functions:

preaction(addrfile,addrlist) program code which is executed before the scanner call, “addrfile” is the file where the addresses from “addrlist” are written to. The number of written addresses must be returned.

execscan(addrfile,outfile) execution of the binary scanner with addresses from “addrfile” and writing the scanning output to “outfile”. The number of entries of the scanning step must be returned.

postaction(outfile) program code which is executed after finishing the scanner step, the post-processing of the “outfile”. The list of the found addresses must be returned.

A.6.1 ipscan.tcl, masscan.tcl

Scanning plugin modules for the Java scanner “ipscan”and for scanner “masscan”.

A.7 Data directory (dat)

Various input and output data like the Linear Address Table files (table data *.dat, index pointer *.seq), RSA key input lists for factoring step (*.mpz), etc.

```
Xipv4shuffle_20150517.dat
Xipv4shuffle_20150517.seq
ipv4rlinear_20150624.dat
ipv4rlinear_20150624.seq
uniq_20151120_1024d64.mpz
uniq_20151120_2048d64.mpz
uniq_20151120_2112hi.mpz
uniq_20151120_512low.mpz
uniq_20151120_all.mpz
```

A.8 Configuration files (etc)

A.8.1 exclude.conf

"masscan" scanner exclude list (see also 8.4 on page 84).

```
# http://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.
    xhtml
# http://tools.ietf.org/html/rfc5735
# "This" network
0.0.0.0/8
# Private networks
10.0.0.0/8
# Carrier-grade NAT - RFC 6598
100.64.0.0/10
# Host loopback
127.0.0.0/8
# Link local
169.254.0.0/16
# Private networks
172.16.0.0/12
# IETF Protocol Assignments
192.0.0.0/24
# DS-Lite
192.0.0.0/29
# NAT64
192.0.0.170/32
# DNS64
192.0.0.171/32
# Documentation (TEST-NET-1)
192.0.2.0/24
# 6to4 Relay Anycast
192.88.99.0/24
# Private networks
192.168.0.0/16
# Benchmarking
198.18.0.0/15
# Documentation (TEST-NET-2)
198.51.100.0/24
# Documentation (TEST-NET-3)
203.0.113.0/24
# Reserved
240.0.0.0/4
# Limited Broadcast
255.255.255.255/32

#Received: from elbmasnwh002.us-ct-eb01.gdeb.com ([153.11.13.41]
#Robert Mandes
#Information Security Officer
#General Dynamics
#Electric Boat
#
#C 860-625-0605
#P 860-433-1553
153.11.0.0/16

#Received: from [165.160.9.58] (HELO mx2.cscinfo.com)
#From: "Derksen, Bill" <bderksen@cscinfo.com>
#Subject: Unauthorized Scanning
#Corporation Service Company
165.160.0.0/16
```

```
# Your IP was observed making connections to TCP/IP IP address 149.93.26.231
# (a conficker sinkhole) with a destination port 80, source port (for this detection)
149.93.26.231

# Von: infosec-noreply@caltech.edu [mailto:infosec-noreply@caltech.edu]
# Gesendet: Donnerstag, 25. Dezember 2014 22:46
# An: abuse@a1telekom.at
# Hetzner-Support
88.198.0.0/16
136.243.0.0/16
138.201.0.0/16
148.251.0.0/16
185.12.64.0/22
213.133.96.0/19
213.239.192.0/18

# Abuse Message 1C4CAC21.txt
#
# ############################################################################
# # Netscan detected from host 94.101.38.222                                 #
# ############################################################################
78.46.0.0/15

# Subject: [NicBr-20150629-137] TCP PORT SCAN 94.101.38.222 -> xxx.xxx.2.0/23
# Date: Mon, 29 Jun 2015 05:27:47 -0300 (BRT)
# From: abuse@registro.br
# Igor Rigonato
200.160.0.0/20

# Das klärt einiges auf. Ich würde mich sehr
# freuen, wenn Sie unsere Adressbereiche 134.94.0.0/16 und 194.95.187.0/24 von
# den Scans ausnehmen.
# Egon Grünter, FZJ-CERT
134.94.0.0/16
194.95.187.0/24

# alfahostig
# "2015-07-08 00:12:28" 94.101.38.222/51763->xxx.xxx.5.34/993 6(0)
94.101.38.0 - 94.101.39.255

# Betreff: [noreply] abuse report about 185.3.232.72 - Mon, 23 May 2016 02:13:35 +0200
#          -- service: portflood (First x 1) RID: 743769679
# Datum: Mon, 23 May 2016 02:13:44 +0200 (CEST)
# Von: Abuse-Team (auto-generated)
#
# Email an support@blocklist.de mit Erklärung der Situation.
# Die IP-Adresse 5.196.200.157 wird aus weiteren Scans Scan ausgenommen.
5.196.200.157

# Subject: 185.3.232.72 blocked at caltech.edu
# Date: Sun, 22 May 2016 18:02:14 -0700
# From: infosec-noreply@caltech.edu
# Die IP-Adresse 66.148.70.33 wird aus weiteren Scans Scan ausgenommen.
# Der Adressbereich wurde auf 66.148.64.0/18 ausgeweitet
66.148.70.0/18

# Subject: Network abuse from 185.3.232.72
# Date: Mon, 23 May 2016 23:27:09 +0000
# From: Rodney Campbell <Rodney.Campbell@citec.com.au>
203.9.184.0 - 203.9.187.255
131.242.0.0 - 131.242.255.255
```

```
# Subject: 185.3.232.72: possible malicious activity from this host
# Subject: [USU Attack Report #31564] Malicious / Suspicious Activity - IP/CIDR Address:
    185.3.232.72
# Date: Mon, 23 May 2016 16:15:17 -0600
# From: Miles Johnson via RT <security@usu.edu>
129.123.0.0/16
144.39.0.0/16
204.113.91.0/24

# Subject: Hacking Activity from 185.3.232.72
# Date: Tue, 24 May 2016 11:49:17 -0300 (BRT)
# From: mtso@matera.com
201.16.192/18

# Hi Gerhard,
# Thanks for the detail response. I'll add 185.3.232.72 to our whitelist.
#
# Regards,
# --
# Juan Gallego (abuse@physics.mcgill.ca)        Department of Physics
### 143.106.0.0/16

# Subject: Loginattempts from Your net
# Date: Wed, 25 May 2016 12:12:44 +0200 (CEST)
# From: elwood@agouros.de
194.77.40.240/29
```

A.8.2 masscan.cfg

"masscan" scanner configuration file - example:

```
rate = 1000000000
excludefile = etc/exclude.conf
randomize-hosts = false
banner = true
# ports = 22,443
ports = 22,443,465,563,636,989,990,991,992,993,994,995,2083,2087,2096,4031,8140,8443,9050
output-status = open
randomize-hosts = false
```

A.8.3 file_db.cfg

Configuration file for Randomized Linear Address Table.

```
#
# FileDB configuration file
#
##################################################
# flat file database
FILE_DB DATA_DIR ./dat
FILE_DB DATABASE ipv4shuffle_20150624
##################################################
#
# MPZ export file
FILE_DB EXPORT KEYFILE ./dat/export.mpz
```

A.8.4 scan_db.cfg

Various configuration parameters used by different modules.

```
##################################################
#
#  Database spectific settings for saving
#
###
# database server
# SCANNER DATABASE SERVER nserver
SCANNER DATABASE SERVER localhost
# database name
SCANNER DATABASE DBASE keyservice
# SCANNER DATABASE DBASE newservice
# database username
SCANNER DATABASE USERNAME keymaster
# database username
SCANNER DATABASE PASSWORD secr-pa$$w0rd
###################################################
#
#  ports to check for SSL service
#
# grep SS /etc/services | \
#   sed 's:/tcp::'|awk '{print $2,$1}'|sort -n| \
#   awk '{printf(" %s %s", $1, $2)}END{print ""}'
#
# REMOTE SSL_PORTS { ssh 22 https 443 ssmtp 465 nntps 563 ldaps 636
#   ftps-data 989 telnets 992 imaps 993 ircs 994 pop3s 995 suucp 4031}
# }
# 22,443,465,563,636,989,992,993,994,995,4031
#
# proto-banner1.c
# 443;   /* HTTP/s */
# 465;   /* SMTP/s */
# 990;   /* FTP/s */
# 991;
# 992;   /* Telnet/s */
# 993;   /* IMAP4/s */
# 994;
# 995;   /* POP3/s */
# 2083;  /* cPanel - SSL */
# 2087;  /* WHM - SSL */
# 2096;  /* cPanel webmail - SSL */
# 8140;  /* puppet */
# 8443;  /* Plesk Control Panel - SSL */
# 9050;  /* Tor */
# REMOTE SSL_PORTS { ssh 22 https 443 ssmtp 465 nntps 563 ldaps 636
#   ftps-data 989 unkn1 991 telnets 992 imaps 993 ircs 994 pop3s 995
#   cPanel 2083 WHM 2086 suucp 4031 puppet 8140 Plesk 8443 9050 Tor}
# }
#
# 22,443,465,563,636,989,990,991,992,993,994,995,2083,2087,2096,4031,8140,8443,9050
#
# max. 7 entries, flag 0xffff is reserved for "ignored"
#
SCANNER SSL_PORTS {
  https     443
  ssmtp     465
  nntps     563
  ldaps     636
  ftps-data 989
  telnets   992
  imaps     993
  ircs      994
  pop3s    4031
  cPanel   2083
  WHM      2086
  suucp    4031
  puppet   8140
  Plesk    8443
  ssh        22
}
#
# 22,443,465,636,993,994,4031
#
####################################################
#
SCANNER SCAN_MODULE masscan
# SCANNER NET_DEVICE eth0
# SCANNER PORT_NUMBER 443
# SCANNER PORT_NUMBER 22,4031,465,636,989,994,8443,993,8140,993
SCANNER PORT_NUMBER 22,443,993
# max. scanning level ...
SCANNER MAX_LEVEL 6
# default number of records to scan or export
SCANNER NUM_RECS 100000
# max. allowed timeout for TLS connection in millis
# SCANNER SOCKET_TIMEOUT 2000
SCANNER SOCKET_TIMEOUT 1000
```

A.9 TCL libraries and modules (lib)

The library directory contains mostly “tcllib” source libraries (https://core.tcl.tk/tcllib/doc/trunk/-embedded/www/toc.html). These are typically direct installable in most Linux distributions, but maybe not available on all platforms as installable packages.

```
lib/aes          AES encryption routines(tcllib)
lib/asn          ASN1 parsing library(tcllib)
lib/base64       Base64 en/decoding (tcllib)
lib/csv          CSV module (tcllib, for statistics)
lib/des          DES en/decryption module (tcllib)
lib/dns          DNS library (tcllib)
lib/emu          Graphical chart library (unused?)
lib/fsdialog     "Beauty" portable file selector in TCL
lib/json         JSON library (tcllib)
lib/l-c          TCL-TLS source code and binary
lib/math         Math modules (tcllib, for statistics)
lib/md5          MD5 module (tcllib)
lib/pki          PKI module (tcllib, incomplete)
lib/sha1         SHA1 library (tcllib)
```

In addition the following package directories are present:

```
lib/pgintcl-3.5.0/
```

Special TCL only portable PostgreSQL interface. This is a fall-back solution for environments where the binary module “libpgtcl” is not available.

```
lib/Plotchart2.3.5/
```

Standard “plotchart” module from tklib (https://core.tcl.tk/tklib/doc/trunk/embedded/www/toc.html) but with some own extensions. This module is only used to create charts for this thesis, it is not necessary for scanning process itself.

A.9.1 TLS C-library directory (lib/l-c)

This directory contains the source code and the binary module for the extended TLS module for TCL and the source for ipshuffle.c (see A.2.4).

A.9.2 Common user libraries (lib/clib)

In this directory the common project specific user library modules are located.

```
AppUtil.tcl   Application config handling and debug-tools
basetools.tcl Common used functions like FileIO, conversions, etc.
file_db.tcl   File database module - Linear Address Table functions
get_cert.tcl  Retrieve certificate module
pg_intf.tcl   Database interface functions - usable with Pgtcl and pgintcl
pkgIndex.tcl  Package loader file
scan_db.tcl   Scanning database interface
whois.tcl     Whois implementation (experimental)
```

A.10 Management shell scripts (scr)

A.10.1 Job controlling commands:

A.10.1.1 rescan_seqjob.sh

Automatically rescan of unfinished scan jobs.

A.10.1.2 manage_jobs.sh

```
$  scr/manage_jobs.sh -h
Usage: manage_jobs.sh [-h] [-x] [job_name [..]]
 -h .. show this help
 -l .. return all open jobs as single list
 -x .. reexecute all open missing jobs
 Given jobs will be reexecuted again
```

Level I scan job management script (calls “save_scan.tcl” and “rescan_seqjob.sh”).

A.10.2 Monitoring scripts:

A.10.2.1 monitor_procs.sh

Concurrent process monitoring.

```
$ scr/monitor_procs.sh
Missing task name to monitor
Usage: monitor_procs.sh taskname [sleeptime]
  Current sleeptime is 10
```

A.10.2.2 log_dbinserts.sh

Database insert record monitoring.

```
$ scr/log_dbinserts.sh
Usage: log_dbinserts.sh [seq|rsa|upd] [waittime]
  seq: table scan, sequence scan
  rsa: table rsa_certs, sequence scan
  upd: table rsa_certs, sequence rsa
  Database: keyservice
  Waittime: 60 seconds
```

A.10.3 Parallel scanning:

A.10.3.1 par_scan_seq.sh

Parallel Level I scan controller manages scan script "scan_seq.tcl" (see section A.1.7), destination table is "scan".

A.10.3.2 par_scan_rsa.sh

Parallel Level II scan controller manages script "retrieve_certs.tcl" (see section A.1.4), destination table is "rsa_certs".

A.10.3.3 monitored_seq.sh

```
$ scr/monitored_seq.sh
Usage: monitored_seq.sh [start|stop]
```

Starts and stops "par_scan_seq.sh" (A.10.3.1), "monitor_procs.sh" (A.10.2.1) and "log_dbinserts.sh" (A.10.2.2).

A.10.3.4 monitored_rsa.sh

```
$ scr/monitored_rsa.sh
Usage: monitored_rsa.sh [start|stop|status]
```

Starts and stops "par_scan_rsa.sh"(A.10.3.2), "monitor_procs.sh" (A.10.2.1) and "log_dbinserts.sh" (A.10.2.2).

A.10.3.5 Listing of monitored_rsa.sh

```
#!/bin/bash
#

# Starting of
#   1. par_scan_rsa.sh (retrieve_cert)
#   2. monitor_procs.sh (retrieve_cert)
#   3. log_dbinserts.sh (retrieve_cert)
#

out()
{ # $* text output
  echo "$(date +%Y-%m-%d/%H:%M:%S): $*"
}
```

```
WAITTIME=60
TIMESTMP=$(date +%Y-%m-%d)
CURR_DIR=$(dirname $0)
cd $CURR_DIR/..
BASE_DIR=$PWD
RLOG_PRE=$BASE_DIR/log/retrieve_cert_$TIMESTMP
SLOG_PRE=$BASE_DIR/log/retrieve_stop_$TIMESTMP
REXECUTE=retrieve_cert.tcl
cd $BASE_DIR/scr/dir_rsa

case "$1" in
  start )
    nohup $BASE_DIR/scr/par_scan_rsa.sh 1>>$RLOG_PRE.slog 2>>$RLOG_PRE.elog &
    nohup $BASE_DIR/scr/monitor_procs.sh $REXECUTE $WAITTIME &
    nohup $BASE_DIR/scr/log_dbinserts.sh rsa $WAITTIME &
    ;;
  stop )
    for PNAME in par_scan_rsa.sh monitor_procs.sh log_dbinserts.sh
    do PROC_ID=$(ps -ef|grep $PNAME|grep -vw grep|awk '{print $2}')
       if [ ! -z "$PROC_ID" ]
       then out "Killing $PROC_ID($PNAME) ..."
            kill $PROC_ID
       else out "Process $PNAME not running."
       fi
    done

    NUM_PROCS=$(ps -ef|grep $REXECUTE|grep -vw grep|wc -l)
    START=$(date +%s)
    while [ $NUM_PROCS -gt 0 ]
    do out "$NUM_PROCS $REXECUTE processes still running ..."
       sleep 5
       NUM_PROCS=$(ps -ef|grep $REXECUTE|grep -vw grep|wc -l)
    done
    ENDP=$(date +%s)
    DIFF=$(expr $ENDP - $START)
    out "all $REXECUTE processes finished after $DIFF seconds."
    ;;
  status )
    for PNAME in par_scan_rsa.sh monitor_procs.sh log_dbinserts.sh
    do PROC_ID=$(ps -ef|grep $PNAME|grep -vw grep|awk '{print $2}')
       if [ ! -z "$PROC_ID" ]
       then out "Running $PROC_ID($PNAME) ..."
       else out "Process $PNAME not running."
       fi
    done
    NUM_PROCS=$(ps -ef|grep $REXECUTE|grep -vw grep|wc -l)
    out "$NUM_PROCS $REXECUTE processes running ..."|tee -a $SLOG_PRE.log
    ;;
   *   )
    echo "Usage: $(basename $0) [start|stop|status]"
    ;;
esac
```

A.10.4 Project phase 2 “basescan” commands:

A.10.4.1 basescan_all.sh

Starts “masscan” executable for hi-speed complete IPV4 range.

```
#!/bin/sh
#

BASE_DIR=$(dirname $0)/..
LOGF_DIR=$BASE_DIR/log
CFG_FILE=$BASE_DIR/etc/masscan.cfg
LOG_FILE=$LOGF_DIR/basescan_all.log
ERR_FILE=$LOGF_DIR/basescan_all.err
OUT_FILE=$BASE_DIR/tmp/basescan_all.out
IP_RANGE="0.0.0.0/0"

echo $BASE_DIR/cmd/masscan -c $CFG_FILE -oL $OUT_FILE $IP_RANGE
     $BASE_DIR/cmd/masscan -c $CFG_FILE -oL $OUT_FILE $IP_RANGE 1>$LOG_FILE 2>$ERR_FILE &
```

A.10.4.2 basescan_imp.sh

Import masscan result files by calling “save_scan.tcl”(section A.1.6).

A.10.4.3 par_import.sh

Parallel import controller for join databases.

A.10.4.4 monitor_base.sh

“basescan” monitoring script (counting lines in “basescan_all.out”).

A.10.4.5 import_log.sh

Database based monitoring log file

```
select count(*) from $TABLE;
```

using “scan” or “rsa_certs” as TABLE, an optional wait loop time can be specified on command line.

A.11 SQL scripts (sql)

Installation script for the keyservice database.

keyservice_inst.sql	SQL dazabse install script (B.1.4)

Schema files retrieved from PostgreSQL database dumps.

keyservice-schema.sql	Complete keyservice schema export
rsa_certs-schema.sql	Schema export for table rsa_certs
scan-schema.sql	Schema export for table scan

A.12 Various utilities (utl)

```
backup.sh          - backup script -> bak
calc_speed.tcl     - calc sanning rate from masscan output files (G.1.2)
dist_calc.sh       - calculate "IP distance" from hetzner IP list
dist_check.tcl     - check "IP distance" from "Randomized Linear Address Table"
dist_create.tcl    - extract "hetzner" IP list from file_db
dist_hetzner.sh    - calculate "IP distance" from hetzner log files
distinct.sh        - extract top ten hashes from all_hashes.txt file
export_all.sh      - export all uniq certs (<512,1024+/-512,2048+/-512,>2560,all)
f_sequential.tcl   - file list serializer, joins listed files in dictionary order
join_crt_lists.sh  - cert file joiner, crt+old+err
join_rsa.tcl       - join rsa_certs from DB new+old (A.12.1)
mark_blocked.sh    - create SQL statement for blocking "scan" records (A.12.2)
rsa_convert.sh     - convert certificate file from/to (der|net|pem) using "openssl"
rsa_data_errors.sh - CSV extrcat RSA "data_insert" errors from scan
rsa_extract.sh     - extract RSA infos from certificate file
split_bigfile.sh   - script to split basescan_all.out to part files
select2CSV.sh      - SQL select into CSV file (A.12.3)
dnsync.sh          - rsync from server to local WS
upsync.sh          - rsync from local WS to server (A.12.4)
dsc.sh             - single file down copy using scp (A.12.6)
usc.sh             - single file up copy using scp (A.12.5)
sc                 - show certificate - display saved certificate or file path
```

A.12.1 join_rsa.tcl

```
$ utl/join_rsa.tcl -h
Scan table join utility 0.1
Usage: utl/join_rsa.tcl [-h] [-s start] [-n count]
 -h .. this help text
 -s .. set start value (rsa_certs id) for next block (22577893)
 -n .. set number of rsa_cert records for joining (100000)
```

A.12.2 mark_blocked.sh

A script which analyzes the masscan exclude list (refer to section A.8.1) and creating the corresponding SQL statements to mark all related IP addresses in the table “scan” as “blocked” (refer to section 7.3.1.1).

A.12.3 select2CSV.sh

```
$ utl/select2CSV.sh -h
Usage: select2CSV.sh [-h] [-d database] [-s sep] [-o oufile] query ..
 -h .. this help text
 -d .. name of database, (def. keyservice)
 -s .. specifies the record separator for CSV-file (def. ';')
 -o .. name of the output CSV-file, (def. stdout)
```

Database evaluation module with CSV file output.

A.12.4 upsync.sh

```
$ utl/upsync.sh
Missing sync destination, valid: LowCost MemPower FHCluster
```

RSync (using SSH) script for syncing a defined list of directories to the other nodes. “LowCost” (EC2), “MemPower” (provider) and “FHCluster” (FH-Archive) are hard coded server aliases in the script.

A.12.5 usc.sh

Up Secure Copy command from the local machine to the provider machine.

```
$ utl/usc.sh
Usage: usc.sh from(local) to(remote)
  copies local file to remote directory
  remote host: d10122.dedicated.alfahosting-pro.de
  remote user: keymaster
  remote base: keyservice/
```

A.12.6 dsc.sh

Down Secure Copy command from the provider machine to the local machine.

```
$ utl/dsc.sh
Usage: dsc.sh from(remote) to(local)
  copies remote file to local directory
  remote host: d10122.dedicated.alfahosting-pro.de
  remote user: keymaster
  remote base: keyservice/
```

A.13 Web service files (web)

HTML index, CSS files and PHP scripts.

```
main.css      - CSS stylesheet
index.html    - Project information and keyservice page
checkkey.php  - Script for checking a single public RSA key
download.php  - Script for downloading the results as ASCII file
include.php   - Common used functions and tools
```

Appendix B

Database related

B.1 Database objects

```
              List of relations
 Schema |        Name        |   Type   |   Owner
--------+--------------------+----------+-----------
 public | parameters         | table    | keymaster
 public | parameters_id_seq  | sequence | keymaster
 public | rsa_certs          | table    | keymaster
 public | rsa_certs_id_seq   | sequence | keymaster
 public | scan               | table    | keymaster
 public | scan_id_seq        | sequence | keymaster
 public | seq_table          | table    | keymaster
```

B.1.1 Table SCAN

Level I scanning results.

```
                                         Table "public.scan"

     Column      |            Type             |           Modifiers
-----------------+-----------------------------+----------------------------------
 id              | bigint                      | not null default nextval('scan_id_seq'::
    regclass)
 ip_address      | inet                        | not null
 service         | integer                     | not null
 level           | integer                     | default 1
 certificate_id  | bigint                      |
 keyinfo         | character varying           |
 keydata         | bytea                       |
 seen_at         | timestamp without time zone | not null default now()
 flag            | character varying           |
 status          | character varying           | not null default ''::character varying
Indexes:
    "scan_pkey" PRIMARY KEY, btree (id)
    "u_a_s_s" UNIQUE CONSTRAINT, btree (ip_address, service)
    "scan_ip_address_service_idx" btree (ip_address, service)
```

B.1.2 Table RSA_CERTS

Level II RSA certificate data.

```
                                           Table "public.rsa_certs"
      Column      |            Type             |          Modifiers
------------------+-----------------------------+--------------------------------------
 id               | bigint                      | not null default nextval('
    rsa_certs_id_seq'::regclass)
 last_mod         | timestamp without time zone | not null default now()
 ip_address       | inet                        | not null
 service          | integer                     | not null
 level            | integer                     | not null default 1
 status           | character varying           |
 sha1_hash        | character varying           | not null
 serial_number    | character varying           | not null
 version          | integer                     |
 subject          | character varying           | not null
 issuer           | character varying           | not null
 rsa_e            | bigint                      |
 rsa_n            | bytea                       |
 rsa_d            | bigint                      |
 rsa_p            | bytea                       |
 rsa_q            | bytea                       |
 sbits            | integer                     | not null
 not_valid_before | timestamp without time zone | not null
 not_valid_after  | timestamp without time zone | not null
 cipher           | character varying           | not null
 chain            | character varying           |
 key_type         | character varying           |
 key_size         | smallint                    |
Indexes:
    "rsa_certs_pkey" PRIMARY KEY, btree (id)
    "u_a_s_r" UNIQUE CONSTRAINT, btree (ip_address, service, sha1_hash)
    "rsa_certs_ip_address_service_idx" btree (ip_address, service)
    "rsa_certs_sha1_hash_idx" btree (sha1_hash)
    "rsa_n_md5_index" btree (md5(rsa_n))
```

B.1.3 Table SEQ_TABLE

User defined sequence counters for Level I, Level II, Join tables and Export functionality.

B.1.3.1 Table schema

```
            Table "public.seq_table"
 Column |       Type        |     Modifiers
--------+-------------------+--------------------
 name   | character varying | not null
 value  | bigint            | not null default 0
Indexes:
    "seq_table_name_key" UNIQUE CONSTRAINT, btree (name)
```

B.1.3.2 Table content - example seq_table

```
     name      |  value
---------------+----------
 next_scanval |       17
 next_export  |        0
 next_rsa_val |    40003
 next_joinval | 22577892
```

B.1.4 SQL Installation script (keyservice_inst.sql)

```
CREATE TABLE scan  (
    id             bigserial NOT NULL  PRIMARY KEY,
    ip_address     inet      NOT NULL,
    service        integer   NOT NULL,
    level          integer   DEFAULT   1,             /* scanning level */
    certificate_id bigint    NULL,                    /* only the last ... */
    keyinfo        varchar   NULL,                    /* SSL/TLS banner */
    keydata        bytea     NULL,                    /* cert from scan level 1 job */
    seen_at        timestamp NOT NULL  DEFAULT now(),
    flag           varchar   NULL,                    /* banner type (ssl, X509, ...) */
    status         varchar   NOT NULL  DEFAULT ''     /* handshake result */
);
alter table scan add constraint "u_a_s_s" unique(ip_address, service);
create index on scan (ip_address, service);
create index on scan (ip_address, service, status);

CREATE TABLE rsa_certs  (
    id               bigserial  NOT NULL        PRIMARY KEY,
    last_mod         timestamp  NOT NULL        DEFAULT now(),
    ip_address       inet       NOT NULL,
    service          integer    NOT NULL,
    level            integer    NOT NULL        DEFAULT 1
    status           varchar,
    sha1_hash        varchar    NOT NULL,
    serial_number    varchar    NOT NULL,
    version          integer,
    subject          varchar    NOT NULL,
    issuer           varchar    NOT NULL,
    rsa_e            bigint,  /* in RSA it is bignum */
    rsa_n            bytea,
    rsa_d            bigint,  /* in RSA it is bignum */
    rsa_             bytea,
    rsa_q            bytea,
    sbits            integer    NOT NULL,
    not_valid_before timestamp  NOT NULL,
    not_valid_after  timestamp  NOT NULL,
    cipher           varchar    NOT NULL,
    chain            varchar,
    key_type         varchar,
    key_size         smallint
);
create index on rsa_certs (ip_address, service);
alter table rsa_certs add constraint "u_a_s_r" unique(ip_address, service, sha1_hash);
-- special index on rsa_n due the following limitation.
-- keyservice=# create index on rsa_certs(rsa_n);
-- ERROR:  index row size 4112 exceeds maximum 2712 for index "rsa_certs_rsa_n_idx"
-- TIP:  Values larger than 1/3 of a buffer page cannot be indexed.
-- Consider a function index of an MD5 hash of the value, or use full text indexing.
create index rsa_n_md5_index on rsa_certs(md5(rsa_n));
create table seq_table (
    name      varchar  NOT NULL UNIQUE,
    value     bigint   NOT NULL DEFAULT 0
);
insert into seq_table values ('next_scanval', 0);
insert into seq_table values ('next_rsa_val', 0);
insert into seq_table values ('next_export' , 0);
insert into seq_table values ('next_joinval', 0);
```

Appendix C

Data related

C.1 Scan collections

Internet-Wide Scan Data Repository - data collections (view date 2016-09-21):

Name	Port	Protocol	Destination	Last Scan
0-icmp-echo_request-full_ipv4		icmp	full ipv4	2016-09-16 23:13:09
21-ftp-banner-full_ipv4	21	ftp	full ipv4	2016-09-19 22:27:06
22-ssh-banner-full_ipv4	22	ssh	full ipv4	2016-09-14 18:41:32
23-telnet-banner-full_ipv4	23	telnet	full ipv4	2016-09-20 23:56:34
25-smtp-dhe_export-1%_sample_ipv4	25	smtp	1% sample ipv4	2016-09-20 14:14:51
25-smtp-starttls-alexa_top1mil	25	smtp	alexa top1mil	2016-09-20 12:31:25
53-dns-lookup-full_ipv4	53	dns	full ipv4	2016-09-18 23:21:49
80-http-get-alexa_top1mil	80	http	alexa top1mil	2016-09-20 12:44:22
80-http-get-full_ipv4	80	http	full ipv4	2016-09-05 23:35:02
102-s7-szl-full_ipv4	102	s7	full ipv4	2016-09-14 12:30:42
110-pop3-starttls-full_ipv4	110	pop3	full ipv4	2016-09-18 00:09:00
143-imap-starttls-full_ipv4	143	imap	full ipv4	2016-09-18 22:45:29
443-https-dhe-alexa_top1mil	443	https	alexa top1mil	2016-09-20 12:56:57
443-https-dhe-full_ipv4	443	https	full ipv4	2016-09-18 23:11:43
443-https-dhe_export-1%_sample_ipv4	443	https	1% sample ipv4	2016-09-21 04:14:52
443-https-dhe_export-alexa_top1mil	443	https	alexa top1mil	2016-09-20 11:38:20
443-https-dhe_export-full_ipv4	443	https	full ipv4	2016-09-15 22:32:34
443-https-heartbleed-alexa_top1mil	443	https	alexa top1mil	2016-09-20 14:23:53
443-https-heartbleed-full_ipv4	443	https	full ipv4	2016-09-21 00:35:30
443-https-rsa_export-1%_sample_ipv4	443	https	1% sample ipv4	2016-09-21 04:29:50
443-https-rsa_export-alexa_top1mil	443	https	alexa top1mil	2016-09-20 14:25:37
443-https-rsa_export-full_ipv4	443	https	full ipv4	2016-09-15 22:39:13
443-https-ssl_3-alexa_top1mil	443	https	alexa top1mil	2016-09-20 15:27:50
443-https-ssl_3-full_ipv4	443	https	full ipv4	2016-09-14 23:00:00
443-https-tls-alexa_top1mil	443	https	alexa top1mil	2016-09-20 10:49:15
443-https-tls-full_ipv4	443	https	full ipv4	2016-09-20 09:36:29
465-smtps-tls-full_ipv4	465	smtps	full ipv4	2016-09-20 22:32:47
502-modbus-device_id-full_ipv4	502	modbus	full ipv4	2016-09-18 07:56:09

993-imaps-dhe_export-full_ipv4	993	imaps	full ipv4	2016-09-21 04:14:50
993-imaps-tls-full_ipv4	993	imaps	full ipv4	2016-09-15 00:09:21
995-pop3s-tls-full_ipv4	995	pop3s	full ipv4	2016-09-17 00:10:25
1911-fox-device_id-full_ipv4	1911	fox	full ipv4	2016-09-19 11:58:34
7547-cwmp-get-full_ipv4	7547	cwmp	full ipv4	2016-09-14 22:41:17
20000-dnp3-status-full_ipv4	20000	dnp3	full ipv4	2016-09-17 12:28:20
47808-bacnet-device_id-full_ipv4	47808	bacnet	full ipv4	2016-09-16 11:58:32

C.2 Open ports list

Open ports vs. TLS certificates compare

Port	Open ports	Found certs	Certs %	Expected service
22	15753958	0	00.00	SSH
443	38529358	19714543	51.17	HTTP/s
465	5223677	2042323	39.10	SMTP/s
563	2170734	29891	01.38	NNTP/s
636	2255812	64122	02.84	LDAP/s
989	2033068	902	00.04	FTP/s-data
990	1993651	138522	06.95	FTP/s
991	1609847	832	00.05	
992	1607972	18218	01.13	Telnet/s
993	3656833	1615414	44.18	IMAP4/s
994	1320289	990	00.07	IRC/s
995	3327227	1443294	43.38	POP3/s
2083	2625086	610982	23.27	cPanel - SSL
2087	2687838	579922	21.58	WHM - SSL
2096	3055931	712072	23.30	cPanel webmail - SSL
4031	1644898	767	00.05	s/UUCP
8140	1776439	13173	00.74	Puppet
8443	3669432	764775	20.84	Plesk Control Panel - SSL
9050	2126644	11767	00.55	Tor
All	97068694	27762509	28.60	Summary values

C.3 Scan logfile examples and timing data

C.3.1 IPScan results

Level I parallel scanning example output (see subsection 5.1.3.2 on page 32):

```
[2048_104135]: scan job start: 2015-05-22/10:35:24
[2048_106183]: scan job start: 2015-05-22/10:35:25
[2048_108231]: scan job start: 2015-05-22/10:35:26
[2048_110279]: scan job start: 2015-05-22/10:35:27
[2048_104135]: scan job finished: 2015-05-22/10:36:14
[2048_104135]: approx. traffic: 3572964 bytes
[2048_104135]: 2048 IPs scanned in 00:00:50 (41.0/sec), 33 SSL related IPs (1.61%) found
    (0.66/sec)
```

```
[2048_106183]: scan job finished: 2015-05-22/10:36:18
[2048_106183]: approx. traffic: 3730304 bytes
[2048_106183]: 2048 IPs scanned in 00:00:53 (38.6/sec), 32 SSL related IPs (1.56%) found
    (0.60/sec)
[2048_110279]: scan job finished: 2015-05-22/10:36:20
[2048_110279]: approx. traffic: 3547280 bytes
[2048_110279]: 2048 IPs scanned in 00:00:53 (38.6/sec), 29 SSL related IPs (1.42%) found
    (0.55/sec)
[2048_108231]: scan job finished: 2015-05-22/10:36:24
[2048_108231]: approx. traffic: 3785842 bytes
[2048_108231]: 2048 IPs scanned in 00:00:58 (35.3/sec), 32 SSL related IPs (1.56%) found
    (0.55/sec)
```

C.3.2 Scan evaluation example

```
  count  |    to_char
---------+---------------
    6278 | 2015-08-21 00
 1080597 | 2015-08-21 01
  640336 | 2015-08-21 02
  182215 | 2015-08-21 10
  187726 | 2015-08-21 11
  200920 | 2015-08-21 12
  203778 | 2015-08-21 13
  204006 | 2015-08-21 14
  203868 | 2015-08-21 15
  133785 | 2015-08-21 16
   72331 | 2015-08-21 17
   70711 | 2015-08-21 18
   71464 | 2015-08-21 19
  170708 | 2015-08-21 20
  210001 | 2015-08-21 21
  208458 | 2015-08-21 22
  208484 | 2015-08-21 23
  211611 | 2015-08-22 00
  208490 | 2015-08-22 01
  134451 | 2015-08-22 02
   72955 | 2015-08-22 03
   75916 | 2015-08-22 04
   76730 | 2015-08-22 05
   75063 | 2015-08-22 06
   74068 | 2015-08-22 07
   76668 | 2015-08-22 08
   75299 | 2015-08-22 09
   60953 | 2015-08-22 10
  341297 | 2015-08-22 13
  498221 | 2015-08-22 14
  470607 | 2015-08-22 15
  410152 | 2015-08-22 16
  464801 | 2015-08-22 17
  459344 | 2015-08-22 18
  416040 | 2015-08-22 19
  362745 | 2015-08-22 20
```

C.3.3 Content of log/scan_20150822_10000___1_7306.log

```
Start: 2015-08-22_22:28:10 scan_20150822_10000_7306
Module masscan successfully loaded
Scan started: 0753800000_100000 2015-08-22/22:28:10
```

```
Pre-Action: 100000 records written to input file /home/keymaster/keyservice/tmp
    /0753800000_100000.txt
etc/exclude.conf: excluding 32 ranges from file
Starting masscan 1.0.3 (http://bit.ly/14GZzcT) at 2015-08-22 20:28:46 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 92251 hosts [9 ports/host]
rate: 15.63-kpps,  9.62% done,   0:00:56 remaining, found=58
rate: 15.07-kpps, 22.54% done,   0:00:40 remaining, found=180
rate: 36.99-kpps, 32.50% done,   0:00:40 remaining, found=271
rate: 19.88-kpps, 43.41% done,   0:00:32 remaining, found=357
rate: 16.46-kpps, 58.88% done,   0:00:17 remaining, found=467
rate: 31.95-kpps, 71.48% done,   0:00:13 remaining, found=592
rate: 15.34-kpps, 85.09% done,   0:00:07 remaining, found=745
rate: 12.98-kpps, 95.62% done,   0:00:02 remaining, found=842
rate:  0.02-kpps, 100.00% done, waiting 10-secs, found=897
rate:  0.02-kpps, 100.00% done, waiting 9-secs, found=897
rate:  0.00-kpps, 100.00% done, waiting 8-secs, found=897
rate:  0.07-kpps, 100.00% done, waiting 7-secs, found=898
rate:  0.00-kpps, 100.00% done, waiting 6-secs, found=898
rate:  0.02-kpps, 100.00% done, waiting 5-secs, found=898
rate:  0.02-kpps, 100.00% done, waiting 4-secs, found=898
rate:  0.02-kpps, 100.00% done, waiting 3-secs, found=898
rate:  0.03-kpps, 100.00% done, waiting 2-secs, found=899
rate:  0.02-kpps, 100.00% done, waiting 1-secs, found=899
rate:  0.00-kpps, 100.00% done, waiting 0-secs, found=899

Exec-Scan finished, 1 results written to /home/keymaster/keyservice/tmp/0753800000_100000
    .csv
Post-Action: 911 results found
[0753800000_100000]: 100000 IPs scanned in 00:01:42 (980.4/sec), 911 SSL related IPs
    (0.91% - 8.93/sec), unknown=1 ssl=4 ssh=4 open=899 db_update=2 db_
insert=777 complete=911 X509=3
Scan finished: 0753800000_100000 2015-08-22/22:29:52
```

C.3.4 File time data of involved files

```
Name: tmp/0753800000_100000.txt
filesize: 1424840 (1.36M)
modified: 2015-08-22 22:28:12

Name: tmp/0753800000_100000.csv
filesize: 36868 (36.0K)
modified: 2015-08-22 22:29:51

Name: log/scan_20150822_10000__1_7306.log
filesize: 12329 (12.0K)
modified: 2015-08-22 22:29:52
```

Appendix D

Final hardware setup

Feature	Value
CPU:	AMD Opteron 6212, 2 x 8 Core (2,6 GHz)
Harddisk:	2 x 2000 GB
RAM:	48 GB
Traffic:	Unlimited
NIC:	1000 Mbit/s
Bandwidth	100 Mbit/s (500 MBit/s in phase 2)
FTP-Backup:	1000 GB
SSH Access:	Yes
IPv4-Address:	1
Availabilty:	99,9 %

Appendix E

Certificate related

E.1 OpenSSL example output

```
$openssl s_client -connect $HOST:$PORT </dev/null 2>/dev/null|openssl x509 -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            35:ce:c0:a2:e8:7c:b6:15:f3:b4:04:41:ae:61:88:09
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=NL, ST=Noord-Holland, L=Amsterdam, O=TERENA, CN=TERENA SSL CA 2
        Validity
            Not Before: Apr 24 00:00:00 2015 GMT
            Not After : May 21 23:59:59 2018 GMT
        Subject: OU=Domain Control Validated, CN=www.fh-joanneum.at
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a8:61:77:f4:c3:6a:32:49:93:48:2a:95:30:b9:
                    bc:4c:39:db:26:75:48:e3:ac:c3:7e:b6:a1:6b:29:
                    d6:8c:76:82:b9:3e:21:5a:70:10:e6:84:da:85:f3:
                    a1:1e:aa:cd:b5:90:ed:a8:8c:87:33:b7:cb:4c:1b:
                    a8:f9:1e:96:ad:94:2b:05:92:0a:d8:5c:84:69:38:
                    1b:f7:ad:e4:be:fa:31:30:e2:0d:32:8f:0b:04:dc:
                    4b:25:80:e9:7c:54:90:f6:59:e3:b4:31:dc:a3:d4:
                    18:78:01:aa:55:25:22:70:01:14:63:5f:7d:ed:b0:
                    4b:50:73:e7:12:65:14:3d:21:62:19:64:f2:35:16:
                    27:15:99:a7:b5:93:9c:41:4a:9f:33:dd:5d:63:13:
                    2d:c4:ea:93:d0:d8:ac:5c:e3:70:fa:e3:89:1a:1b:
                    ce:3a:7a:7c:ac:57:e5:a9:c8:2a:43:5f:76:94:33:
                    b4:1d:b3:a3:87:50:70:ea:52:aa:6e:ac:72:f4:b0:
                    df:fa:2d:c2:ea:f8:97:38:e7:91:53:b1:48:8e:09:
                    6c:39:ec:af:cf:69:89:df:e8:3b:ee:c9:6f:d6:60:
                    c5:bc:7f:9a:12:2e:d8:43:a6:77:75:2c:7a:e6:00:
                    b4:a5:c1:da:7e:e7:d3:4f:19:7c:35:c3:25:f8:20:
                    1e:81
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:5B:D0:8A:1C:9A:32:5B:E0:B5:DD:96:54:1B:E1:86:28:B0:FD:B6:BD
            X509v3 Subject Key Identifier:
                17:80:89:DD:10:68:43:5A:B4:1C:85:82:15:67:CE:47:EC:46:8B:CC
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.6449.1.2.2.29
                Policy: 2.23.140.1.2.1
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://crl.usertrust.com/TERENASSLCA2.crl
            Authority Information Access:
                CA Issuers - URI:http://crt.usertrust.com/TERENASSLCA2.crt
                OCSP - URI:http://ocsp.usertrust.com
            X509v3 Subject Alternative Name:
                DNS:www.fh-joanneum.at
    Signature Algorithm: sha256WithRSAEncryption
         8b:c4:83:3b:80:83:bc:f0:5c:d3:7b:fb:43:8c:cb:43:67:65:
         42:a3:85:2f:1d:79:8c:11:75:29:63:41:10:7f:ec:42:4d:f4:
         0b:4e:01:a2:9a:9a:88:b2:5e:5f:da:da:00:4b:22:0d:f4:de:
         ba:1b:6e:8e:72:f5:3a:63:0a:81:52:3a:4b:5f:fe:09:8b:bb:
         6d:44:e2:9f:27:77:c0:fb:8f:9e:9c:5b:f6:7a:30:b7:c5:ad:
         59:5d:5b:df:95:ac:14:04:ad:f2:5d:8f:60:97:96:b3:5d:5d:
         71:db:4f:5b:17:0d:25:58:b3:60:d2:85:46:25:9b:2e:25:04:
         f5:ea:2d:80:ef:58:0b:9d:d0:5f:0f:2f:fe:5f:d5:04:8e:b5:
         58:55:89:18:85:a9:73:b8:7f:ce:40:ad:4e:8f:1b:f2:47:83:
         d8:d5:91:c1:cc:01:51:f1:9a:e9:e8:f6:9e:44:c3:2d:43:88:
         65:9f:98:2f:b7:55:b5:71:32:f4:62:52:61:07:ac:e7:5c:dc:
         98:60:a9:0b:b3:4e:26:b9:9c:7c:70:fb:ff:e4:4e:fa:17:34:
         f4:b4:27:da:0e:38:b6:cc:94:b5:13:90:bb:9f:b4:51:c6:3c:
         48:91:35:38:73:84:48:75:4f:ae:23:66:10:a3:df:75:d7:e5:
         79:8d:cb:4f
```

E.2 Certificate examples

E.2.1 Certificate in PEM format

```
$ openssl s_client -connect $HOST:$PORT </dev/null 2>/dev/null|openssl x509 -outform PEM
-----BEGIN CERTIFICATE-----
MIIEnjCCA4agAwIBAgIQNc7Aouh8thXztARBrmGICTANBgkqhkiG9w0BAQsFADBk
MQswCQYDVQQGEwJOTDEWMBQGA1UECBMNTm9vcmQtSG9sbGFuZDESMBAGA1UEBxMJ
QW1zdGVyZGFtMQ8wDQYDVQQKEwZURVJFTkExGDAWBgNVBAMTD1RFUkVOQSBTU0wg
Q0EgMjAeFw0xNTA0MjQwMDAwMDBaFw0xODA1MjEyMzU5NTlaMEAxITAfBgNVBAsT
GERvbWFpbiBDb250cm9sIFZhbGlkYXRlZDEbMBkGA1UEAxMSd3d3LmZoLWpvYW5u
ZXVtLmF0MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqGF39MNqMkmT
SCqVMLm8TDnbJnVI46zDfrahaynWjHaCuT4hWnAQ5oTahf0hHqrNtZDtqIyHM7fL
TBuo+R6WrZQrBZIK2FyEaTgb963kvvoxMOINMo8LBNxLJYDpfFSQ9lnjtDHco9QY
eAGqVSUicAEUY1997bBLUHPnEmUUPSFiGWTyNRYnFZmntZOcQUqfM91dYxMtxOqT
ONisXONw+uOJGhvOOnp8rFflqcgqQ192lDOOHbOjh1Bw6lKqbqxy9LDf+i3C6viX
OOeRU7FIjglsOeyvz2mJ3+g77slv1mDFvH+aEi7YQ6Z3dSx65gCOpcHafufTTxl8
NcMl+CAegQIDAQABo4IBbjCCAWowHwYDVR0jBBgwFoAUW9CKHJoyW+C13ZZUG+GG
KLD9tr0wHQYDVR0OBBYEFBeAid0QaENatByFghVnzkfsRovMMA4GA1UdDwEB/wQE
AwIFoDAMBgNVHRMBAf8EAjAAMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcD
AjAiBgNVHSAEGzAZMA0GCysGAQQBsjEBAgIdMAgGBmeBDAECATA6BgNVHR8EMzAx
MC+gLaArhilodHRwOi8vY3JsLnVzZXJ0cnVzdC5jb20vVEVSRU5BU1NMQ0EyLmNy
bDBsBggrBgEFBQcBAQRgMF4wNQYIKwYBBQUHMAKGKWh0dHA6Ly9jcnQudXNlcnRy
dXN0LmNvbS9URVJFTkFTU0xDQTIuY3J0MCUGCCsGAQUFBzABhhlodHRwOi8vb2Nz
cC51c2VydHJ1c3QuY29tMB0GA1UdEQQWMBSCEnd3dy5maC1qb2FubmV1bS5hdDAN
```

```
BgkqhkiG9w0BAQsFAAOCAQEAi8SDO4CDvPBc03v7Q4zLQ2dlQqOFLx15jBF1KWNB
EH/sQk30C04BopqaiLJeX9raAEsiDfTeuhtujnL1OmMKgVI6S1/+CYu7bUTinyd3
wPuPnpxb9nowt8WtWV1b35WsFASt8l2PYJeWs11dcdtPWxcNJVizYNKFRiWbLiUE
9eotgO9YC53QXw8v/l/VBI61WFWJGIWpc7h/zkCtTo8b8keD2NWRwcwBUfGa6ej2
nkTDLUOIZZ+YL7dVtXEy9GJSYQes51zcmGCpC7NOJrmcfHD7/+RO+hcO9LQn2g44
tsyUtROQu5+0UcY8SJE1OHOESHVPriNmEKPfddfleY3LTw==
-----END CERTIFICATE-----
```

E.2.2 Error certificate

Broken certificate found in service 146.247.90.133:443

```
-----BEGIN CERTIFICATE-----
MIIDqDCCAxGgAwIBAgIJAP/uCjzC9TuMMA0GCSqGSIb3DQEBBQUAMIGUMQswCQYD
VQQGEwJGUjEPMA0GA1UECBMGRnJhbmNlMRQwEgYDVQQHEwtNYWxhdGF2ZXJuZTEP
MA0GA1UEChMGTk9WQVJDMQswCQYDVQQLEwJTSTEYMBYGA1UEAxMPb2NzaW52ZW50
b3J5LW5nMSYwJAYJKoZIhvcNAQkBFhdsYXVyZW50LnBvbnNAbm92YXJjLmNvbTAh
Fw0xMzAzMDYxNDIyMjNaGBAyNjY1NDA1MDUxNDIyMjNaMIGUMQswCQYDVQQGEwJG
UjEPMA0GA1UECBMGRnJhbmNlMRQwEgYDVQQHEwtNYWxhdGF2ZXJuZTEPMA0GA1UE
ChMGTk9WQVJDMQswCQYDVQQLEwJTSTEYMBYGA1UEAxMPb2NzaW52ZW50b3J5LW5n
MSYwJAYJKoZIhvcNAQkBFhdsYXVyZW50LnBvbnNAbm92YXJjLmNvbTCBnzANBgkq
hkiG9w0BAQEFAAOBjQAwgYkCgYEA0Xv40bEDYzAJnjw4DNMicG5QymDb54ddbyA0
JQe4p4c3w8MEz0wS+ZCnxKk7T63KqWpW+ZCXIbLG1yAC/cvH5rFlX3x+gmzJMxLD
dJPd8qrzdPE5ggpbKBp7WPyE/knzYQyJkwILExDgs8jbwMzHDfODYwsrhQ3vWQZq
xy7rDW0CAwEAAaOB/DCB+TAdBgNVHQ4EFgQUTSWRCHBXIn3sEKvKKHy/9TvCXzEw
gckGA1UdIwSBwTCBvoAUTSWRCHBXIn3sEKvKKHy/9TvCXzGhgZqkgZcwgZQxCzAJ
BgNVBAYTAkZSMQ8wDQYDVQQIEwZGcmFuY2UxFDASBgNVBAcTC01hbGF0YXZlcm5l
MQ8wDQYDVQQKEwZOT1ZBUkMxCzAJBgNVBAsTAlNJMRgwFgYDVQQDEw9vY3NpbnZl
bnRvcnktbmcxJjAkBgkqhkiG9w0BCQEWF2xhdXJlbnQucG9uc0Bub3ZhcmMuY29t
ggkA/+4KPML1O4wwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQUFAAOBgQAXlLHd
2ORwwaij119OmGZFZdjYs2GZWnc7gFymKQiSmFjhrhGb0EG9FTbqGQLW9HzbjNDW
0W4DFHt8GM4Y1miKAja4BO9K1pI6+ezLKAvgsBTt6L8lF2dP7s5X4hYVmIQ/J1kN
ZTA2TD2H7FEORYRKmjVX1hvDkgv0+Z7nweM6iQ==
-----END CERTIFICATE----
```

Certificate creates a “Bad time” parsing runtime error.

E.2.3 Very long running certificate

Certificate validation period 7985 years. Not Before: Jan 4 23:44:33 2015 and Not After : Dec 31 23:59:59 9999 from service 13.93.212.202:8443.

```
-----BEGIN CERTIFICATE-----
MIIBoTCCAQqgAwIBAgIIKYcTAgMJNJUwDQYJKoZIhvcNAQEFBQAwEjEQMA4GA1UE
AxMHMC4wLjAuMDAgFw0xNTAxMDQyMzQ0MzNaGA85OTk5MTIzMTIzNTk1OVowEjEQ
MA4GA1UEAxMHMC4wLjAuMDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAw+I5
LGtKC4eWYMtkRJiB/1wT4lmg+uyBUrZYpqh/cJv4TsBGIWmZ/eRAAZmLFvCrk0xh
uhgocO5mfnUoo4juIUoKlB532JwRjK0yh5Uhy41/ig9sU3ZrhzjDy25Gu8qrHcp6
KcFoBbRytalpocRNqLwTQz7S/x2G97UlesK9yLcCAwEAATANBgkqhkiG9w0BAQUF
AAOBgQAR/SsXlvOIQR5/nDMPyUS72lI7T2r+lkRk5DJ4ZwiUUHyXHNpGuBqCzDO1
aJF2oHlFHUP1NrmfFncvQdhR3UycNlNDiYr4Ucq173TIhD3yyIGyp/IjIM5Clsxp
65ybl586OOArrL7gQ2HndPp6iSAT0YaoO9kSKBaJb2ev9ndy9g==
-----END CERTIFICATE-----
```

E.2.4 One second validation period

Not Before date and Not After date is identical (Jan 27 18:48:24 2011 GMT) in service 88.87.203.213:443.

```
-----BEGIN CERTIFICATE-----
MIIB2jCCAYQCAQAwDQYJKoZIhvcNAQEEBQAweDELMAkGA1UEBhMCVVMxEDAOBgNV
BAgTB1JhZHdhcmUxEDAOBgNVBAcTB1JhZHdhcmUxFjAUBgNVBAMTDTEwLjEyNS41
My4yMDAxEDAOBgNVBAoTB1JhZHdhcmUxGzAZBgNVBAsTElJhZHdhcmUgd2ViIHNl
cnZlcjAeFw0xMTAxMjcxODQ4MjRaFw0xMTAxMjcxODQ4MjRaMHgxCzAJBgNVBAYT
AlVTMRAwDgYDVQQIEwdSYWR3YXJlMRAwDgYDVQQHEwdSYWR3YXJlMRYwFAYDVQQD
Ew0xMC4xMjUuNTMuMjAwMRAwDgYDVQQKEwdSYWR3YXJlMRswGQYDVQQLExJSYWR3
YXJlIHdlYiBzZXJ2ZXIwXDANBgkqhkiG9w0BAQEFAANLADBIAkEA5CtZ1OldJlPv
GtRR7zQumKkcw6j7Z8xK7omN58x3xlzKJeRduu7ikxlZS/Qk+hYKBXWlq1BQm/nD
Q6l5Jzx7CwIDAQABMA0GCSqGSIb3DQEBBAUAA0EALU/A6Yc1d9q6ftXY4K56XW1s
zVcRKkcLTwFWTfMoY47sJS5IUokmm2NjEE3O1ICbD8L+SA8K72WMd8GEkBVnbw==
-----END CERTIFICATE-----
```

E.2.5 Minus 1 day validation period

Not After= May 24 10:09:43 2016 GMT, Not Before = May 25 10:09:43 2016 GMT in service 77.222.40.124:443.

```
-----BEGIN CERTIFICATE-----
MIIDvDCCAqQCCQDMT7CxQXVnSzANBgkqhkiG9w0BAQsFADCBnzELMAkGA1UEBhMC
UlUxDDAKBgNVBAgMA1NQYjEMMAoGA1UEBwwDU1BiMR4wHAYDVQQKDBVUZXN0IFNl
bGYgU1NMIENvbXBhbnkxDTALBgNVBAsMBFRlc3QxJTAjBgNVBAMMHHRlc3Qtc2Vs
Zi1zaWduZWQuc3BhY2V3ZWIucnUxHjAcBgkqhkiG9w0BCQEWD3N1cHBvcnRAc3dl
Yi5ydTAeFw0xNjA1MjUxMDA5NDNaFw0xNjA1MjQxMDA5NDNaMIGfMQswCQYDVQQG
EwJSVTEMMAoGA1UECAwDU1BiMQwwCgYDVQQHDANTUGIxHjAcBgNVBAoMFVRlc3Qg
U2VsZiBTU0wgQ29tcGFueTENMAsGA1UECwwEVGVzdDElMCMGA1UEAwwcdGVzdC1z
ZWxmLXNpZ25lZC5zcGFjZXdlYi5ydTEeMBwGCSqGSIb3DQEJARYPc3VwcG9ydEBz
d2ViLnJ1MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsBp0uCvSkxr0
IvRJBdqlbnP+Wlgl4sJV45l5lESKG5oJgeLNIKM/Y9XXVVsb6eg7XtJ7jly3c1zv
0QtRcuuPE1+0VrBZfzF6TSdPlW26+Sg7cOQr4Ts0YI7i/3zbsp9gA4dMbYJ3dNEe
5IbkgjtWvt4XNjsTJcQapgb8nSJiKWpepoWT4WoIrcrS94GMVCNGmgyBvbRfR6to
yOdNIxcQwXK4Qh5lXR/2e8Oto3keECzllbO1VE8AbO317XS69UutKVi6JW/EqbK3
HWSf+jz+G2EJbjEHCZgACuWEl1YB6y8COCahvL+2va4l94cTUR6LUmYxxz9AEj65
gS2uZfD52wIDAQABMA0GCSqGSIb3DQEBCwUAA4IBAQAo+dh4mESNIGwXnq4+Ud6R
rCvS5Hju9SSx1VWSXExT83VmLIoD4j1Nl8nm6GL4FeiFqQ0GELMUHGcmaeThKN9B
7/MresxY6sKUbiMrso8CrNqeaEDBD9fzzMQV+PhjwYFQZhZIumayjKQdrnCa0sRJ
Ubf5uxM4jh7ANKJipkgybQPKp5P0SlJZGxMYkSQhEJoveG2YuEZUBiYW497MtI/j
JVv/c3aGFNz7h66Wv016nVmGXRGHMbsb9iRnQLOBXOKjxB0uGRvHyLcLPWShCSJs
4tGqbacBw6JzDUw4t+aVSO/BeOrS4MId3XeLwkZ+U6ZXQE0ymU2pAkQlh2qUYHJb
-----END CERTIFICATE-----
```

E.2.6 Invalid validation period

Validation period is -9710 days -07:28:16 which is ca. -26,6 years.

Not Before: Sep 23 16:49:29 2007 GMT, Not After: Feb 21 10:21:13 1981 GMT in service 62.245.182.68:443.

```
-----BEGIN CERTIFICATE-----
MIIDijCCAvOgAwIBAgIJAInsHmhpv2KnMA0GCSqGSIb3DQEBBQUAMIGLMQswCQYD
VQQGEwJERTEQMA4GA1UECBMHQmF2YXJpYTEPMA0GA1UEBxMGQW1iZXJnMRUwEwYD
VQQKEwxBU0FNbmV0IGUuVi4xDzANBgNVBAsTBlN5c3RlbTEPMA0GA1UEAxMGU3lz
dGVtMSAwHgYJKoZIhvcNAQkBFhFzeXN0ZW1AYXNhbW5ldC5kZTAeFw0wNzA5MjMx
NjQ5MjlaFw04MTAyMjExMDIxMTNaMIGLMQswCQYDVQQGEwJERTEQMA4GA1UECBMH
QmF2YXJpYTEPMA0GA1UEBxMGQW1iZXJnMRUwEwYDVQQKEwxBU0FNbmV0IGUuVi4x
DzANBgNVBAsTBlN5c3RlbTEPMA0GA1UEAxMGU3lzdGVtMSAwHgYJKoZIhvcNAQkB
FhFzeXN0ZW1AYXNhbW5ldC5kZTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA
```

```
475BrjWWxhqA9ghkgEYCULXlESdh9Kqf29STHsYkctF0Z2wNxPKx763mdc2XmvHg
GqE08O6Q3JCCjjMwB1zj5V+BQTX0FGobqttuIVjTa6HN/fKZe/ajBKqkYex2imI2
BBVVYmgwcH0v/7QE0QRwK/ZWWpMQsItPk5032qMKXIECAwEAAaOB8zCB8DAdBgNV
HQ4EFgQU11sK9dKYiUZe3tR7vLmpchS1pD8wgcAGA1UdIwSBuDCBtYAU11sK9dKY
iUZe3tR7vLmpchS1pD+hgZGkgY4wgYsxCzAJBgNVBAYTAkRFMRAwDgYDVQQIEwdC
YXZhcmlhMQ8wDQYDVQQHEwZBbWJlcmcxFTATBgNVBAoTDEFTQU1uZXQgZS5WLjEP
MA0GA1UECxMGU3lzdGVtMQ8wDQYDVQQDEwZTeXN0ZW0xIDAeBgkqhkiG9w0BCQEW
EXN5c3RlbUBhc2FtbmV0LmRlggkAieweaGm/YqcwDAYDVR0TBAUwAwEB/zANBgkq
hkiG9w0BAQUFAAOBgQA482el0BBf5AyC22YGNDzo+0LcOCjX7mwUPdp0+rld9E3Q
QBktaY4tW4MLon3/CH1niUivng/oXi6u2+VuMoAzqHbr3kmBPhQQ6gtn+qwm/ljo
FxJrKYbvnEdW+4YT0ZGGtc4FyQvQnVw08cEyogECe0iFtYPt/F6uql8lnljggg==
-----END CERTIFICATE-----
```

E.3 Error analysis

```
  count   |    %    |                            statustext
----------+---------+--------------------------------------------------------------------
 40503588 |  43.647 | timeout
 27762510 |  29.917 | ok
 15195367 |  16.375 | blocked
  3026784 |   3.262 | handshake failed: 5 attempts
  2983096 |   3.215 | handshake failed: unknown state
  1354009 |   1.459 | handshake failed: unknown protocol
  1197553 |   1.290 | handshake failed: socket is not connected
   267247 |    .288 | wrong algorithm type
   251642 |    .271 | handshake failed: unsupported protocol
   219239 |    .236 |
    12515 |    .013 | handshake failed: sslv3 alert bad record mac
     9190 |    .010 | handshake failed: wrong version number
     6523 |    .007 | handshake failed: unable to find public key parameters
     3071 |    .003 | handshake failed: wrong cipher returned
     2328 |    .003 | handshake failed: dh key too small
      829 |    .001 | Invalid certificate data: sbits 0 cert {} chain {}
      446 |    .000 | handshake failed: data between ccs and finished
      415 |    .000 | handshake failed: unsupported algorithm
      217 |    .000 | handshake failed: bad message type
      163 |    .000 | handshake failed: sslv3 alert handshake failure
      145 |    .000 | handshake failed: block type is not 01
       98 |    .000 | handshake failed: unexpected message
       77 |    .000 | handshake failed: unknown cipher returned
       69 |    .000 | handshake failed: unable to find ecdh parameters
       68 |    .000 | handshake failed: bad handshake length
       65 |    .000 | handshake failed: excessive message size
       64 |    .000 | handshake failed: broken pipe
       53 |    .000 | handshake failed: old session cipher not returned
       42 |    .000 | handshake failed: data length too long
       26 |    .000 | handshake failed: wrong signature type
       24 |    .000 | handshake failed: sslv3 alert bad certificate
       17 |    .000 | handshake failed: header too long
       14 |    .000 | handshake failed: wrong tag
       14 |    .000 | handshake failed: sslv3 alert illegal parameter
       13 |    .000 | handshake failed: invalid utf8string
       12 |    .000 | handshake failed: wrong signature length
       11 |    .000 | handshake failed: data too large for modulus
       10 |    .000 | handshake failed: bad signature
        8 |    .000 | handshake failed (5):
        6 |    .000 | handshake failed: too long
        6 |    .000 | handshake failed: unexpected record
        4 |     .000 | handshake failed: internal error
        4 |     .000 | handshake failed: cert length mismatch
        4 |     .000 | handshake failed: ccs received early
        4 |     .000 | handshake failed: sslv3 alert certificate unknown
        3 |     .000 | handshake failed: tlsv1 alert decryption failed
        3 |     .000 | handshake failed: tlsv1 alert internal error
        3 |     .000 | handshake failed: connection reset by peer
        3 |     .000 | handshake failed: unknown digest
        3 |     .000 | handshake failed: EC lib
        3 |     .000 | handshake failed: tlsv1 alert access denied
        2 |     .000 | handshake failed: mstring wrong tag
        2 |     .000 | handshake failed: sslv3 alert unexpected message
        1 |     .000 | handshake failed: invalid encoding
        1 |     .000 | handshake failed: unknown alert type
        1 |     .000 | data_insert rsa_certs ERROR:  null value in column "not_valid_after
    " violates not-null constraint+
        1 |     .000 | handshake failed: decryption failed or bad record mac
        1 |     .000 | handshake failed: point is not on curve
(58 rows)
```

E.4 RSA related information

E.4.1 Distribution of the user Cipher Suites

count	cipher
17392469	ECDHE-RSA-AES256-GCM-SHA384
9969981	AES256-SHA
8752833	DHE-RSA-AES256-GCM-SHA384
7686206	DHE-RSA-AES256-SHA
4405751	ECDHE-RSA-AES128-GCM-SHA256
3113879	RC4-SHA
2587127	AES128-SHA
1916487	AES256-GCM-SHA384
1650812	ECDHE-RSA-AES256-SHA384
1446109	ECDHE-RSA-AES256-SHA
1063321	RC4-MD5
498795	AES256-SHA256
439343	DHE-RSA-AES128-GCM-SHA256
430165	DHE-RSA-AES128-SHA
384697	AES128-SHA256
270236	ECDHE-RSA-AES128-SHA256
196350	DES-CBC3-SHA
109368	DHE-RSA-AES256-SHA256
107022	ECDHE-RSA-AES128-SHA
106442	AES128-GCM-SHA256
54420	ECDHE-RSA-RC4-SHA
35666	DHE-RSA-CAMELLIA256-SHA
12599	EDH-RSA-DES-CBC3-SHA
12370	DHE-RSA-SEED-SHA
9707	DES-CBC-SHA
7023	CAMELLIA256-SHA
5391	SEED-SHA

5029	DHE-RSA-AES128-SHA256
2681	ECDHE-RSA-DES-CBC3-SHA
2371	EDH-RSA-DES-CBC-SHA
2281	DHE-DSS-AES128-SHA
580	DHE-DSS-AES128-SHA256
486	DHE-DSS-AES256-SHA
221	DHE-DSS-AES256-GCM-SHA384
195	DHE-DSS-AES256-SHA256
131	DHE-DSS-AES128-GCM-SHA256
106	CAMELLIA128-SHA
75	EDH-DSS-DES-CBC3-SHA
28	DHE-RSA-CAMELLIA128-SHA
1	DHE-DSS-CAMELLIA256-SHA

E.4.2 Key size results

Table of all found key sizes ordered by size.

#	size	count	#	size	count	#	size	count	#	size	count
1	384	1	49	2000	1	97	2548	1	145	4048	58
2	511	32	50	2014	21	98	2549	1	146	4056	228
3	512	37252	51	2015	1	99	2560	23	147	4065	1
4	640	1	52	2018	7	100	2580	1	148	4068	4
5	700	2	53	2021	3	101	2922	7	149	4069	38
6	768	5143	54	2024	84	102	2948	3	150	4084	1
7	1009	2	55	2028	33	103	3000	4	151	4086	70
8	1010	8	56	2038	3	104	3024	84	152	4087	2
9	1011	2	57	2040	25	105	3027	4	153	4092	73
10	1012	1	58	2045	3	106	3042	1	154	4094	34
11	1013	13	59	2046	14	107	3047	2	155	4095	34
12	1014	12	60	2047	6684	108	3048	3	156	4096	654857
13	1015	28	61	2048	21381173	109	3071	1	157	4097	8
15	1017	165	63	2050	1	111	3073	2	159	4192	3
16	1018	175	64	2054	1	112	3075	2	160	4196	27
17	1019	245	65	2056	117	113	3076	2	161	4296	1
18	1020	501	66	2057	4	114	3077	1	162	4317	1
19	1021	972	67	2058	156	115	3094	1	163	4444	2
20	1022	1329	68	2059	3	116	3096	33	164	4567	1
21	1023	2197	69	2060	1	117	3098	1	165	4906	1
22	1024	5751379	70	2063	171	118	3120	3	166	5000	2
23	1025	3	71	2064	251	119	3121	1	167	5012	4
24	1027	1	72	2066	1	120	3124	1	168	5096	4
25	1028	37	73	2078	1	121	3128	4	169	5098	2
26	1029	29	74	2080	7	122	3137	1	170	5120	26
27	1030	7	75	2084	55	123	3172	2	171	5192	1
28	1034	2	76	2086	1	124	3191	1	172	6096	2

14	1016	53	62	2049	51	110	3072	8141	158	4098	86
29	1039	15061	77	2096	53	125	3192	2	173	6142	2
30	1040	60708	78	2148	5	126	3210	2	174	6144	16
31	1042	10	79	2160	1	127	3248	118	175	6666	1
32	1048	15	80	2176	2	128	3456	1	176	7680	1
33	1054	1	81	2222	1	129	3560	2	177	7777	1
34	1080	1	82	2240	1	130	3584	12	178	8092	1
35	1088	1	83	2248	1	131	3600	3	179	8096	19
36	1095	1	84	2270	1	132	3652	1	180	8112	1
37	1204	2	85	2272	2	133	3653	1	181	8182	1
38	1232	103	86	2291	1	134	3702	1	182	8192	1418
39	1234	6	87	2296	1	135	3753	1	183	8196	6
40	1248	1	88	2304	2	136	3925	1	184	9128	1
41	1280	3723	89	2307	1	137	3931	1	185	9216	1
42	1500	1	90	2344	1	138	3966	1	186	10240	3
43	1512	1	91	2345	3	139	3968	1	187	15360	3
44	1536	2863	92	2393	1	140	3981	1	188	15390	1
45	1537	1	93	2400	1	141	4000	4	189	15424	1
46	1538	1	94	2408	23	142	4007	7	190	16383	1
47	1568	1	95	2432	1584	143	4028	3	191	16384	28
48	1800	1	96	2480	1	144	4046	13			

Appendix F

Service related

F.1 Caltech feedback

```
Von: infosec-noreply@caltech.edu [mailto:infosec-noreply@caltech.edu]
Gesendet: Donnerstag, 25. Dezember 2014 22:46
An: abuse@a1telekom.at
Betreff: 178.189.100.130 blocked at caltech.edu

178.189.100.130 was observed probing caltech.edu for security holes. It has been blocked
at our border routers. It may be compromised.

For more info contact security@caltech.edu Please include the entire subject line of the
original message

   Greg

(time zone of log is PST, which is UTC-0800, date is MMDD) log entries are from Cisco
netflow, time is flow start time

date       time         srcIP           srcPort dstIP           dstPort flags  proto #
    pkts
2014-12-25 12:32:18.589 178.189.100.130 44564   131.215.241.75  443     ....S. 6     1
2014-12-25 12:37:21.459 178.189.100.130 54293   131.215.202.131 443     ....S. 6     1
2014-12-25 12:36:31.549 178.189.100.130 36804   131.215.98.76   443     ....S. 6     1
2014-12-25 12:40:04.613 178.189.100.130 47546   131.215.187.4   443     ....S. 6     1
2014-12-25 12:41:02.651 178.189.100.130 40587   131.215.84.235  443     ....S. 6     1
2014-12-25 12:42:37.618 178.189.100.130 53535   131.215.156.174 443     ....S. 6     1
2014-12-25 12:48:45.579 178.189.100.130 36088   131.215.93.106  443     ....S. 6     1
2014-12-25 12:45:01.630 178.189.100.130 41663   131.215.223.134 443     ....S. 6     1
2014-12-25 12:51:42.502 178.189.100.130 35841   131.215.20.228  443     ....S. 6     1
2014-12-25 12:55:13.506 178.189.100.130 38834   131.215.137.91  443     ....S. 6     1
2014-12-25 12:56:36.495 178.189.100.130 53439   131.215.56.31   443     ....S. 6     1
2014-12-25 12:59:21.481 178.189.100.130 60738   131.215.245.114 443     ....S. 6     1

contact info from:
  spamcop hosttracker
  whois
contact: abuse@telekom.at
```

F.2 Provider feedback

```
Absender: network-abuse@hetzner.de

Dear Sir or Madam,
our monitoring system noticed a network scan (or network attack) from an IP adress under your responsibility.
Please take the necessary actions to avoid this in future. Any feedback would be appreciated. Please use the
following link for your feedback: http://abuse.hetzner.de/statements/?token=2ec25e22bbc11fb1a03402a7066a15de

Important note:
When you reply to us, please leave the abuse ID [AbuseID:1C4CAC:21] unchanged in the subject line.
You should get this information only a few minutes after the incident.
All timestamps are in Central European Time (Berlin)

Best Regards,
Hetzner-Support

Hetzner Online AG
Stuttgarter Str. 1
91710 Gunzenhausen
Tel: +49 (9831) 505-0
Fax: +49 (9831) 505-3
abuse@hetzner.de
www.hetzner.de
Register Court: Registergericht Ansbach, HRB 3204
Management Board: Dipl. Ing. (FH) Martin Hetzner
Chairwoman of the Supervisory Board: Diana Rothhan

Abuse Message 1C4CAC21.txt
##########################################################################
# Netscan detected from host 94.101.38.222 #
##########################################################################
time protocol src_ip src_port dest_ip dest_port
---------------------------------------------------------------------------
Thu Jun 25 22:49:10 2015 TCP 94.101.38.222 51980 => 78.46.120.159 443
Thu Jun 25 22:38:11 2015 TCP 94.101.38.222 51319 => 78.46.121.79 443
Thu Jun 25 22:53:39 2015 TCP 94.101.38.222 52249 => 78.46.132.157 443
Thu Jun 25 22:41:46 2015 TCP 94.101.38.222 51535 => 78.46.133.148 443
Thu Jun 25 22:51:19 2015 TCP 94.101.38.222 52109 => 78.46.136.131 443
Thu Jun 25 22:42:00 2015 TCP 94.101.38.222 51549 => 78.46.139.151 443
Thu Jun 25 22:45:49 2015 TCP 94.101.38.222 51779 => 78.46.150.158 443
Thu Jun 25 22:50:39 2015 TCP 94.101.38.222 52069 => 78.46.165.74 443
Thu Jun 25 22:46:39 2015 TCP 94.101.38.222 51829 => 78.46.172.105 443
Thu Jun 25 22:54:31 2015 TCP 94.101.38.222 52301 => 78.46.172.158 443
Thu Jun 25 22:49:14 2015 TCP 94.101.38.222 51984 => 78.46.176.242 443
Thu Jun 25 22:50:39 2015 TCP 94.101.38.222 52069 => 78.46.182.31 443
Thu Jun 25 22:51:30 2015 TCP 94.101.38.222 52119 => 78.46.185.124 443
Thu Jun 25 22:46:39 2015 TCP 94.101.38.222 51829 => 78.46.185.184 443
Thu Jun 25 22:54:46 2015 TCP 94.101.38.222 52315 => 78.46.186.43 443
Thu Jun 25 22:48:09 2015 TCP 94.101.38.222 51919 => 78.46.188.85 443
Thu Jun 25 22:49:38 2015 TCP 94.101.38.222 52007 => 78.46.193.152 443
Thu Jun 25 22:55:54 2015 TCP 94.101.38.222 52384 => 78.46.206.69 443
Thu Jun 25 22:55:55 2015 TCP 94.101.38.222 52385 => 78.46.208.110 443
```

F.3 Conficker botnet reply

```
This IP is infected (or NATting for a computer that is infected) with the Conficker botnet.

More information about Conficker can be obtained from Wikipedia

Remember: Conficker is not a spam sending botnet. It does not send email or spam. It does not use port 25.

Please follow these instructions.

Dshield has a diary item containing many third party resources, especially removal tools such as Norton Power
Eraser, Stinger, MSRT etc.
```

One of the most critical items is to make sure that all of your computers have the MS08-067 patch installed. But even with the patch installed, machines can get reinfected.

There are several ways to identify Conficker infections remotely. For a fairly complete approach, see Sophos.

If you have full firewall logs turned on at the time of detection, this may be sufficient to find the infection on a NAT:

Your IP was observed making connections to TCP/IP IP address 149.93.26.231 (a conficker sinkhole) with a destination port 80, source port (for this detection) of 41368 at exactly 2015-05-22 06:29:11 (UTC). All of our detection systems use NTP for time synchronization, so the timestamp should be accurate within one second.

If you don't have full firewall logging, perhaps you can set up a firewall block/log of all access (any port) to IP address 149.93.26.231 and keep watch for hits.

WARNING: DO NOT simply block access to 149.93.26.231 and expect to not get listed again. There are many conficker sinkholes - some move around and even we don't know where they all are. Blocking access to just one sinkhole does not mean that you have blocked all sinkholes, so relistings are possible. You have to monitor your firewall logs, identify the infected machine, and repair them if you wish to remain delisted.

Recent versions of NMap can detect Conficker, but it's not 100% reliable at finding every infection. Nmap is available for Linux, xxxBSD, Windows and Mac. Nessus can also find Conficker infections remotely. Several other scanners are available here.

Enigma Software's scanner is apparently good at finding Conficker A.

University of Bonn has a number of scan/removal tools.

If you're unable to find the infection, consider:

If you used a network scanner, make sure that the network specification you used to check your network was right, and you understand how to interpret a conficker detection.

Some network conficker scanners only detect some varieties of conficker. For example, nmap misses some. If you can't find it with nmap, try other scanners like McAfee's. In other words, try at least two.

Are you sure you have found _all_ computers in your network? Sometimes there are machines quietly sitting in back rooms somewhere that got forgotten about. It would be a good idea to run

```
      nmap -sP <ALL of your network specifications>
```

which should list all your computers, printers and other network devices. Did you see all the computers you expected to see?

The infected computer may be turned off at the time you ran the scan or not on the network. Double-check everything was turned on during the scan.

If you have wireless, make sure it's secured with WPA or WPA2, and that "strangers" can't connect. WEP security is NOT good enough.

Many versions of Conficker propagate via infected thumbdrives/USB keys. When an infected machine is found, ALL such devices associated with the machine should be considered suspect, and either destroyed or completely reformatted.

Conficker also propagates by file and printer shares.

If you simply remove the listing without ensuring that the infection is removed (or the NAT secured), it will probably relist again.

How to resolve future problems and prevent relisting

Norton Power Eraser is a free tool and doesn't require installation. It just needs to be downloaded and run. One of our team has tested the tool with Zeus, Ice-X, Citadel, ZeroAccess and Cutwail. It was able to detect and clean up the system in each case. It probably works with many other infections.

Is this IP address a NAT gateway/firewall/router? In other words, is this IP address shared with other computers? See NAT for further information about NATs and how to secure them.

If this IP address is shared with other computers, only the administrator of this IP address can prevent this happening again by following the instructions in NAT to secure the NAT against future infections. In this way, no matter how badly infected the network behind the NAT is, the network can't spam the Internet. The administrator can also refer to Advanced BOT detection for hints and tips on how to find the infected computer behind a NAT.

What affect is this listing having on you?

The CBL is intended to be used only on inbound email from the Internet.

If you are being blocked from IRC, Chat, web sites, web email interfaces (eg: you're using Internet Explorer or Firefox to send email) or anything other than basic email with a mail reader like Exchange, Thunderbird etc, the provider of this service is using the CBL against our recommendations. Contact the provider and refer them to http://cbl.abuseat.org/tandc.html and refer them to item 2 and 7.

If you are an end user: If you get an immediate popup indicating your email was blocked when you attempt to send email, this means one of two things:

You aren't using your provider's preferred configuration for sending email. This is most frequent with roaming users (eg: you're using an Internet Cafe, and are using your home provider to send email). A provider will normally give you instructions on how your mail reader should authenticate to their mail servers, perhaps on a different port (usually 587). Make sure that you comply with the provider's instructions on mail reader configuration where it refers to "SMTP relay server", "SMTP authentication" etc.

If you are complying with your provider's instructions, your provider is violating the CBL Ter78.46.120.159ms and Conditions and blocking their own users. Contact your provider and refer them to http://cbl.abuseat.org/tandc.html and refer them to item 6 and 7.

If you get the blocking email message by return email (instead of by immediate popup), your provider is listed in the CBL, not you. Contact your provider and tell them that their IP address is listed by the CBL.

Note that the CBL is not responsible for how providers misuse the CBL. This is their problem, not ours.

If your IP address changes periodically (such as with reconnecting to your provider, connecting through an Internet Cafe etc), this is usually a dynamic (DHCP) IP address, meaning that it's most likely not you that is infected. As above, make sure that your mail reader is configured correctly as per your provider. In this case, delisting the IP address will probably not do anything useful.

If this listing is of an unshared IP address, and the affected access is email, then, the computer corresponding to this IP address at time of detection (see above) is infected with a spambot, or, if it's a mail server, in some rare cases this can be a severe misconfiguration or bug.

The first step is to run at least one (preferably more) reputable anti-spam/spyware tools on your computer. If you're lucky, one of them will find and remove the infection.

If you are unable to find it using anti-virus tools, you may want to take a close look at the discussions of netstat or tcpview in the "Per-machine methods" section of Finding BOTs in a LAN.

If the above does not help, you may have to resort to taking your computer to a computer dealer/service company and have them clean it.

If all else fails, you may need to have your machine's software re-installed from scratch.

WARNING: If you continually delist 178.189.100.130 without fixing the problem, the CBL will eventually stop allowing the delisting of 178.189.100.130.

If you have resolved the problem shown above and delisted the IP yourself, there is no need to contact us.

Click on this link to delist 178.189.100.130.

F.4 Mail feedback reply

Dear user, dear administrator,
in reply to your abuse message we want to inform you:

Subject: Hacking Activity from 185.3.232.72
Date: Tue, 24 May 2016 11:49:17 -0300 (BRT)
From: mtso@matera.com

The background of the supposedly treatment is a project of our University of Applied Science (FH-Joanneum Kapfenberg) to implement a security service. The planned service's goal is to analyze TLS keys according implementation flaws. To achieve this goal we need a large amount of public TLS keys which will become analyzed for multiply used RSA factors.

This service is planned as on-demand service, where users can validate their public keys and also active notification in legal scale of operations can be requested to become informed about the found weaknesses.

An overview of the complete security project, further information about the basics, some mathematical background and the list of the involved modules can be found on:

http://d10122.dedicated.alfahosting-pro.de/

* It is not planned to contact ip-addresses more than once within an acceptable time range, which is normally guaranteed by the used algorithms. We only try to verify the existence of TLS related services in this phase. Only the public certificate of your TLS based service is retrieved once at a later time to investigate it regarding aforesaid weakness.

```
The first phase consists of scanning as much as possible ip-addresses to identify TLS-services, which will be
examined if the mentioned weaknesses appear, such that the affected users can be informed accordingly.

*           Statement:  We will at no time try to get access to your systems (login) or access any other private
            area!

Your address range 201.16.192/18 has been removed from our project.
If you wish other ranges to become removed, please send us a short mail.
If you want completely or for additional network ranges become removed from our project, please tell us all your
relevant network ranges and we will avoid to contact any affected address.
But if you want to become part of our project, please inform us about all relevant network ranges and provide
an email address where we can contact you in the case of found vulnerable certificates.
Please apologize if we perturbed your service and we hope for your understanding and your support of our security
project.
If you have any further questions, please do not hesitate to contact us, we will answer all your questions regarding
our FJreSafe project.
With best regards,
Gerhard Reithofer
```

Appendix G

Statistics and evaluation

This tools are only used to create evaluations, data and charts for this thesis and are not necessary for scanning process itself.

G.1 Statistic tools (sta)

```
calc_certretr.sh    - extract statistics from retr_cert error log (G.1.1)
calc_rsadb.sh       - extract statistics from retr_rsa log files (G.1.4)
calc_speed.tcl      - extract statistics from temp. scan files (G.1.2)
extract_scanlog.sh  - extract scan speed values from scan log files (G.1.3)
show_bannerinfo.sh  - show banner info, type, ftp or all (G.1.5)
stats_cpu_det.sh    - statistics about var. CPU usage from top logs (G.1.6)
stats_db_group.sh   - table SCAN timed group statistics (G.1.7)
stats_db_join.sh    - grouped statistics from db_join log files (G.1.8)
stats_grpscan.sh    - grouped values from parscan logfiles (G.1.9)
stats_grp_task.sh   - grouped values from task log files (G.1.10)
stats_log_vs_db.tcl - ports statistics log filex vs DB values (G.1.11)
stats_parscan.sh    - evaluation of the temporary scan files (G.1.12)
stats_port_scan.sh  - port statistics from basescan log files (G.1.13)
stats_procs.sh      - process statistics from nohup output files (G.1.14)
stats_retrieve.sh   - statistics from retrieve_* log files (G.1.15)
stats_scanlog.sh    - statistics from scan process log files (G.1.16)
stats_sql_rsa.sh    - table RSA_CERTS timed group statistics (G.1.17)
stats_single_cpu.sh - CPU statistics from top_numproc log files (G.1.18)
stats_xfer.sh       - statistics for openssl retrieve certificate step G.1.19
```

G.1.1 calc_certretr.sh

```
$ sta/calc_certretr.sh -h
Statistics from retrieve cert log files
Error: input file '-h' does not exist
Usage: calc_certretr.sh logfile
Available logfiles:
log/retrieve_cert_2016-06-04.elog ...
```

G.1.2 calc_speed.tcl

```
$ sta/calc_speed.tcl -h
Evaluate scan speed tool 0.2
```

```
Usage: sta/calc_speed.tcl [-h] [-t tmpdir] [-p pattern]
 -t .. temp directory, def. '/dat/d4/ase/master/keyservice/tmp'
 -p .. file pattern, def. '*', '.csv' is appended
 -d .. do a more detailed analyzation
 -h .. this help text
```

Calculate scanning rate from timestamps of masscan output files.

G.1.3 extract_scanlog.sh

```
$ sta/extract_scanlog.sh
Extract scan speed from scan log files
Usage: extract_scanlog.sh log_dir
```

G.1.4 calc_rsadb.sh

```
$ sta/calc_rsadb.sh
Statistics from DB importlog retrieve files
Usage: calc_rsadb.sh [-s] [-a] [-d] [-v] [-t type] [-f from] [-u until] logfiles ..
 -s do not show summary
 -a do not show average
 -d do not show details
 -v show more information
 -t filters records of a specific type
 -f first entry which is used (all or part of in format 'YYYY-MM-DD HH:MM:SS')
 -u last entry which is used (all or part of in format 'YYYY-MM-DD HH:MM:SS')
 dblogfiles '-' for stdin or files created by log_dbinserts.sh command, available:
log/retrieve_new_2016-06-08.log ...
```

G.1.5 show_bannerinfo.sh

```
$ sta/show_bannerinfo.sh
Scanner banner statistics
Usage: show_bannerinfo.sh [--types|--all|--ftp]
```

G.1.6 stats_cpu_det.sh

```
sta/stats_cpu_det.sh -h
Detailed statistics from top log files
Usage: stats_cpu_det.sh [-s {0|1}] [-c {0|1}] [-p {0|1}] [file ..]
 -h or --help - this usage message
 -s .. cpu mode (def. 1)
 -c .. summary mode (def. 0)
 -p .. process mode (def. 0)
 -f .. top file to analyze (def. top_2016-06-07_numproc_48_3.out)
```

G.1.7 stats_db_group.sh

```
$ sta/stats_db_group.sh -h
Extract grouped values from table SCAN - 0.3
Usage: stats_db_group.sh [-h] [-o outfile] [-d database] [-S|-M|-H|-D] start until
  -o .. specify output file (def. stdout)
  -d .. specify database (def. 'oldservice')
  -f .. group list by seconds
  -M .. group list by minutes (default)
  -H .. group list by hours
  -D .. group list by days
  start, until must be in format 'YYYY-MM-DD HH:MM:SS'
```

G.1.8 stats_db_join.sh

```
$ sta/stats_db_join.sh -h
Extract statistics from db_join log files
usage: stats_db_join.sh [-t tmp_log] [-s] [-d] [-M|-H|-S] [-b begin_time] [-e end_time]
 -t .. define tmp_log dir, default is: sta/../log
 -s .. supress summary information
 -d .. show detailed information
 -H .. show hour group CSV values
 -M .. show minute group CSV values
 -S .. show second group CSV values
 begin_time, end_time must be in format: YYYY-MM-DD HH-MM-SS
```

G.1.9 stats_grpscan.sh

```
$ sta/stats_grpscan.sh -h
Extract grouped values from parscan logfiles
Usage: stats_grpscan.sh [-h] [-f|-S|-M|-m|-H|-d] [scan_options]
  -f .. group list by seconds
  -S .. group list by 10 seconds
  -M .. group list by minutes (default)
  -m .. group list by 10 minutes
  -H .. group list by hours
  -d .. group list by days
Possible scan_options are:
  usage: stats_parscan.sh [-t tmp_log] [-s] [-d] [-b begin_time] [-e end_time]
  -t .. define tmp_log dir, default is: sta/../log/other_ports
  -d .. supress detailed information
  -s .. supress summary information
  begin_time, end_time must be in format: YYYY-MM-DD HH-MM-SS
```

G.1.10 stats_grp_task.sh

```
$ sta/stats_grp_task.sh -h
Extract grouped values from task log files
Usage: stats_grp_task.sh [-h] [-f|-S|-M|-m|-H|-d] [file_pat]
  -f .. group list by seconds
  -S .. group list by 10 seconds
  -M .. group list by minutes (default)
  -m .. group list by 10 minutes
  -H .. group list by hours
  -d .. group list by days
  default file pattern is: sta/../log/keysvc/tasks_*.mon
```

G.1.11 stats_log_vs_db.tcl

Script for comparing log file port statistics with database evaluation values.

G.1.12 stats_parscan.sh

```
$ sta/stats_parscan.sh --help
usage: stats_parscan.sh [-t tmp_log] [-s] [-d] [-b begin_time] [-e end_time]
 -t .. define tmp_log dir, default is: log/other_ports
 -d .. supress detailed information
 -s .. supress summary information
 begin_time, end_time must be in format: YYYY-MM-DD HH-MM-SS
```

G.1.13 stats_port_scan.sh

```
$ sta/stats_port_scan.sh -h
Port statistics from basescan log files
Usage: stats_port_scan.sh [-h] [-q] [-a|-S|-M|-m|-H|-d] [-t tempdir] [file_pat]
  -a .. output all service records
  -S .. group list by seconds
  -M .. group list by minutes (default)
  -H .. group list by hours
  -d .. group list by days
  -q .. no output, only summary
  -t .. specify other temp dir (def. sta/../tmp)
  default file pattern is: basescan_all.out
```

G.1.14 stats_procs.shpp

```
$ sta/stats_procs.sh -h
Extract process statistics from nohup output files
usage: stats_procs.sh [date_pattern]
  -b .. begin date
  -e .. end date
 begin and end date must match format: YYYY-MM-DD HH-MM-SS
```

Process statistics with time filter. Input file “nohup_out-20150826_28.out” is hard coded and must be changed on demand.

G.1.15 stats_retrieve.sh

```
$ sta/stats_retrieve.sh -h
Statisics from retrieve log files
Usage: stats_retrieve.sh logfiles
sta/../log/retrieve_new_2016-06-08.log
...
```

G.1.16 stats_scanlog.sh

```
$ sta/stats_scanlog.sh -h
Statistics from scan process log files
usage: stats_scanlog.sh [-t tmp_log] [-p pattern]
default tmp_log is: sta/../log
```

G.1.17 stats_sql_rsa.sh

```
$ sta/stats_sql_rsa.sh -h
RSA_CERTS DB grouped statistics
Error: missing time period grouping value (must be D, H, M or S)
Usage: stats_sql_rsa.sh {D|H|M|S} [extra]
  Average values of Days, Hours, Minutes and Seconds of scanned RSA certificates.
  If the extra parameter is specified only succesful retrieved certificates are counted.
  Evaluation range is from '2016-06-04 18:34:04' until '2016-06-15 15:45:43'.
```

G.1.18 stats_single_cpu.sh

```
$ sta/stats_single_cpu.sh -h
Statistics for each single CPU from top_numproc log files
Missing logfile paramaters
Usage: stats_single_cpu.sh logfiles
sta/../log/top_2016-06-05_numproc_48_0.out
...
```

G.1.19 stats_xfer.sh

Evaluation tool for openssl scanning tests (refer to section 4.1.1).

G.2 Graphics and Charts

A “chart manager” program with pre-evaluated data and data extraction files (modules).

```
GraphManager.tcl - simple management tool for creating charts
data             - subdirectory for various GraphManager input data files
modules          - subdirectory containing GraphManager modules
```

Bibliography

Bernstein, D. J. (2005). “Factoring into coprimes in essentially linear time”. In: *Journal of Algorithms* 54, pp. 1–30. URL: `http://cr.yp.to/papers.html#dcba`.

Bertalanffy, Ludwig Von (1968). *General system theory: Essays on its foundation and development.* New York: George Braziller.

BKA.Österreich (1999). *Bundesgesetz über den Schutz personenbezogener Daten (Datenschutzgesetz 2000 - DSG 2000).* Bundeskanzleramt Österreich. URL: `https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=bundesnormen&Gesetzesnummer=10001597` (visited on 04/16/2016).

BSI (2016). “Kryptographische Verfahren: Empfehlungen und Schlüssellängen”. In: *BSI - Technische Richtlinie* BSI TR-02102-1.2016-01.

Buchmann, Johannes. Prof. (1994). “Faktorisierung großer Zahlen”. In: *Spektrum der Wissenschaft,* September 1994. Spektrum der Wissenschaft Verlagsgesellschaft mbH, p. 80..

Cooper, D. et al. (2008). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc5280.txt`.

cve@mitre.org (2013). *CVE-2014-0160.* Ed. by Common Vulnerabilities and Exposures List. The MITRE Corporation. URL: `https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160` (visited on 04/16/2016).

Daigle, L. (2004). *WHOIS Protocol Specification.* RFC 3912. RFC Editor.

Development-Group (2016). *PostgreSQL.* Ed. by Development-Group. Web. The PostgreSQL Global Development Group. URL: `https://www.postgresql.org/` (visited on 07/21/2016).

Dierks, T. and E. Rescorla (2008). *The Transport Layer Security (TLS) Protocol Version 1.2.* RFC 5246. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc5246.txt`.

Durumeric, Zakir (2013). *Parsing X.509 Certificates with OpenSSL and CCVE-2014-0160.* University of Michigan. URL: `https://zakird.com/2013/10/13/certificate-parsing-with-openssl` (visited on 04/16/2016).

– (2014). *Internet-Wide Scan Data Repository Internet-Wide Scan Data Repository Internet-Wide Scan Data Repository.* Ed. by Zakir Durumeric. University of Michigan, University of Illinois. URL: `https://zakird.com/` (visited on 09/16/2016).

Durumeric, Zakir, David Adrian, et al. (2015). “A Search Engine Backed by Internet-Wide Scanning”. In: *Proceedings of the 22nd ACM Conference on Computer and Communications Security.* University of Michigan, University of Illinois. URL: `https://www.censys.io/`.

Durumeric, Zakir, James Kasten, et al. (2013). “Analysis of the HTTPS Certificate Ecosystem”. In: *Proc. IMC.* Barcelona, Spain: ACM, pp. 291–304. ISBN: 978-1-4503-1953-9. DOI: `10.1145/2504730.2504755`. URL: `http://doi.acm.org/10.1145/2504730.2504755`.

Durumeric, Zakir, Eric Wustrow, and J. Alex Halderman (2013a). “ZMap: Fast Internet-Wide Scanning and its Security Applications”. In: *Proceedings of the 22nd USENIX Security Symposium.*

– (2013b). “ZMap: Fast Internet-wide Scanning and Its Security Applications”. In: *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13).* Washington, D.C.: USENIX, pp. 605–

620. ISBN: 978-1-931971-03-4. URL: `https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric`.

Eastlake, D. (2011). *Transport Layer Security (TLS) Extensions: Extension Definitions.* RFC 6066. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc6066.txt`.

Freier, A., P. Karlton, and P. Kocher (2011). *The Secure Sockets Layer (SSL) Protocol Version 3.0.* RFC 6101. `http://www.rfc-editor.org/rfc/rfc6101.txt`. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc6101.txt`.

Granlund, Torbjörn and the GMP development team (2014). *The GNU Multiple Precision Arithmetic Library.* 6.0.0. Free Software Foundation, Inc. URL: `https://gmplib.org/gmp-man-6.0.0a.pdf`.

Hellmann, Martin E. (1979). "The Mathematics of Public-Key Cryptography". In: *Scientific American* August 1979, pp. 146–157.

Holz, Ralph-Günther (2014). "Empirical analysis of Public Key Infrastructures and Investigation of Improvements". PhD thesis. Technische Universität München. URL: `https://mediatum.ub.tum.de/download/1182735/1182735.pdf`.

Housley, R. et al. (2002). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 3280. `http://www.rfc-editor.org/rfc/rfc3280.txt`. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc3280.txt`.

ISO8348 (2002). *Information technology - Open Systems Interconnection - Network service definition.* ISO ISO/IEC 8348:2002(E). ISO copyright office, Case postale 56, CH-1211 Geneva 20, Switzerland: International Organization for Standardization. URL: `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35872`.

John Viega Matt Messier, Pravir Chandra (2002). *Network Security with OpenSSL, Cryptography for Secure Communications.* Ebook ISBN: 978-0-596-10345-3 | ISBN 10: 0-596-10345-X. O'Reilly Media Final Release.

Jonsson, J. and B. Kaliski (2003). *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1.* RFC 3447. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc3447.txt`.

Kerckhoffs, Auguste (1883). "La cryptographie militaire". In: *Journal des sciences militaires* IX, pp. 5–83. URL: `https://en.wikipedia.org/wiki/Kerckhoffs's_principle#cite_ref-4`.

Linn, J. (1993). *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures.* RFC 1421. RFC Editor. URL: `https://tools.ietf.org/rfcmarkup?doc=1421`.

Lyon, Gordon "Fyodor" (2011). *Nmap Network Scanning.* ISBN 978-0-9799587-1-7 | ISBN-100-9799587-1-7. Insecure.Com LLC. URL: `https://nmap.org/book/man-port-scanning-techniques.html` (visited on 08/04/2016).

Mogul, Jeffrey (1984). *Broadcasting Internet Datagrams.* STD 5. `http://www.rfc-editor.org/rfc/rfc919.txt`. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc919.txt`.

Postel, J. (1981). *Internet Control Message Protocol.* STD 5. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc792.txt`.

Postel, Jon (1981). *Transmission Control Protocol.* STD 7. `http://www.rfc-editor.org/rfc/rfc793.txt`. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc793.txt`.

Rekhter, Yakov et al. (1996). *Address Allocation for Private Internets.* BCP 5. `http://www.rfc-editor.org/rfc/rfc1918.txt`. RFC Editor. URL: `http://www.rfc-editor.org/rfc/rfc1918.txt`.

X.690, ITU-T (2015). *Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).* ISO ISO/IEC 8825-1. Geneva, Switzerland: International Telecommunication Union. URL: `http://www.itu.int/rec/T-REC-X.690-201508-I`.