ASCEDS White Paper

Maintained by: Florin B. Manolache
florin@andrew.cmu.edu

I. Introduction

This is a white paper based on [1], describing the structure and the functionality of the Automated SSL/TLS Certificate Distribution System.

There was a time in the early days of the Internet, when all the communication online was clear text, nobody was talking about encryption. In the late 90's, traffic sniffing went rampant, credentials were harvested, ssh made its debut to replace ftp and telnet with encrypted channels [2]. While encryption was useful, nobody had any quick way to verify who sits at each end of the communication. So black hats got creative, developing a range of other tricks, like man in the middle attacks [3], for performing active eavesdropping in their pursuit for credentials. These attempts were promptly countered by the appearance of Certificate Authorities (CAs) which introduced trust by signing encryption certificates, this way vouching for the identity of the entities using them. But who can vouch for the CA? Well established and well known companies like Microsoft, Google, Mozilla, Apple, started root programs [4], including CAs' own certificates into their software, to establish the much needed trust into a list of CAs. Since getting into a root list is a very slow process because it depends on the software upgrade cycle which can take decades, newer generations of CAs were vouched by older CAs, transforming certificates into certificate chains.

Typically, a CA signing process is based on a Certificate Signing Request (CSR) which contains data about the entity using the certificate and other information like Subject Alternate Names (SANs) for servers which run virtual host names. All this info, together with validity and lifespan data, are incorporated into the signed certificate, the final result consisting of a private key, a public key, and a chain of public keys belonging to the series of CAs enlisted by various root programs or vouching for each other [5].

The SSL/TLS certificates are very popular in services running on modern computing devices. One can easily observe the following trends:

- certificates were originally used mainly by web servers distributing private contents; currently they are used by most services with network exposure: e.g. all web servers, protocols involved in messaging, email handling and delivery, even ssh authentication;
- root programs are forcing shorter lifespans for the signed certificates onto CAs. (currently no more than 13 months enforced by Apple in 2020 [6]); shorter lifespans allow for more efficient encryption protocol upgrades, and for maintaining trust through more frequent checks of the identity.

Historically, certificates were signed by a CA as a paid service, being a significant expense for small companies or individuals. After various attempts of offering access to free or cheap certificates made by various entities, the standard was set by Let's Encrypt [7] by providing short lived (90 days) certificates which can be renewed through the Automated Certificate Management Environment (ACME) protocol [8]. ACME was implemented by Electronic Frontier Foundation (EFF) through a software named certbot [9] which is python-based, thus available for most devices and operating systems.

While Let's Encrypt certificates are really the way to go for small companies or single servers, the system is pretty much unusable for an enterprise environment. The main problems experienced by the authors were connected to:

- the lack of modularity to help extend certificate usage to new services;
- major problems when certbot was enforcing certain configuration paradigms onto the apache web server, which won't play nicely with a large number of virtual hosts running various software packages;
- limits over how many certificates can be generated per month.

But Carnegie Mellon University had access to the InCommon CA [10] which offers an ACME interface working quite flawlessly with certbot. However, all the certificate management operations through ACME were confined to a single server and protected by a set of credentials. Our solution was to create the Automated SSL/TLS Certificate Distribution System (ASCEDS) which is presented in this paper. ASCEDS automatically generates/renews certificates and distributes them to any number of servers on the network. It also allows for manual generation/renewal/revoking of certificates through a web interface and a CLI.

The ASCEDS design is presented next. The following sections describe different typical cases of using ASCEDS, while showing the role of utilities included with the software.

II. ASCEDS Architecture

The main design objectives of ASCEDS were the following:

- simple structure: one certificate manager governs certificates for all the servers in the authorized domains, organized in a site;
- efficient operation: the renewal process should be as transparent as possible, and should involve little or no human intervention;

- flexibility: services using certificates can be easily added or removed, and their configuration scripts can be customized in a way which is robust to ASCEDS upgrades;
- accountability: multiple users can be configured on the certificate manager interface, each user having authority over a different set of domains;
- friendly functionality: the interface should offer step-by-step guidance and feedback for users unfamiliar with ASCEDS;
- completeness: the interface and the scripts should offer modular and flexible solutions for most of the operations regularly performed with certificates.

The functional structure of ASCEDS shown in Fig. 1. All the servers requiring certificates are grouped into a site governed by a certificate manager which is authorized to get certificates from a CA. The generate/reconfigure/revoke certificate manipulation is performed by the certificate manager based on secret tokens issued by the CA, and valid for a set of domain names. The certificate manager communicates with the CA externally using ACME through certbot. The set of authorized domains is managed by the *asceds-authorized-domains* script.

Inside the enterprise network, the certificate manager communicates through ssh/scp on an asceds local account with the clients having ASCEDS installed. Communication consists in:

- pulling the certificate configuration file *cert.conf* from the client, containing information about FQDN, SAN list, certificate encoding, and type of client;
- pushing the newly generated or renewed certificate files;
- pushing the updated site configuration file *asceds-site.conf*, as described in Section III.

Three types of clients are supported, offering different levels of automation:

- fully managed: ASCEDS installed and configured on the client, the certificate update process happens automatically;
- privately managed: while the software is installed and configured, the client needs to be put in full ASCEDS access mode to propagate certs, i.e. the asceds account must be accessible in r/w mode from the certificate manager;
- unmanaged: no ASCEDS installed on the client; certs and updates are propagated by hand by the sysadmin of the client (e.g. NAS, printer, switch, Windows/MacOS).

Most of the automation is realized through cron scripts triggered at configurable intervals, both on the certificate manager and on the clients.

ASCEDS contains a set of shell scripts which are self-consistent and independent, such that they can be recombined in any number of ways to perform the certificate management, or to fix broken/inconsistent states of certificates. All the ASCEDS scripts have a "-h" option which lists a manual page. There is a stand alone *asceds* command which describes the structure of the software, lists the local computer configuration as a client, and extracts any recent warnings or errors from the log files. For certificate managers, options of *asceds* allow listing of details and status for the clients based on different criteria, as client type or substring matches in their FQDN.

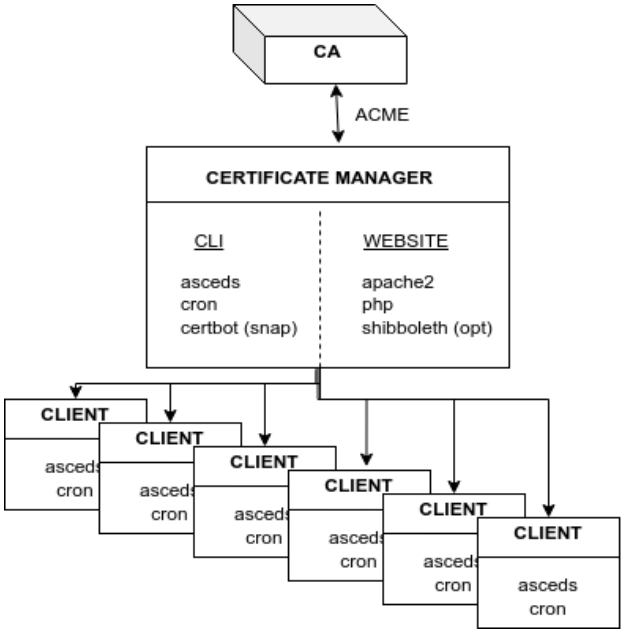

Fig. 1. ASCEDS architecture.

All the scripts start by executing a set of configuration files provided by ASCEDSCONF in the order specified by the variable. First, all the default settings are loaded from prototype files distributed with ASCEDS, then the corresponding custom files from /etc/asceds/ overwrite the defaults.

To support large environments with multiple units which don't share trust, ASCEDS starts a php web user interface on the certificate manager. The interface should run under the root of a virtual website, and was tested with the Apache webserver. A multi-user environment tested with shibboleth and with the Apache native simple authentication is offered. Each user has authority to generate certificates for a certain set of domains, which can be configured through the *asceds-web-user* script.

The web interface works based on a set of request queues for generating certificates for unmanaged clients, revoking certificates, and creating/propagating certificates for managed clients. The requests in the queues are logged, and can be executed immediately by ssh, or later on by the cron jobs in asceds-web-actions. This mechanism ensures that requests which cannot be performed immediately will be re-tried later until their execution is successful.

The next section presents the workflow of several typical scenarios implemented in ASCEDS.

III. ASCEDS Site Management

An ASCEDS site has one certificate manager and a set of clients as described in Section II. The certificate manager generates the site configuration file /etc/asceds/asceds-site.conf during the initial setup, as described in Section IV-A. There

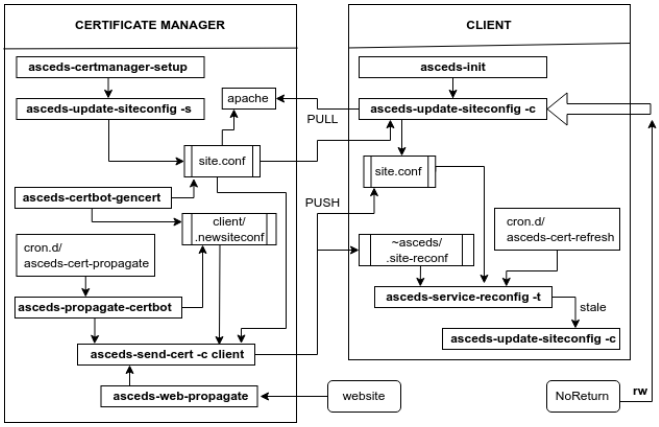

Fig. 2. Site configuration file distribution.

are two distribution mechanisms to share the site configuration file with the clients: pull from the client through wget and push from the certificate manager through scp. The diagram for the site configuration file distribution mechanisms is shown in Fig. 2.

For the pull mechanism, the site configuration file is symlinked to the public directory of the certificate manager website, and accessible only to the domains served by the certificate manager. During the initial setup of each client, the site configuration file is retrieved from the certificate manager's website as described in Section IV-B. Pulling the site configuration file is also useful for the privately managed clients, or cases when pushing doesn't work (e.g. the asceds ssh keys on the certificate manager are changed).

The site configuration file can be updated using *asceds-update-siteconfig -s* and pushed to the clients. The following information is distributed with the file:

- the alarm email address ALARM where fatal errors are sent (typically a ticketing system queue); if ALARM is empty, no emails are sent;
- the certificate manager hostname ASCEDSCRTMGR;
- information about certificates being self-signed or generated by a CA, which should reflect into wget arguments;
- the asceds account ssh public key ASCEDSSSHPUBKEY used to create authorized_keys files on the clients, to facilitate pushing updated certificates and site configuration files;
- the list of certificate encoding algorithms ASCEDSENCLIST offered by the certificate manager.

The site configuration file has an automatic push distribution mechanism which takes care of propagating an updated version to the known managed clients. This mechanism works similarly to propagation of renewed certificates, described in Section IV-C. When *asceds-update-siteconfig -s* is used on the certificate manager, new site configuration triggers .newsiteconf are created for all the managed clients, except the certificate manager itself, and placed in ∼asceds/⟨managed_client⟩/. The new site configuration file asceds-site.conf is copied in the same locations. Then, cron runs *asceds-propagate-certbot* which pushes the new site configuration file to the fully managed clients, creates a .site-reconf trigger in ∼asceds/ on the client, and deletes .newsiteconf. The cron driven *asceds-service-reconfig -t* on the client completes refreshing asceds-site.conf and deletes the .site-reconf trigger. The privately managed clients must run *asceds-update-siteconfig -c* by hand to refresh the site configuration file as shown in Fig. 2.

The *asceds-update-siteconfig* script is used to handle the site configuration file both on the certificate manager ("-s" flag) and on the clients ("-c" flag). On the certificate manager, it performs the following actions:

- creates/updates the site config file asceds-site.conf; the name of the certificate manager can be changed only if *asceds-update-siteconfig* is ran from *asceds-certmanager-setup*;
- if certificate manager name is a CNAME, checks if it resolves to the same IP address, and adds it to hosts, postfix, openssl.cnf;
- refreshes symlink to make the site configuration file available by wget to clients;
- copies the new site configuration file into the managed client directories and adds the .newsiteconf trigger file to all the managed clients except cert manager itself;
- on the cert manager, copies the new site configuration file into ∼asceds/.

The certificate manager name used by the site can be specified through the variable ASCEDSALTCRTMGR (passed by *asceds-certmanager-setup -s*), through ASCEDSCRTMGR from an existing asceds-site.conf, or through terminal input, whichever method succeeds first.

On the clients, *asceds-update-siteconfig -c* performs the following actions:

- retrieves the site configuration file from the certificate manager webserver using wget;
- parses the file and creates /etc/asceds-site.conf;
- uses the site ssh public key from the new asceds-site.conf to create/update authorized_keys.

If the asceds ssh key is changed on the certificate manager, the ability to automatically push updates to the clients is lost. To avoid this situation, the cron driven *asceds-service-reconfig* script refreshes the site configuration file pulling it through *asceds-update-siteconfig -c* if the certificates in ∼asceds/certs/ are stale (expiring in the next 5 days), as described in Section IV-B. Migrating to a new certificate manager (different site name) can be accomplished only by running *asceds-init* by hand on each of the managed clients.

IV. ASCEDS Action Chains

Handling certificates typically consists of a small number of scenarios which involve executing a set of ASCEDS scripts, named here action chains.

A. Initial Setup of the Certificate Manager

The central role in the ASCEDS structure is played by the certificate manager. It can be installed by running the

asceds-certmanager-setup script as root on the server which is authorized by the CA to handle certificates. The "-s" option can be used if a CNAME is preferred for the certificate manager. Otherwise, ASCEDS sets the name provided by the "hostname -f" output.

The *asceds-certmanager-setup* script performs the following operations:

- checks/fixes the network configuration of the certificate manager;
- checks for certbot; if not found, it offers to use a fake certbot which generates self signed certificates, which is described in Section VI; otherwise, it requests a clean certbot install and exits;
- configures certbot credentials and authorized domains in the asceds-certbot.conf file;
- creates asceds user ssh keys;
- creates site configuration file using the *asceds-update-siteconfig -s* script as described in Section III;
- creates symlinks to certobot-related ASCEDS scripts;
- sets cron job asceds-cert-propagate for driving the *asceds-propagate-certbot* script to push renewed certificate files to ∼asceds/⟨client⟩/ and propagate them to managed clients through *asceds-send-cert*;
- prepares up the web interface.

Web interface preparation involves the following steps:

- gets info: organization name, website URL, request execution methods;
- customizes etc/config.php and asceds.php for the website;
- copies and customizes the ASCEDS site configuration file for Apache;
- initializes log file for web request history;
- builds the directory structure for the website;
- creates ssh keys for requests by ssh;
- creates symlink to make the site configuration file available by wget to clients;
- sets asceds-web-actions crontab for running the request queues generated by the web interface.

Next step is to run the *asceds-authorized-domains* script if this is a fresh install, otherwise the certificate manager rejects any request.

The website still needs to be protected by the desired type of authentication in the .htaccess file, then taken online in the Apache setup after inspecting the configuration files created by *asceds-certmanager-setup*. Before doing that, since the website handles sensitive data, it is recommended to set up the same computer as a client as well, as described in Subsection IV-B, and to obtain certificates for itself first.

After the website was taken live in the Apache web server, users can be configured through the *asceds-web-user* script. The script is configuring simple authentication if the file .htpasswd exists in the same directory as .htaccess. Otherwise, shibboleth authentication is assumed. The list of users and their domains is kept in the users.php website configuration file. Web interface user names must be valid email addresses or email aliases on the certificate manager. They are used for sending email notifications when various requests are completed by cron, and also when renewal certificates become available.

The web interface limits the ability of web users to request information about clients which are not in their domains, or to add SANs which are in such domains. However, web users can remove from the SAN list a CNAME which is not in their domains. Also, web users can revoke the certificate if the client FQDN is in their domain list, even if the SAN list contains a CNAME which is not in the user's domain list.

The *asceds-propagate-certbot* script detects new certificate files created by certbot, and copies them to a location accessible to ASCEDS for managed clients, or into the website space for unmanaged clients. A new certificate trigger is created as .newcerts in ∼asceds/⟨managed_client⟩/ for managed clients. The script is used to automatically propagate renewed certificates through the asceds-cert-propagate cron job, as well as to perform initial setup or configuration changes of certificates through the website requests and through the CLI scripts. The certificates are propagated to the fully managed clients by running the *asceds-send-cert* script as user asceds, or the requestor is notified by email about the certificate files availability for the other cases.

The *asceds-send-cert* script runs as user asceds. It sends the new site configuration file (if the trigger .newsiteconf exists on the certificate manager) to the clients by scp, and sets a site reconfiguration file trigger ∼asceds/.site-reconf on the client, as shown in Fig. 2. Then it sends the new certificate files (if the trigger .newcerts exists on the certificate manager) to the clients by scp, and sets a certificate reconfiguration trigger as ∼asceds/.asceds-reconf on the client. If the transfer is successful, the triggers on the certificate manager are removed. The clients use the reconfiguration triggers through the cron driven *asceds-service-reconfig* script which refreshes the site configuration file and reconfigures services with the new certificate as described in Subsection IV-A. Fully managed clients which are unreachable for pushing updates are flagged through files in ∼asceds/down, and an email message is sent to ALARM. When the client reconnects to the network, the flag is removed.

B. Initial Setup of a Managed Client

Initializing ASCEDS has the purpose to attach a managed client to the certificate manager. This is the most important action for proper certificate handling, and should be performed only once after a fresh install by using the *asceds-init* script. The "-s" option may be used to specify the certificate manager of the site.

The initializing procedure is the same for fully managed or privately managed clients. both categories will need to offer read/write access for the certificate manager to the asceds user home directory for the duration of the procedure. The privately managed clients should use the "-r" flag for denying further certificate manager access after initialization is completed.

There are two ways to initialize ASCEDS:

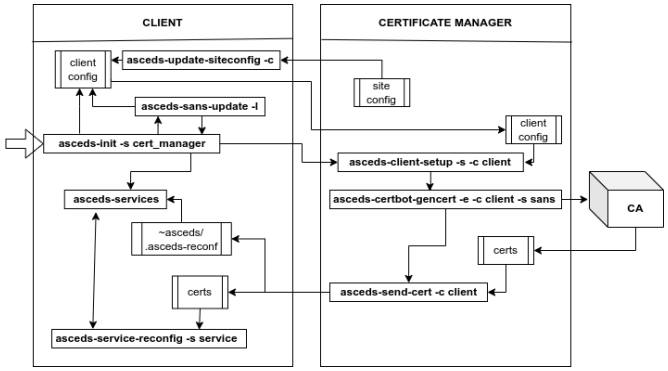

Fig. 3. Initial setup sequence of a client.

- if root access to the certificate manager is known, the *asceds-init* command as root performs the entire operation, as shown in Fig. 3.
- otherwise, the *asceds-init* command on the client should be followed by requesting creation and propagation of the certificates through the website as described in Section V, then followed by running *asceds-services* on the client after the certificates were delivered, to configure them into the services.

The *asceds-init* script performs the following operations:

- checks/fixes the network configuration of the certificate manager;
- builds the certificate manager website URL, and checks its availability;
- retrieves by wget and parses the site configuration file from the certificate manager, using the *asceds-update-siteconfig -c* script as described in Section III and Fig. 2;
- sets the certificate encoding in ∼asceds/asceds-hostconf.conf; to change the encoding, *asceds-init* should be ran again;
- generates the configuration file ∼asceds/cert.conf for the certificate, containing the name, the type of client, the SAN list, and the encoding, using the *asceds-sans-update* script;
- creates asceds-cert-refresh crontab which automatically detects and reconfigures the site configuration file and the services when updates are available;
- if root access to the certificate manager is available, the *asceds-client-setup* script is run remotely to generate certificate files and/or push them to the client;
- if a new certificate was obtained and propagated to the client, runs *asceds-services*.

The *asceds-sans-update* script is used to change the SAN list for the client. Normally, SANs are parsed out of /etc/ssl/openssl.cnf and loaded into the certificate configuration file which is made available to the certificate manager. However, alternate host names affect the network configuration of the client and other services like postfix. Currently modifying the SAN list in openssl.cnf also reconfigures /etc/hosts and /etc/postfix/main.cf. If *asceds-sans-update* is used in standalone mode, it provides instructions about how to complete the operation, as described in Subsection IV-D.

The *asceds-client-setup* script assists the initial setup of a new managed client on the certificate manager. It performs the following operations:

- retrieves the cert.conf file by scp from the client and stores it into ∼asceds/⟨client⟩_cert.conf;
- generates the certificate files into ∼asceds/⟨client⟩/, by using the *asceds-certbot-gencert* script;
- propagates the certificates back to the client by using the *asceds-send-cert* script described in Subsection IV-A.

The *asceds-certbot-gencert* script generates new certs using certbot which stores the files in /etc/letsencrypt/live/. First the hostname and the SANs are matched against the authorized domains. Then the certificate files are requested using certbot. If the certificates were generated successfully, for managed clients they are copied to ∼asceds/⟨client⟩/ which is accessible to ASCEDS. For unmanaged clients the certificate files are copied into the website space at an obfuscated location under /usr/share/asceds/cert/keys/. Since *asceds-certbot-gencert* is basically a ”certbot certonly” wrapper, for easy debugging and to prevent errors the certbot command is actually issued only if the ”-e” option is used or if CERTBOTEXEC is not empty, otherwise it is just listed to the standard output.

The *asceds-services* script can add/remove/refresh services which need certificates and can be reconfigured by ASCEDS. The library of available services is given by the *.sh.proto scripts in /usr/lib/asceds/bin/service.d/. Activating a service, copies the prototype file to /etc/asceds/service.d/ keeping just the .sh extension. Only these files are executed when a new certificate triggers service reconfiguration. The .sh files can be adjusted easily to match the specifics of the service on that particular client. They will not be overwritten by ASCEDS upgrades. Refreshing the service restores the default script, equivalent with first de-activate and then activate. The last version of the removed script is backed-up using a .bak extension. Deactivation requests are executed first. The script offers a simple CLI interactive interface if no request is specified in the command line. If a service is activated, the script *asceds-service-reconfig* is run to configure it with the certificate. Some of the scripts are self-configuring at the first use, to be able to find the current location of the existing services. Thus, the .sh file might be different from the .sh.proto file from the distribution.

The *asceds-service-reconfig* script has two modes of operations:

- a cron mode with the ”-t” option, which reconfigures all the active services but only if the .asceds-reconf trigger file exists; the ”-s” option is ignored;
- an individual service mode with the ”-s” option, which reconfigures only the specified services and doesn’t check on any trigger.

The script performs the following operations:

- updates the site configuration file using the trigger if the .site-reconf trigger file exists; removes the trigger file;

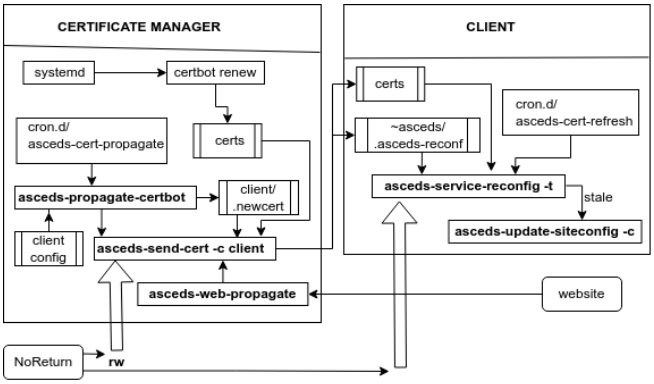

Fig. 4. Mechanism for automatic certificate update.

- checks if the available certificate is not stale; if the certificate is stale, refreshes the site configuration file if stale; if it is almost expired, send message to ALARM;
- copies certificate files to locations in /etc/ssl/, with the right permissions to preserve their security;
- reconfigures/restarts services through the .sh scripts from /etc/asceds/service.d/.
- in cron mode it removes the trigger file if execution was successful.

C. Push Automatically Renewed Certificates to Clients

The main time-saving feature of ASCEDS is to offer automatic handling of certificate renewals on the fully managed clients. The mechanism for pushing renewed certificates to managed clients is shown in Fig. 4.

When certificates are close to expiration, certbot-renew obtains renewals, process driven either by a cron job or by a systemd timer. Then, depending on the client type, the certificate files are propagated through a set of cron jobs and triggers.

On the certificate manager side, the asceds-cert-propagate cron job is checking for renewed certificates through the *asceds-propagate-certbot* script, as described in Subsection IV-A.

On the clients side, there are several possible cases:

- fully managed clients use the asceds-cert-refresh cron job, driving the *asceds-service-reconfig* script which is reconfiguring services automatically if the trigger ∼asceds/.asceds-reconf exists, and deletes the trigger if successful reconfiguration took place;
- privately managed clients receive email notification; read-/write access to their asceds account needs to be enabled, then the new certificate is propagated through the web interface;
- unmanaged clients receive email notification; the administrator uses the web interface to download the new certificate files, then reconfigures the services.

Using cron jobs and triggers ensures that operations which did not complete successfully because of temporary causes (e.g. network failure), will be attempted again later.

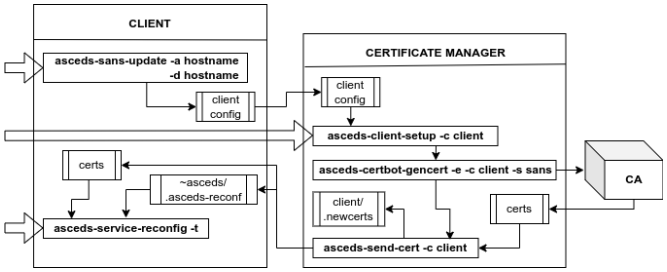

Fig. 5. Change-configuration sequence for a certificate.

D. Reconfiguration of a Managed Client

Another typical scenario is when a certificate, even if not close to expiration, needs to be reconfigured or re-created. The most common configuration change is adding or deleting components of the SAN list.

Reconfiguration of the managed clients is done using the *asceds-sans-update* script, described in Subsection IV-B, which generates a new certificate configuration file cert.conf. The action chain continues by running the *asceds-client-setup* script described in Subsection IV-B on the certificate manager. If root access is available, this can done directly as shown in the diagram from Fig. 5. Otherwise, the web interface can be used to run it through the *asceds-web-propagate* script which will be explained in Section V. At this stage, the reconfigured certificate files were generated and propagated to the client. To finalize the process, the script *asceds-service-reconfig* described in Subsection IV-A should be run on the client, or will be automatically run when it is scheduled by the asceds-cert-refresh cron job on the fully managed clients.

For the unmanaged clients, the process is completed manually through the web interface, which allows editing the SAN list and generating a new certificate which can be then downloaded as will be described in Section V.

E. Revoking Certificates

Revoking certificates is needed when servers are decommissioned. Currently, ASCEDS allows certificate revocation through the *asceds-certbot-revoke* script. The procedure is the same for managed and unmanaged clients. The script can be run directly if root access for the certificate manager is available, or using the web interface which thriggers the *asceds-web-unmanaged* script explained later in Section V.

The *asceds-certbot-revoke* script should be used as root on the certificate manager. It lists the status of the certificate (validity, SAN list), then prompts certbot for revocation. Since the operation cannot be reversed, the revoked certificate files are removed from ∼asceds/, and also from the web interface space for unmanaged clients. Since *asceds-certbot-revoke* is basically a ”certbot revoke” wrapper, for easy debugging and to prevent errors, the certbot command is actually issued only if the ”-e” option is used or if CERTBOTEXEC is not empty, otherwise it is just listed to the standard output.

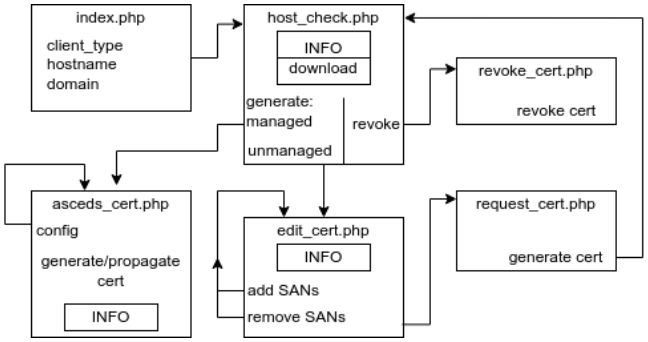

Fig. 6. Web interface structure for self-service.

V. ASCEDS Web Interface

The web interface is intended to:

- limit superuser access to the certificate manager, which may be a security concern in enterprise networks;
- allow multiple users to control their own set of domains, as described in Section II;
- distribute the site configuration file, as described in Section II.

Functionally, the web interface can deal with three types of operations:

- revoking certificates for managed or unmanaged clients;
- configuring and creating certificates for unmanaged clients;
- propagating requests from managed clients, generating the certificates, and submitting them back to the client.

Each of these operations is associated with a request queue which is executed as described in Section II. To avoid race conditions between different requests and to preserve atomicity of operations, only one request of any type may exist for a specific host name in the queues. All the operations performed through the queues, and the users requesting them, are logged into /var/log/asceds/request.history.

The web interface has the structure shown in fig. 6. The index.php page has the purpose of collecting information about the host name of interest, as well as documenting the layout of the entire process. This page uses GET to send the variables client_type, hostname, and domain to host_check.php. Here, the information is validated against the user authority and against the list of allowed domains. If everything checks out, the page searches for existing information about the provided host name. Data about an existing certificate is listed and download links for certificate files are shown for unmanaged clients. The page shows a "Create/Renew" button which acts differently based on the type of client, while taking care that the type is not changed inadvertently. Also, if a certificate exists, a "Revoke" button is exhibited.

The "Revoke" button uses POST to submit the host information (name, domain, and SANs) to revoke_cert.php. This page validates the data, checks for race conditions, then places a request in the revoke queue. If the queue is ssh-driven, it is executed immediately by the *asceds-web-unmanaged* script with the "-r" option. Otherwise, the asceds-web-actions cron job will run the script at the time it is scheduled.

The "Create/Renew" button on host_check.php submits the unmanaged client host information to edit_cert.php. This page allows recursively submitting the data to itself while editing the SAN list. In the end, the host information is submitted to request_cert.php through the "Generate Certificate" button.

The code in request_cert.php validates the data, checks for race conditions, then places a request in the new certificate queue. If the queue is ssh-driven, it is executed immediately by the *asceds-web-unmanaged* script with the "-n" option. Otherwise, the asceds-web-actions cron job will run the script at the time it is scheduled. The certificate files can be downloaded through a link pointing back to the host_check.php page.

For managed clients, the "Create/Renew" button on host_check.php submits the managed client host information to asceds_cert.php. This page performs the following checks:

- validates the POST data;
- checks if the certificate manager has ssh read/write access to the client;
- checks if the new certificate configuration file is available on the client.

To fulfill all these conditions, the page recursively evolves through the tests and offers instructions to the user for fixing any problems which are encountered. If all the tests are passed, the page checks for race conditions and places a request in the propagate queue. If the queue is ssh-driven, it is executed immediately by the *asceds-web-propagate* script. Otherwise, the asceds-web-actions cron job will run the script at the time it is scheduled.

The *asceds-web-unmanaged* script executes web interface revoke and new certificate requests. The revoke request triggers the execution of the *asceds-certbot-revoke* script described in Subsection IV-E. The new certificate request builds a certificate configuration file in ∼asceds following the format used by ASCEDS managed clients. Then it executes the *asceds-certbot-gencert* script described in Subsection IV-B.

The *asceds-web-propagate* script executes web interface propagate requests. If new certificate files don't exist (signaled by a missing .newcerts file) they are created using the *asceds-client-setup* script described in Subsection IV-B, and the .newcerts flag file is set in ∼asceds/⟨managed_client⟩/. Otherwise, the existing certificate files are sent to the client by the *asceds-send-cert* script. This way, if an automatically generated renewal exists, it will be propagated before generating a new certificate, which should happen in a different request after the renewed certificate is on the client. As a side effect, *asceds-web-propagate* sends the updated site configuration file to the client if the trigger file .newsiteconf exists, as shown in Fig. 2.

Upon successful completion, the request files are removed from the queue. Also, if the request script is run by cron (without the "-c" option), an email notification is sent to the web user.

The website can be customized by adding banners named org-custom.png and dept-custom.png in the public/figs/ directory of the website. These banners will be displayed in the header of the web pages. Php code placed in ∼asceds/custom.php is displayed under the title of each page.

VI. Testing with Self-Signed Certificates

Asceds offers a fake certbot testing environment which can be used with self signed certificates, without any ACME server or credentials. This environment can be optionally triggered during the *asceds-certmanager-setup* execution, if no existing or previous installation of real certbot is detected.

The fake certbot environment is provided in the in /usr/lib/asceds/ss directory. It contains a *certbot* shell script which mimics the real certbot behavior, and a *certbot-selfsigned-refresh* script which is ran by the asceds-selfsigned-refresh crontab to renew certificates older than 3 days before expiration.

During the installation, the fake certbot environment creates an /etc/letsencrypt directory tree similar to the one used by a real certbot, and places a .fake signature file inside. This signature is verified during future installs, to make sure a real certbot installation is not overwritten.

The configuration file used by the fake certbot is /etc/asceds/certbot-ss.conf. The file contains the default certificate expiration term CERTTERM (default 10 years), as well as information used for self-signed certificate content.

The fake certbot environment is useful for testing and development of ASCEDS, and as emulator for demo purpose.

VII. Conclusions

This paper presents the ASCEDS software which performs automated SSL/TLS certificate handling for a large computer network using one certificate manager and a CA provider offering an ACME interface. The software was developed and is currently used in a production environment at Carnegie Mellon University. The certificate management system can interact through a multi-user web interface or through a CLI, offering a simple API to the managed clients consisting of a read/write asceds account accessible through ssh from the certificate manager, and a certificate configuration file. ASCEDS decreased significantly the system administrator's time to generate/reconfigure/revoke certificates, by replacing most of these operations by several mouse clicks, or making them completely automatic wherever possible.

Currently, ASCEDS clients work on several flavors of Linux (Debian, Ubuntu, ArchLinux, Raspbian, CentOS 7), and the list of supported services includes apache, dovecot, grafana, kvmd-nginx, nginx, prayer, and jupyterhub. The certificate manager was tested on Ubuntu 18.04 and newer. Future work will be oriented towards developing clients for other operating systems, as well as adding new modules for various services which may benefit from SSL/TLS certificates.

References

[1] Florin Manolache, Octavian Rusu, *Automated SSL/TLS Certificate Distribution System*, 20th RoEduNet Conference: Networking in Education and Research, 2021.

[2] Jonathan Katz, Yehuda Lindell, *Introduction to Modern Cryptography*, CRC Press, 2007.

[3] David R. Mirza Ahmad, et al, *Hack Proofing Your Network*, 2nd Edition, Syngress, 2000.

[4] Let's Encrypt, *Chain of Trust*, https://letsencrypt.org/certificates/

[5] Rolf Oppliger, *SSL and TLS: Theory and Practice*, Artech House, 2009.

[6] Patrick Nohe, *Maximum SSL/TLS Certificate Validity is Now One Year*, https://www.globalsign.com/en/blog/maximum-ssltls-certificate-validity-now-one-yea

[7] Let's Encrypt, *Let's Encrypt Documentation*, https://letsencrypt.org/docs/

[8] Internet Engineering Task Force, *Automatic Certificate Management Environment (ACME)*, https://tools.ietf.org/html/rfc8555

[9] Electronic Frontier Foundation (EFF), *Certbot*, https://certbot.eff.org/

[10] InCommon, *InCommon Certficate Service*, https://incommon.org/certificates/repository/