University of Piraeus

Department of Digital Systems

MSc Thesis

“TLS-Observer : A framework for real-time TLS data visualization, audit, and incident response.”

Dimitris Bachtis

MTE1328

10/2015

Abstract

Web PKI and the TLS ecosystem as a whole is fragile. It is as strong as its weakest link and its integrity can be easily compromised by any of its links. TLS infrastructure requires continuous monitoring for any organisation to be able to respond to incidents that come up, from bugs in specific implementations and software to root CAs or certificates being compromised or even backdoored. TLS-Observer is a framework that provides real-time and distributed compliance auditing of TLS configurations amongst the vast network hosts of an organisation. It can help system administrator have an overview of the compliance level of their systems and even help them identify deficiencies and reach a better level of security. It can also help incident response by crawling the network in real time for specific vulnerabilities.
Building on Go's out of the box concurrency, TLS-Observer offers a fast and reliable framework that can be used in all kinds of organisations and can also aid researchers get an overview of the TLS space all over the Internet without having to crunch all the data themselves.

More info and source code can be found on : https://github.com/mozilla/TLS-Observer

Introduction

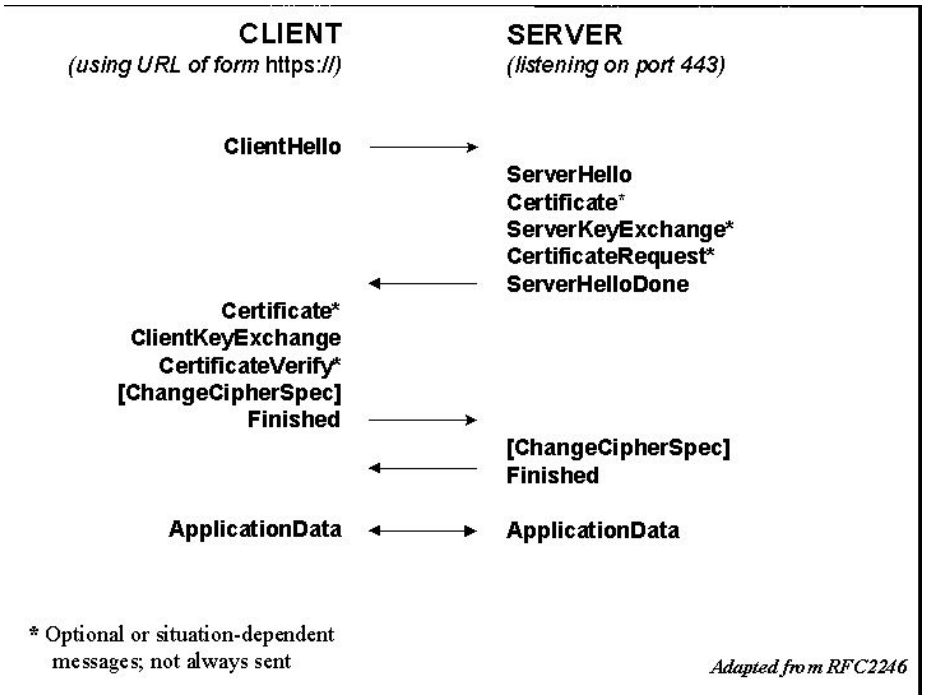
Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols designed to provide communications security over a network.

SSL was created by Netscape in 1994 and its goal was to provide encrypted communication between a client and a server. It's usage has grown and it is now the standard for secure communications through the world wide web. The protocol had 3 major updates, SSLv2 and SSLv3, till its name was changed to TLS (v1) in 1999. All the versions before TLSv1 are considered broken today and are deprecated. Another 2 updates to the newly named protocol up until now to take it to TLS v1.2. The responsible bodies and authorities are currently developing the next version of the protocol (TLS v1.3) which, among other things, removes support for older and weaker hashing algorithms.

Inner Workings of TLS

Since this thesis is not a presentation of the TLS protocol, only a brief overview of the way TLS works will be provided in this chapter. For more info on the exact details the reader can always study the RFCs.

The TLS protocol relies on PKI, more specifically x509 certificates, to be able to function and provide authentication on top of its built in confidentiality.



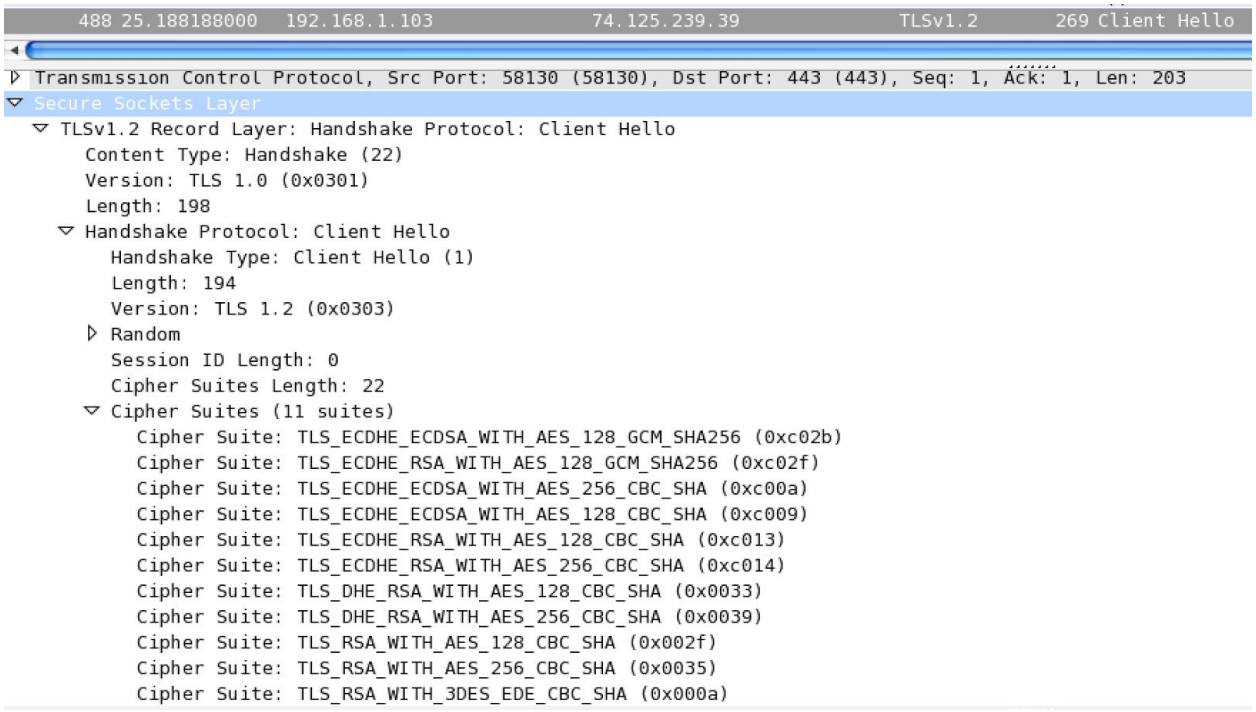
TLS Handshake

The above image shows the flow of messages that takes place when a client tries to authenticate a web server, in order for them to communicate “secretly”.

The 2 most crucial parts of the negotiation between the client and the server are the certificate provided by the server and the negotiation regarding the algorithms that will be used (mostly done in the clientHello and serverHello messages but can also be changed through the ChangeCipherSpec message).

The certificate provided by the server must be associated with the domain requested by the client in order for it to be valid. The x509 certificate also contains information on certificate validity range, certificate allowed uses and more.

Certificates are provided and signed by the trust providers of the internet PKI, the Certificate Authorities (CAs). They are well recognised organisations which, by signing certificates with their secret private keys, provide a model of trust for the use of TLS around the web.



Network Trace of TLS handshake

Now on the TLS protocol itself, the client and the server use a set of algorithms to ensure confidentiality, authenticity and integrity of the data transferred between them. This set of algorithms is called a ciphersuite. Each ciphersuite's name describes the algorithms used, for example ECDHE-RSA-AES128-GCM-SHA256 uses Ephemeral Elliptic Curve Diffie-Hellman key exchange with RSA negotiation, AES128 in GCM mode for encryption and SHA256 for hashing. The ciphersuite used can be decided by either the server's preferences (server side ordering) or by the client's preferences (client side ordering). Either way, the ciphersuite that is going to be used must be supported by both sides.

Great thought has to be put into the complex issue of ciphersuite ordering. It is very difficult to balance a high level of security (provided by the use of state of the art algorithms) with performance (which is degraded by complex algorithms) and without messing with backwards compatibility, on which a lot of older clients and servers rely in order to access services.

TLS offers the feature of Perfect Forward Secrecy (with perfect being a pretty "heavy" word). In cryptography this means that the keys used in a session cannot be derived from the master key which they were derived from. This feature is provided by the use of Ephemeral Diffie-Hellman algorithm for key agreement, either it is the elliptic curve alternative or not - (EC)DHE.

x509 Certificates, as do normal certificates, have an expiration date. They can also be invalid for other reasons though, the issuer CA has for some reason stopped being trusted etc. The clients must be alerted in order to be able to identify revoked certificates. That is done through Certificate Revocation Lists (CRLs) which are huge lists of revoked certificates that clients can search for certificates of interest.

Another x509 feature to check certificate revocation status is Online Certificate Status Protocol (OCSP). It is an alternative to CRLs and essentially it provides a way to query certain entities, OCSP responders which are operated by CAs, for revocation info about certain certificates. That way client minimise the work they have to do and the network traffic they have to create to get revocation information.

State of TLS

TLS Ecosystem

Web PKI and the TLS ecosystem as a whole is fragile. Because of the way x509, on which TLS on the web depends on, is structured it is as strong as its weakest link and its integrity can be easily compromised by any of its links.

Due to its generic nature, x509, has through the years become the defacto standard on privacy and security on the World Wide Web. Although its age is starting to show the protocols and ecosystem around are far from going away. It is on the way of securing its way into the IoT world which will bring billion of computers, which will need x509 certificates to be able to communicate and authenticate each other, online.

In the recent years more and more research is being conducted on the security of the TLS ecosystem, x509 and the TLS protocol itself.

This research combined with the widespread nature of the TLS protocol, which can be leveraged by attackers, has led to the announcement of several vulnerabilities regarding the TLS ecosystem. The ecosystem is mentioned instead of the protocol because the vulnerabilities do not threaten the protocol only, but can also be found in specific implementation or specific processes that govern the usage of TLS. TLS security research has also offered us some of the most unusual vulnerability names, as you will see in the next paragraphs.

TLS Vunerabilities

TLS vulnerabilities, as mentioned shortly earlier, can, for most of the cases, be sorted in three categories. The first category are the most obvious ones that come from the protocol itself. With this we describe the vulnerabilities that lie in the design and specification of the protocol which can be leveraged by attackers to perform malicious activities. The second category includes vulnerabilities in the different implementations of the TLS protocol. The third category includes vulnerabilities that were caused by “holes” in the TLS ecosystem. These can include malfunctions in processes that were not followed exactly as they should or attacks at the weak links of the ecosystem. Other generic or newer vulnerabilities have been seen in the last years and will be presented on their own. The goal of this chapter is not to present all the

vulnerabilities of the TLS protocol but to make a general introduction to the categories these vulnerabilities fall into and introduce the most known of them. This way the reader will be familiar with all the basic aspects of the TLS protocol and capable of perceiving and reviewing the uses of the TLS auditing and monitoring framework developed and presented later in this thesis.

TLS protocol oriented vulnerabilities

The attacks and vulnerabilities described in this chapter have to do with specific design decisions of the TLS protocol or vulnerabilities in the algorithms used in the protocol, either for encryption or authentication.

First of all we have md5 collision attacks that were used to create malicious certificates.
Next there are the known attacks on RC4, an algorithm that has been used for many years and has been known to have several weaknesses. Although these vulnerabilities are not yet proven to be practical as they require a lot of resources, the algorithm is considered exploitable and not sufficiently secure.

Next up, and with our first catchy names, is a category of side channel attacks that take advantage of compression either at the TLS protocol level or at the HTTP level. These attacks are, in chronological order of disclosure, CRIME, TIME and BREACH. CRIME builds on the use of TLS-level compression and allows an attacker to guess secrets by observing and manipulating compressed traffic. CRIME can be easily mitigated by turning off TLS compression.
BREACH and TIME attacks make use of compression on the HTTP protocol level and allow the attacker to perform chosen plain-text attacks. They are more hard to be mitigated because they attack on a different level of the communication stack.

The BEAST attack uses implementation issues in the CBC of TLS v1.0 in order to be able to decrypt parts of the packets that are run over TLS. For internet traffic that uses HTTP these parts were the HTTP cookies. This is a classic attack that build upon protocol specification problems and can be mitigated by using more secure ciphers, that is going to TLS v1.2 and AES-GCM.

POODLE is the latest variant of a series of padding oracle attacks that appear in the TLS protocol. Most of them can be mitigated by using the AES-GCM algorithm. POODLE itself is an attack on SSLv3 which makes the specific version of the protocol vulnerable and considered broken. The only mitigation against this attack defaulting to versions newer than SSLv3, so TLSv1 and on.

Dual EC DRBG backdoor is another protocol vulnerability, which can also be put under the ecosystem category due to its intriguing background. This vulnerability is actually a back door in the Dual Elliptic Curve Deterministic Random Bit Generator (Dual EC DRBG) standardized and placed by NSA in the standardisation of NIST in 2006. When this backdoor and the NSA's story came up, NIST proposed, as a mitigation, all TLS users abandon use of the aforementioned random bit generator.

Implementation Specific Vulnerabilities

Vulnerabilities lie in all software that is produced. Whether they are found before this software is shipped relies on the security and QA testing every organisation has in place and the secure software development training and experience of its employees. TLS/SSL implementations are software and, as it is in their nature, have vulnerabilities. The security aspect of this software makes vulnerabilities much more serious as they do not just lead to a crash but can leak data thought to be secret and confidential, provide unauthorised access to sensitive infrastructure or let malicious actors get access to user' private data, be that banking information (with catastrophic consequences) or facebook chat (with severe leak of privacy).

There also have been a lot of cases where, due to development mistakes, libraries skipped certificate validation altogether, validated chains incorrectly or were DOSed by maliciously crafted SSL certificates.

Sadly, TLS implementations show their age as software constructions and have a lot of vulnerabilities. Source code auditing of these libraries has become intense in the latest years, after some major vulnerability disclosures, and more and more vulnerabilities come up. Below we will go through some of the most serious and known vulnerabilities in TLS implementations.

The most known vulnerability of such type, mostly because it was found in the most used TLS library OpenSSL, is Heartbleed. The attack behind this vulnerability exploits a development error in the Heartbeat protocol extension. By exploiting this attack, attackers could read arbitrary memory chunks (up to 64 KB) from the servers which, in the worst cases, to the server's private key being leaked with devastating consequences.

The publication of Heartbleed, in spite of its severity as a vulnerability, might have done the TLS ecosystem more good than harm as it shed light on the state of security of TLS libraries and led to a lot of sponsorships to aid the work of open source developers and improved the code quality of these libraries as it pointed out the need for continuous auditing.

FREAK (Factoring RSA Export Keys) is another one of the latest vulnerabilities in OpenSSL, although not as severe as Heartbleed, that also affected multiple mainstream web browser and other SSL implementations. This attack exploits the “old” concept of Export Cryptography and does so by using guiding the server to use short RSA keys (<512 bits) which are computationally easy to crack with today’s , resources.

The latest vulnerabilities of this type, and in the same context as FREAK, is Logjam. This vulnerability targets the key Diffie-Hellman exchange algorithm (only the non Elliptic Curve one) and it’s Perfect Forward Secrecy feature. It was inspired by the FREAK attack as it also uses export grade Diffie-Hellman keys do weaken the encryption. By requesting DHE_EXPORT instead of DHE the client instructs the server to use short 512 keys that are considered really weak (same goes for 768 and 1024 keys). This allows MitM attackers to intercept and decrypt traffic easily. Another aspect that makes this attack even more serious is that the parameter keys used during the DH key exchange are highly reused by websites (19.7% of Alexa top 1M websites used the same 1024 keys). This allows attackers to do most of the computations needed beforehand and crack even 1024 keys in a matter of minutes.
To mitigate this attack, disabling export grade Diffie-Hellman and not allowing short, weak (< 2048) keys is mandated.

Ecosystem related vulnerabilities

The TLS ecosystem relies heavily on Certificate Authorities (CAs) as the root of trust at its foundation. That makes them the most logical sought after target for attackers that want to use TLS for malicious purposes. Due to the nature of PKI and the power lying in the hands of CAs they are a really weak link in the stability of the ecosystem. Due to the freedom they have to issue certificates, at will, the public can only trust them to do the right thing. Due to the age of PKI and x509 there aren’t any sufficient built in processes and safeguard to make sure CAs cannot issue certificates that are not meant for valid usage. Because of all that attackers try to manipulate CAs to issue certificates for domains they do not own, by appearing as a legitimate owner, or even try to infiltrate the CA’s network and take control of certificate issuance altogether.

In this chapter we will showcase some attacks that have happened or have become plausible in the latest years so the reader can understand how a CA compromise of any type can impact the TLS ecosystem as a whole.

One of the greatest CA compromisations happened in 2011. Diginotar, a CA responsible for handling the Dutch PKI program, was totally compromised with attackers gaining full access to the company's keys and being able to issue and sign malicious certs at will. Fox-IT , an information security company, conducted research on the attack and found out that the issued certs were heavily used for man in the middle attacks. That probably allowed the attackers to 'listen' to encrypted communication all over the world. Immediately after the disclosure Diginotar stopped issuing certificates and a while after that went out of business, which is a logical outcome when a CA loses all of its credibility.

The hacker that took responsibility for the Diginotar breach, also known as ComodoHacker, has performed attacks against multiple certification authorities, with the biggest one being against Comodo and some of its resellers, which gave him the mentioned name, that led to issuance of certificates for sites owned by mozilla, google and microsoft.

A very interesting case is Flame. Flame was a very widespread malware with a lot of attack vectors and impact. The interesting side of it, regarding TLS-x509, was that its creator had somehow acquired a Microsoft certificate and were in a position to sign binaries in order to pass fake windows updates to windows machines in the network and spread the Flame infection.

One of the latest trends in TLS encrypted traffic inspection is interception. Although it is not an actual vulnerability there are ways for internet providers and software and hardware creators to intercept the end users traffic.

The latest case of that type of surveillance attempt was by Lenovo and was uncovered in February 2015. It was driven by the need for Lenovo to serve ads in user's machines. To do that the company installed a root certificate of its own making in the user's local trust store and, through a local proxy, man-in-the-middled all traffic (encrypted and confidential or not). The inexperienced user had no way of knowing the traffic was intercepted as, from the user's perspective, all the traffic was encrypted but with lenovo's interceptor as a first destination. After that it was uncovered that the root certificate was the same for every machine and the TLS capabilities of the interceptor were really weak, which made the "encrypted" communication vulnerable to third parties too.

The latest incident of that kind happened less than a month ago (September 18th ,2015) during which some test certificates, issued to Google, Opera and other companies, by Symantec got out of hand. Although the impact of this incident was not serious it is mentioned because it wrote the first proof of usability for the advanced measures that are being taken lately on top of TLS, more specifically it was Certificate Transparency that is explained in the next chapter, to improve on it and improve auditing and monitoring capabilities.

A latest attack vector against TLS, although it is purely academic and no recorded attacks have been made, is by exploiting the BGP protocol which is used to "operate the internet".

By hijacking the BGP protocol an, admittedly very strong, attacker can change the internet routes used to pass internet traffic. That way he can fake owning a domain and issue a perfectly valid certificate by a perfectly valid CA for that domain. Although the possibility of that kind of attack is remote, it can only stress further the need for improvement of monitoring of TLS infrastructure so fraudulent certificates and actors are uncovered within short time periods to minimize the damage they can do.

It is obvious that there are, and there will be, many loopholes that attackers can use to exploit TLS and eavesdrop on encrypted communications or even impersonate other entities or authorities. As TLS becomes a default means of communication over the internet and these communications become an inseparable part of people's everyday life and transactions attackers are going to avidly exploit newer and newer vulnerabilities.
The above fact dictates the provisioning and further research of addons to TLS to further strengthen its security aspects and prepare for immediate remediation in case of failures, which in turn have to be found out in the least time possible.

Despite the whole lot of attacks against TLS the most important problem for the casual users are wrong configurations on browsers and web servers that allow attacks to be possible by adversaries. Continuous auditing of CAs to stay sure that nothing has been compromised (willingly or not / internally or not) takes time and a lot of resources. Also, because of the many implementations of TLS out there it is really difficult to be able to audit the security of all of them. Even for the most used implementation, openSSL, it took a major hit to be able to collect funding and afford serious code reviews and general security audit. That leaves us with what appears to be the most "boring" part, the configuration. In order for TLS to be able to provide its full potential it has to be configured extremely carefully and .. as possible.

Manual configuration of (not only) web servers is of crucial importance to the unproblematic operation of TLS. The variety of servers and most importantly the many different configuration options around TLS make things really complicated. Administrators need to follow best practices and keep their servers up-to-date not only in terms of software updates but also latest advances, protocol and algorithm vulnerabilities.

Right around here comes another problem. Because all the internet depends on it, the TLS ecosystem has become heavy and cumbersome. That creates problems when algorithms get obsolete and best practices have to change.

Backwards compatibility is deeply woven in the TLS protocol in the form of algorithm renegotiation and even TLS version fallback. That might enable old clients and servers to

communicate but leaves open huge doors for algorithms and protocol vulnerabilities to be exploited even years after they might “appear” to have been obsoleted.

In the next Chapter we review the measures that are being taken to keep TLS security strong. The mechanisms and processes that help achieve that will be described so the reader can gain insight in where the ecosystem is guided.

Steps Further

The vulnerabilities that are constantly uncovered around TLS and the mainstream nature of the protocol, it has become the de facto protocol for encrypted communication over the internet, have pushed organisations and communities to implement counter measures that will aid in strengthening the ecosystem around TLS. The Snowden leaks on the NSA programs regarding breaking encrypted TLS communications have made an even greater impact in the community and dictated that all parties involved and in charge of securing TLS take a step forward.

There are a lot different perspectives in the measure that have to be taken and the mechanisms that are being developed. Some parties implement solutions that work along TLS in the protocol stack and strengthen it. Others implement mechanisms further up the stack from TLS to be able to monitor the ecosystem and provide additional measure to secure TLS users from eavesdroppers. Those believing that the first step to a better, more mature, TLS ecosystem is widespread adoption and becoming the default means of communication, by making all HTTPS traffic secure, go forward to provide free certificates and secure configurations to the world. On the other hand there are others that argue that the CA model, on top of which TLS works today, is broken and must be replaced with something more distributed, less prone to weak links and something that any user can tailor to her needs.

Some solutions from all of these categories are presented in the following units.

Protocol Stack solutions

The solutions described in this chapter implement protocol addons for TLS and provide, almost transparent to the end user, more secure communication.

HTTP Strict Transport Security (HSTS)

HSTS is a security mechanism that allows the web server to force the clients that interact with it to do so only using HTTPS traffic and no insecure HTTP. This is communicated by the server to the client within an HTTP response, more specifically in the "Strict-Transport-Security" header of the HTTP response. The header contain the HSTS policy the client must follow in order to

communicate with the server. This mechanism eliminated the surface for SSL stripping attacks and fallback attacks that are a great problem for TLS implementations.

HTTP Public Key pinning (HPKP)

HPKP is a mechanism that enables clients to “pin” certificates to certain domains. That means that the client can match a domain with the certificate it trusts for that domain, which is communicated by the server when the client visits a certain domain. That way if the domain or certificate are spoofed the client will see the fraudulent or mistaken activity and report it. This mechanism can help mitigate man-in-the-middle and spoofing attacks. On the downside, it’s trust on first use nature expects that the first time a client visits a certain domain it is communicating with the valid web server and no malicious behaviour is conducted.

TLS Addons

The mechanisms described in this sections act as additions on top of TLS to provide further insight and capabilities to the ecosystem.

Certificate Transparency (CT)

Google’s CT is an experimental protocol that works as a distributed certificate issuance logging system. It is based on append-only logs that contain certificate chains. When a certificate is issued the CA and the certificate holder are expected to write that certificate in the logs. In that way every valid certificate is logged and can be found in the logs. Clients can request certificate logging info and expect that every certificate can be found in logs. Also if fraudulent,invalid certificates get into the logs to pass themselves as valid, CAs can find them quickly and easily by constantly auditing the logs.

DNS-based Authentication of Named Entities (DANE)

DANE aims to provide another layer of security to the TLS protocol by associating certificates with DNS names through utilisation of the DNSSEC protocol. It enables the administrators of a domain to store certificates in DNS Records and sign them with DNSSEC. That way there is another barrier against man in the middle and spoofing attacks.

Certificate Authority Authorisation (CAA)

CAA works alongside DANE to provide CA authorisation for domains. It allows domain administrators and owners to restrict certificate issuance for their domain to certain CAs, thus eliminating the chance of fraudulent or accidental double certificates being issued by other CAs.

Replacing existing models

Many argue that trusting certain entities with all the power a CA has is not safe, and their assumptions are backed up by the numerous incident of CA breaches that have been recorded. Many systems for replacing the CA model have been proposed, although none of them has been heavily used out in the wild, 3 of which are described in a little more detail in the next paragraphs.

Convergence

Convergence hopes to replace the classic hard-coded, immutable CA model. It encapsulates the concept of Notaries which act as trust entities and make decision for the users that trust them. The system is highly distributed as anyone can create a notary and the system allows you to trust as many notaries you want and the results are based on the consensus of all your trusted notaries. This system offers the security that lies in the fact that a malicious or compromised notary cannot harm the whole system.

Perspectives

The perspectives project build on the same foundation as Convergence. It, too, uses notaries to provide insight and trust.

DNSChain

This mechanism, offered by okTurtles Foundation, build on the concept of the blockchain which powers cryptocurrencies, the latest trend in online currency. It claims to solve many of the problems that are not addressed by TLS, x509 and their addons described in previous chapters but it is in its very early steps.

All of the above proposed alternatives to CAs offer a more free and distributed secure web where, based on the concept of trust, even self-signed certificates can be trusted throughout the whole web.

Making HTTPS/TLS easy and really secure

Let's Encrypt

Let's Encrypt is an initiative by the Internet Security Research Group to provide free domain validated certificates to everyone who needs it. It essentially is a CA, under EdenTrust, that is going to provide free, fully valid and trusted certificates. The most important thing though, not that the free certificates are not important, is that the process of obtaining and installing a certificate is going to, ideally, be fully automated through trivial command line tools running on the administrator's server. That is going to provide valid certificates and strong configurations to anyone, even admins that are not TLS experts. Let's Encrypt's first certificate has already been issued and they are planning for full rollout during the end of 2015.

CloudFlare Universal SSL

CloudFlare is a security oriented content distribution and DNS company. Their Universal SSL program provides TLS encryption, with really high configuration standards, for free to their customers. That is achieved by utilising SNI connections (hosting more than one domain from the same IP) and including multiple domain names in the same certificate. That way they essentially provide encryption from the client to the Cloudflare hosted part of the website, which is enough to keep casual eavesdroppers from attacking, and not for the part connecting Cloudflare to the domain itself. To acquire full protection, the domain and website owners have to acquire a valid certificate from a CA and configure TLS on their servers.

TLS server configuration best practices

As said before the most important part of doing TLS right is getting the configuration right. That might be simple, or not, for someone who has knowledge of the TLS protocol but that cannot be expected by all the administrators and web masters. To make up for that mozilla and SSL labs are publishing best practices for TLS deployment and update them regularly.
In addition, Mozilla has an automatic configuration generator, that follows its best practices, that can create TLS configurations for multiple web servers (NGINX, Apache etc). SSL Labs the most trusted TLS scanner for domains, can provide scores on TLS configurations and guidance on how to fix any flaws found.

Algorithm Deprecation

With the lead of Mozilla and Google, and the help of all the community, several attempts are made to move on to new, more secure, algorithms and abandon broken , or soon to be, old ones. The latest such attempts are those to deprecate RC4 and SHA1 with the first known to be vulnerable to specific attacks and the latter predicted to be broken in the next years, due to increase in attacker computing resources.

TLS-Observer

Problem Definition and drivers

TLS infrastructure requires continuous monitoring for any organisation to be able to respond to incidents that come up, from bugs in specific implementations and software to root CAs or certificates being compromised or even backdoored.

TLS-Observer is a framework that provides real-time and distributed compliance auditing of TLS configurations amongst the vast network hosts of an organisation. It can help the system administrator have an overview of the compliance level of their systems and even help them identify deficiencies and reach a better level of security. It can also help incident response by crawling the network in real time for specific vulnerabilities.

Building on Go's out of the box concurrency, TLS-Observer offers a fast and reliable framework that can be used in all kinds of organisations and can also aid researchers get an overview of the TLS space all over the Internet without having to crunch all the data themselves.

The need for automated tools to assess security and compliance levels is not a new one. This also applies to the TLS and x509 ecosystem. One can find a great variety of tools to test each and every aspect of the security level of any TLS server web server.

When coming to TLS security there are many tools one can use. From open-source to closed source, commercial ones. The current defacto tools for testing TLS infrastructure is Qualys' SSL Labs scanner, which is operated by TLS expert Ivan Ristic. It offers a full blown TLS security scanner updated with specific scans for latest vulnerabilities and best practices. Unfortunately, it is closed source, solely maintained by Qualys and has limitations in usage (not being a commercial product). -- Maybe describe why open source is important (extendability, not depending on anyone, can be tailored to every need)--

When creating TLS-Observer our goal was not to create the best TLS security assessment tool (although we actually created a tool) but to create a dependable and high-end framework for TLS security testing and monitoring. Our main goal was for our tool to be able to provide real time monitoring of TLS infrastructure and alarming for high risk vulnerabilities found to aid in incident response. Another goal was for the tool to be highly extendable. That means that anyone can write a module to scan for a specific vulnerability, algorithm or anything he pleases and that can be done without having to acquire any knowledge about the internal workings of the TLS-Observer.

System Architecture

Our main goal when designing the system was to build a fast and extendable tool that would provide deep and useful insight of TLS infrastructure.

We decided to use Go as the development language to take advantage of its built in support for concurrency through the use of channels and goroutines, which meant that we would not get into the trouble of taking care of all the concurrency needs ourselves. One can think of Go-routines as lightweight threads managed automatically by the go runtime. They actually operate within a single thread and the automatic management makes it effortless and really chip to spawn thousands of goroutines without caring about performance overhead.

Also Go, despite being in its very early years of development, comes with a great set of cryptographic packages and libraries that could easily support our use case of TLS validation and inspection.

We decided on a distributed architecture where each module does only a pretty specific job, can work independently from all the other modules and publishes its results. To support the distributed nature of this architecture we use the AMQP messaging protocol which enables the modules of our framework, which essentially is a standalone application, to communicate asynchronously with each other, share results and distribute the workload.

In the AMQP model messages from publishers are published to certain exchanges which act as routers. These exchanges distribute message to queues by categorising them using certain rules called bindings. Message consumers subscribe to the queues that are of interest to them and consume messages from them.

In our implementation we used the RabbitMQ server for message communication with a go amqp library (https://github.com/streadway/amqp) implementing the client.

By utilising asynchronous messaging and a distributed architecture we can extend the framework without being burdened by any dependencies. That way new modules, addons and extensions can be developed easily, independently and in any language. That enables us to build modules by utilising already successful tools and not having to implement all of our future extensions from scratch.
The distributed architecture also allows us to have a systems that can scale horizontally without having to change anything in the code to improve performance. At any time, you can just spawn a new server, run the software and let it consume messages from RabbitMQ queues.

Our 4 basic modules are certRetriever, certAnalyzer, tlsRetriever and tlsAnalyzer. The retrievers handle all the connection part and retrieve certificate and connection info accordingly. They turn

that information to readable format and pass it to the next modules by publishing to the rabbitMQ exchange (the certificate modules publish with the routing key cert_scan_results and the tls ones with conn_scan_results). The analyzer modules consume from the aforementioned queues and perform a basic analysis of the retrieved information. tlsAnalyzer checks if the connection is TLS enabled, processes the available ciphersuites and analyzes every attribute of the TLS connection. It then converts its results to a specified JSON format, ideal for storing and optimised for indexing and searching in the database. The certAnalyzer receives the certificates that were published by the analyzer and checks their validity. It processes all the available certificate chains and analyzes problems in the validation chain, if any. Lastly, it distills the information and outputs a certain set of results that were chosen as most useful for indexing and further research. The results of the certAnalyzer are also published as JSON messages to the DB.

Both analyzers, in addition to pushing their results to the database, also publish their results to other queues that are consumed by minimal workers performing specific research analysis, for example searching for certificates with really long validity duration, searching for connections with deprecated algorithms etc.
These workers also perform a great part of offering incident notification and fast response. If any alarming results are found the output of the workers can be pushed to the central mozilla incident response platform and an alarm is raised. These workers are easily extendable, as is the whole created framework, and can easily be developed in any language with just minimal support of the AMQP protocol.

For the storage part of the framework at first we decided to use a standard SQL database, so we went with postgres. Although the performance was more than enough the way to use it was not so intuitive. The problem was that we had to change between native JSON, that we are using throughout the framework to transfer and store information, to structs that represented the database schema we had decided on. That led us to change our decision and go with Elasticsearch.

Elasticsearch is a Lucene-based NoSQL database and search server. Being one of the most popular enterprise text search engines makes sure it is enterprise grade dependable software. It saves entire JSON documents and can provide near real-time data searching. Through its powerful indexing mechanisms it provides great performance for searching through text based JSON documents. Building on the distributed architecture Elasticsearch is based on it provides scalability and high-availability, so it enables us in our goal for our system to be able to scale easily and horizontally. In a typical elasticsearch setup you have to create specific indexes for the data to be stored inside. In order to fully utilise searching you have to provide a specific mapping for each index. Mappings the way you want the data to be stored in the database, for example you can select to keep strings in a tokenized and non-tokenized form (elasticsearch by default tokenizes string fields, which is not helpful if you want to search for whole sentences etc). Indexes can encapsulate different types to hold data of different nature. In our setup we used 1 main index , observer, and 2 types inside it, certificate and connection which store information

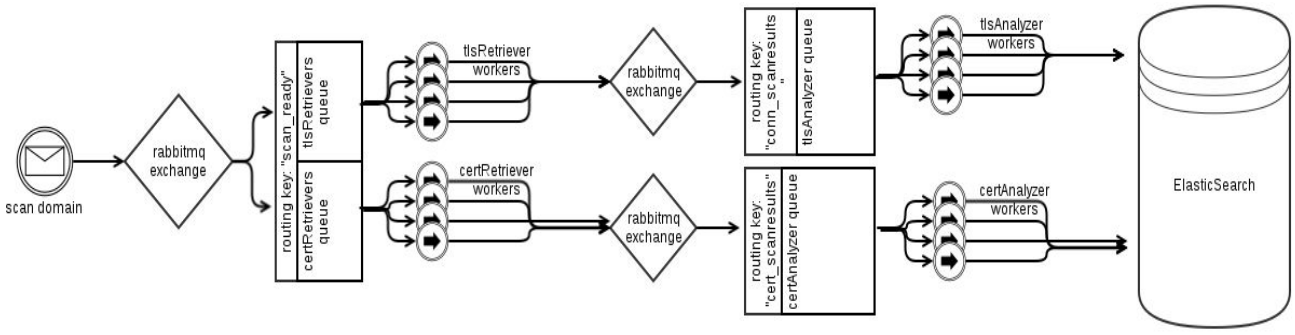
for the according scan results of domains. The mappings used for each index can be found in the project's public repository on Github.

Another perk of using Elasticsearch is is accompanying software. We get out of the box support for Kibana. Kibana is a flexible analytics and visualisation platform. It supports real-time analysis and visualisation of streaming data. Building on Elasticsearch's fast responses it provides a really intuitive and sophisticated analytics visualisation platform without having to build anything from scratch.

In order for past TLS scanning information about web servers and domains to be available for reference and comparison we have implemented internal mechanisms for document versioning. We include timestamps with every scan so we can be able to build timeseries of data easily. Every time we scan a domain we check if we have an existing entry for it in the database. If there is one there we check if there is any difference

In order for the system to work, input must be provided for analysis. That can be done either by programmatically feeding domain names to the *_scan_ready queues (the retrieveTLSinfo tool, found in our repository, already does that) or by using the provided web API which is a part of the framework.

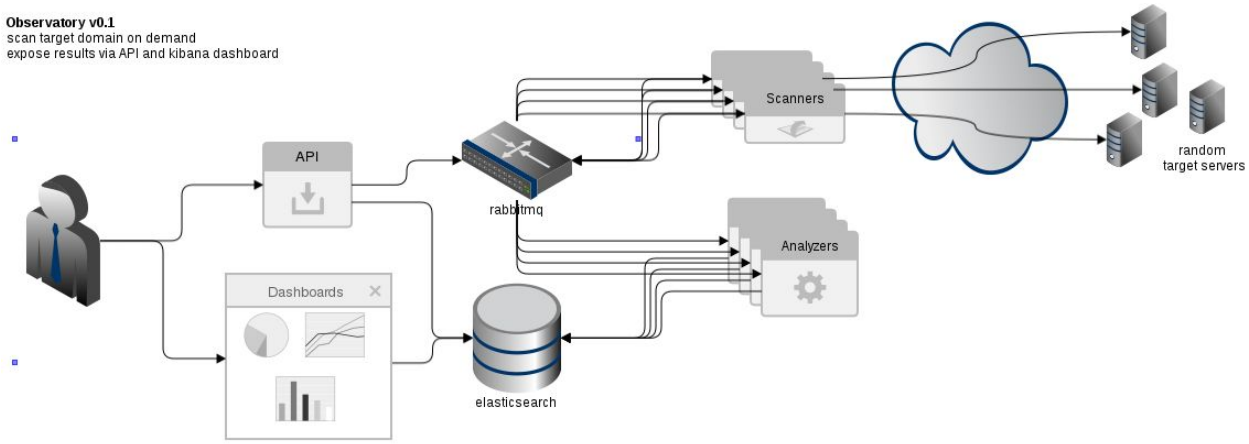
The below image summarises the main architectural design of the framework, showing the flow of messages through the above described modules and software.



AMQP architecture diagram

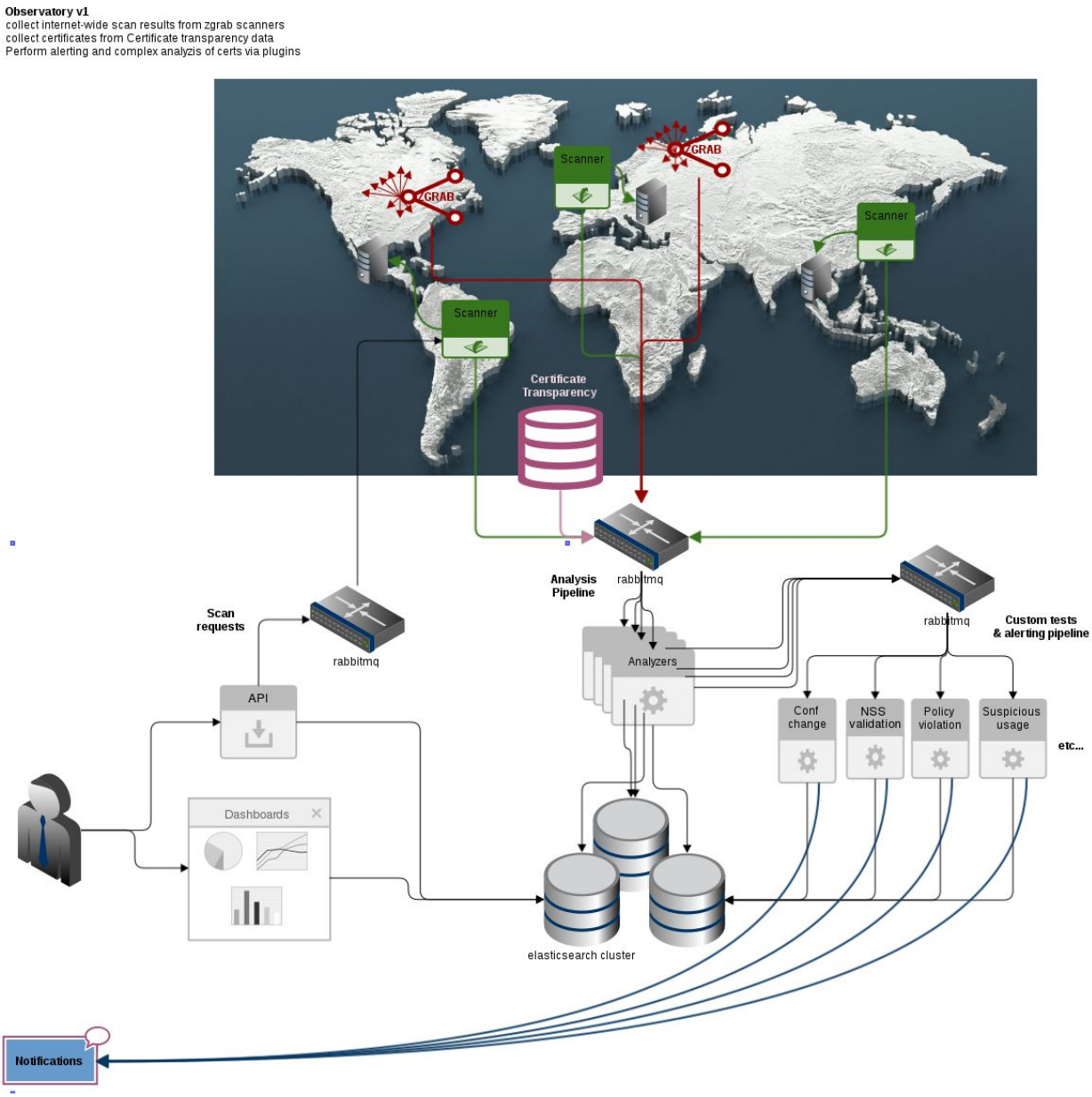
Usage Models

TLS-Observer has many usage models. That is granted by its highly abstract nature. During the first releases and tests we had basic retrievers and analyzers for TLS connections and certificates which could be triggered either by the API or programmatically (as described extensively in the previous chapter). After that the users could check and visualise aggregated results from the Kibana dashboards. A visualisation of that usage scenario can be found in the image below.



Observatory v0.1 high level overview

Although the, up until now, presented system is versatile and scalable there is the barrier of the single methods of input and result output. Results could only be viewed through the Kibana interface and only programmatically fetched through the ElasticSearch REST API for further processing. That is what led us to the next design.



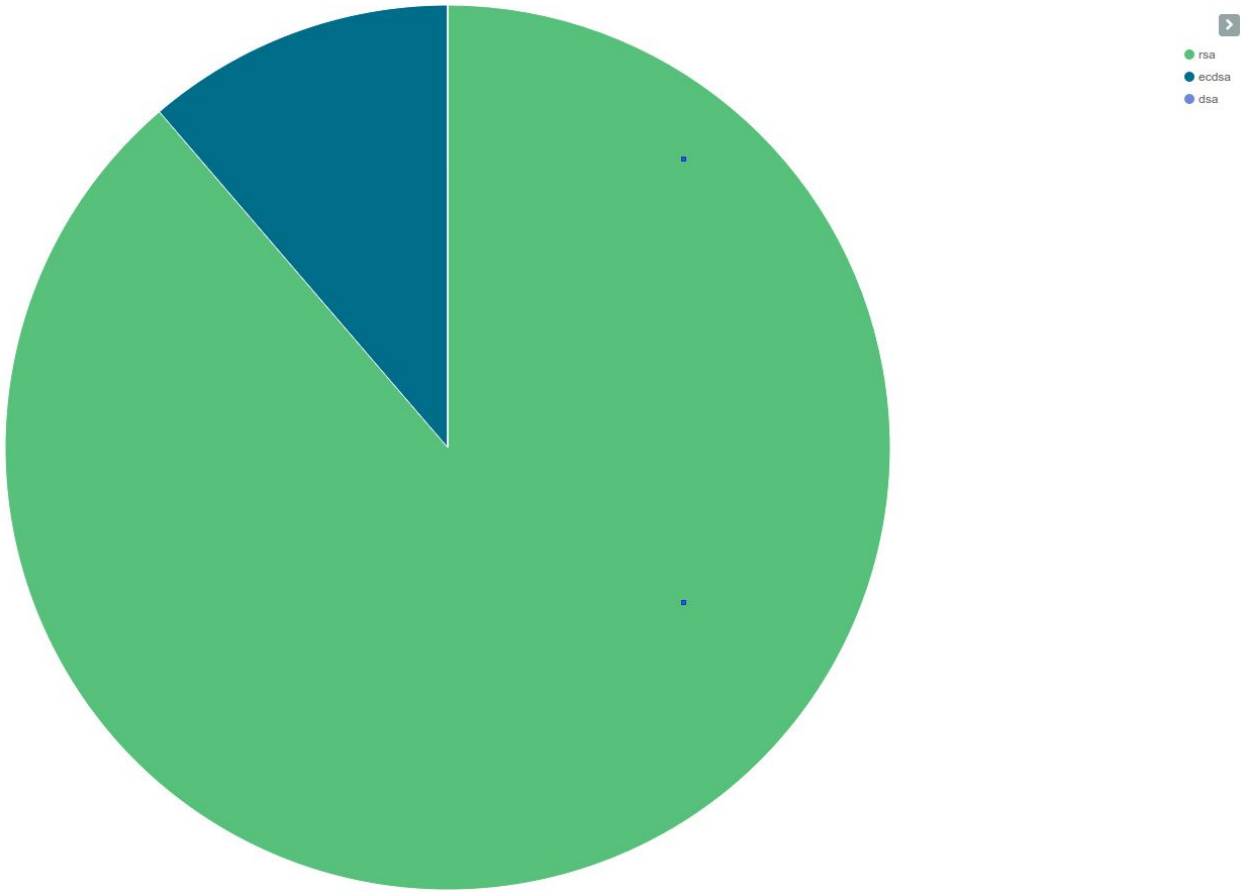
Observatory v1 high level overview

The above picture shows the complete usage scenario of TLS-Observer. As you can see the input is provided, not only through the API but also, through zgrab files, which can provide all the domains found in the Internet HTTP space. Scan inputs are processed and the results are aggregated with results from third party tools , eg Certificate Transparency, and sent for archival and are available for analysis and visualisation. Meanwhile raw scan results are passed to plugin workers that are designed to do specific jobs, for example trigger alerts for suspicious usage or scan for a specific type of vulnerability. Those results are also aggregated and archived for later analysis.

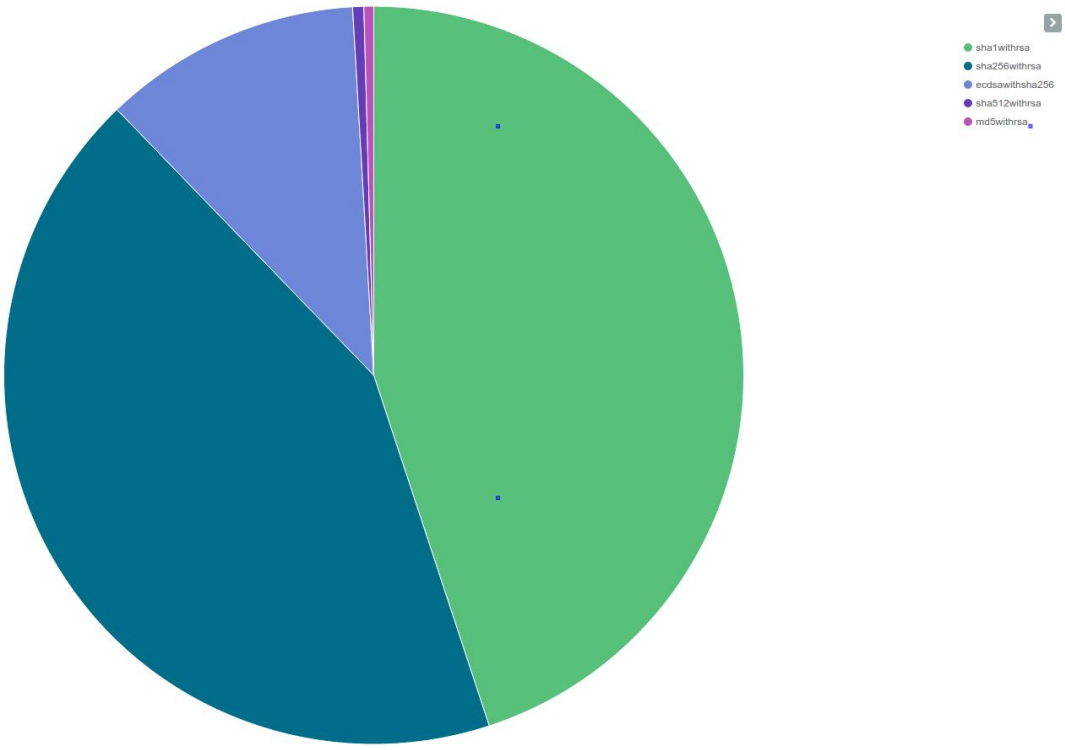
In the way described above, TLS-Observer, can provide a fully automated real-time analytics, auditing and alerting framework for TLS infrastructure on a worldwide scale.

Visualisation & data analysis showcase

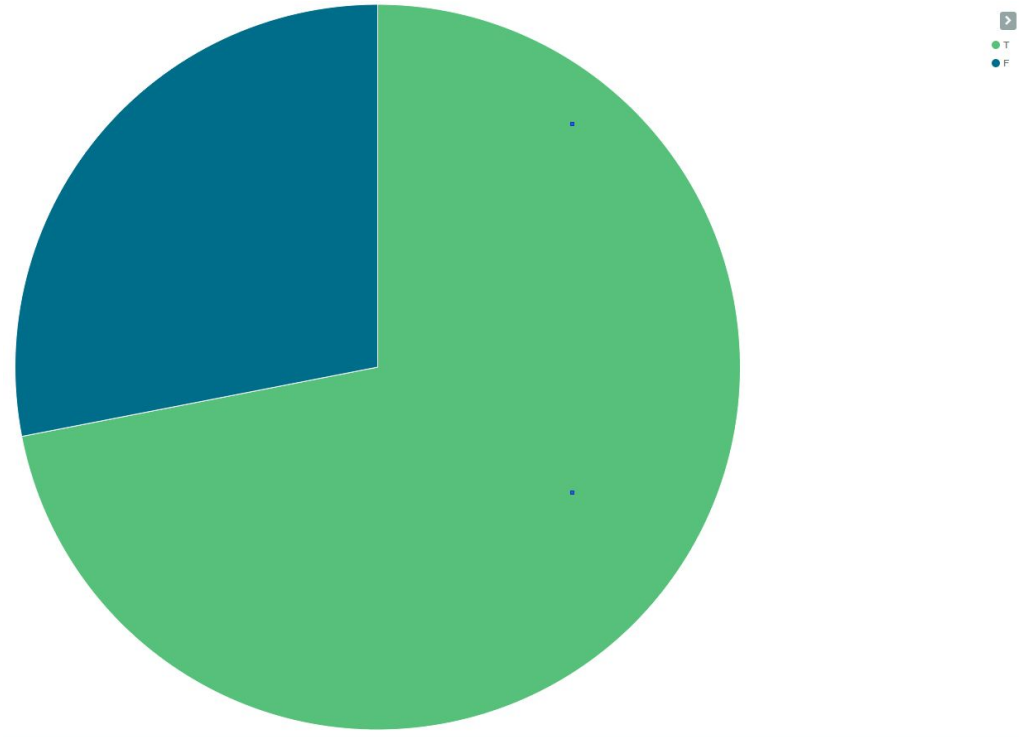
The below images were taken from the Observer UI and demonstrate the data insights it can provide for TLS certificates and connections.



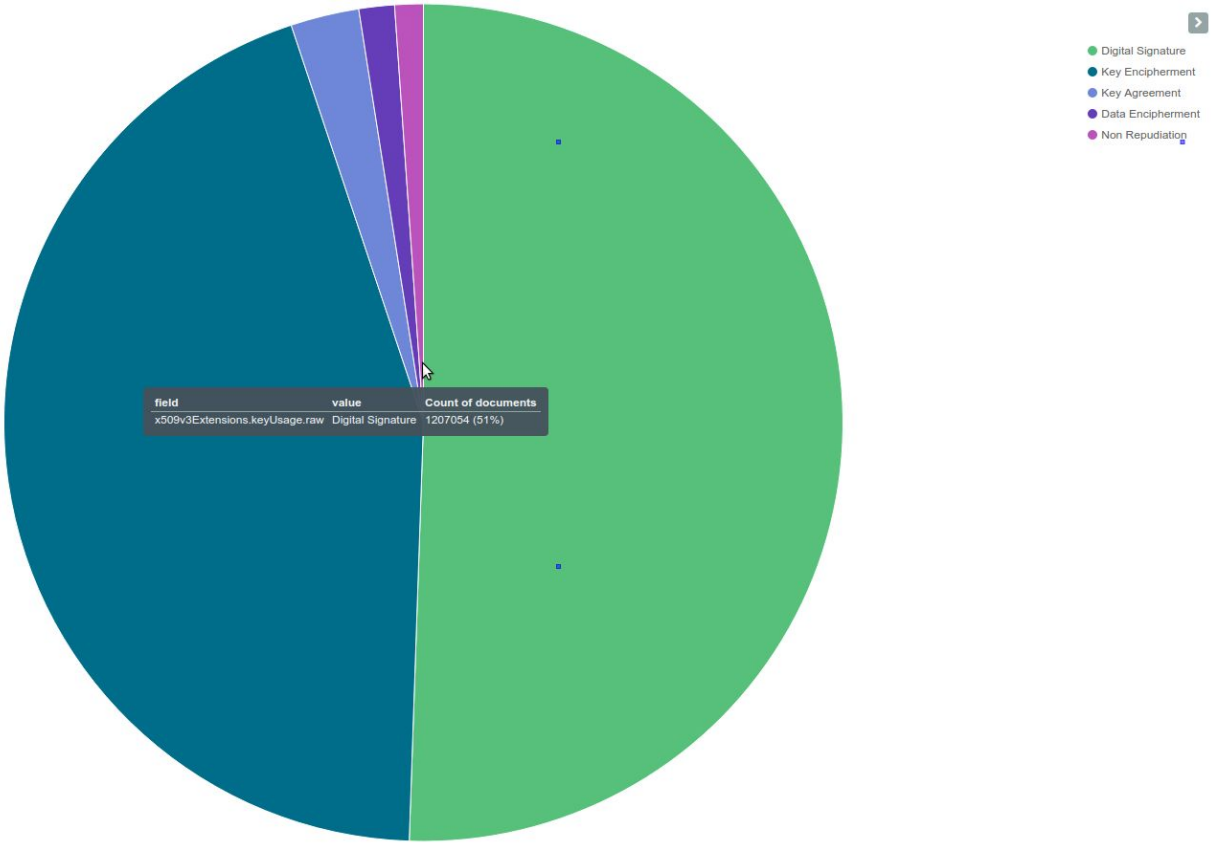
Public Key Algorithm Overview



Signature Algorithm Overview



Server Side Ordering Overview



Key Usage Overview

Bibliography

[1] Algorithms, Key Sizes and Parameters Report - https://www.enisa.europa.eu/publications/algorithms-key-sizes-and-parameters-report

[2] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile - http://tools.ietf.org/html/rfc5280

[3] TLS for the pragmatic - https://jve.linuxwall.info/blog/index.php?post/2014/11/20/SSL/TLS-for-the-Pragmatic

[4] OWASP Testing for Weak SSL - https://www.owasp.org/index.php/Testing_for_Weak_SSL/TLS_Ciphers,_Insufficient_Transport_Layer_Protection_(OTG-CRYPST-001)

[5] Summarizing Known Attacks on TLS and DTLS - https://tools.ietf.org/html/rfc7457

[6] Certificate Transparency - https://tools.ietf.org/html/rfc6962

[7] SSL Pulse - https://www.trustworthyinternet.org/ssl-pulse/

[8] Bulletproof TLS & SSL, Ivan Ristic, 01 August 2014