SAGE2 Security Document

Laboratory for Advanced Visualization & Applications, University of Hawai'i at Mānoa
Electronic Visualization Laboratory, University of Illinois at Chicago
http://sage2.sagecommons.org/

September 2017

Table of Contents

SAGE2 receives major funding from NSF #ACI-1441963

Introduction

This document describes security features in the SAGE2 environment, addressing frontend and backend aspects. At its core, SAGE2 is a web server enabling collaborative work on large displays. Like any web server, it is susceptible to attacks and, like any web user, a SAGE2 user is vulnerable to security issues. However, SAGE2 controls both the frontend (user experience) and the backend (the web server itself), which somewhat limits the types of attacks.

As for any web server within your infrastructure, you must apply the same best practices – network access, user login, firewall, etc. – which are outside the scope of the SAGE2 software.

SAGE2 developers apply best practices to ensure a secure environment:

- SAGE2 is built upon Node.js (https://nodejs.org/), the industry-leading technology for web services (Node.js's package ecosystem is the largest ecosystem of open source libraries in the world). It is used by many companies (Netflix, IBM, Walmart, PayPal, …).
- SAGE2 is always built with the latest and most secure version of Node.js.
- SAGE2 requires administrators to install secure and up-to-date SSL certificates to ensure secure and encrypted communication between the server and the users. All communication between the server and users is encrypted (using HTTPS and Secure Websockets).
- SAGE2 relies on Google Chrome and Electron as display and user interface (UI) frameworks (https://electron.atom.io/). Electron is a framework that creates native applications with web technologies, and is used by companies such as Microsoft, Facebook, Slack, and Docker to deliver desktop applications
- User access to the SAGE2 server can be controlled by a password (simple MD5 hashing). This is just intended as a meeting access token, and does not replace secure authentication to the network.
- Various security features can be enabled to enforce better security (cross-site scripting, clickjacking protection, content type options, content security policy, etc.), which can limit functionality in some cases. Best practices from Apache or NGINX can be applied (see description below).
- External websites can be shown within SAGE2 and are run within ‘sandboxed’ browser instances (i.e., webviews).
- Several companies ran security audits (e.g., using HP Fortify) after installing SAGE2 and reported several CVEs (Common Vulnerabilities and Exposures, https://cve.mitre.org/), which were promptly fixed. Various university campuses with SAGE2 installations (UIC, UCSD) run continuous network audits as well.
- For maximum isolation, a SAGE2 server can be run within an application container (SAGE2 supports Docker). Firewall and port mapping must be configured to enable communication among the server, the displays and the users.

Server Configuration

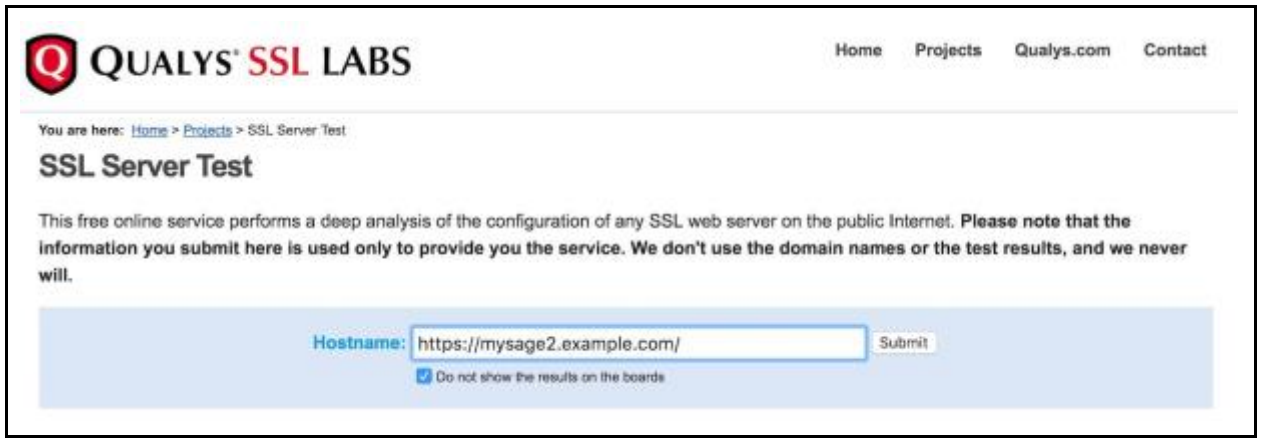
As the SAGE2 server is an HTTP/HTTPS server, it is susceptible to common web attacks. The first step to ensure security is to provide valid SSL certificates as described in this section.

The HTTP and HTTPS ports are the only ports that need to be opened for SAGE2 use: port 80 and 443 for production scenarios. The default ports for user development are 9292 and 9090.

To verify your setup, use “ssllabs.com” to ensure that your certificates are valid, up to date, and using proper cyphers. Put your HTTPS address at the following URL:

https://www.ssllabs.com/ssltest/

Within a few minutes, you should get a report of the validity of your setup. The SAGE2 server should get an A or A+ rating.



The HTTP header responses from the SAGE2 server can be configured with the following attributes:

Content type of the HTTP response, default value

- "Content-Type" ⇒ "text/html; charset=utf-8";

X-Frame-Options: the header can indicate whether a browser is allowed to render a page within an <iframe> element or not. This is helpful to prevent clickjacking attacks by ensuring your content is not embedded within other sites. For example, values can be "SAMEORIGIN" or "DENY". See more here <https://developer.mozilla.org/en-US/docs/HTTP/X-Frame-Options>.

- “X-Frame-Options" ⇒ "SAMEORIGIN";

X-XSS-Protection: This header enables the Cross-Site Scripting (XSS) filter built into most recent web browsers. It's usually enabled by default, so the role of this header is to re-enable the filter for this particular website if it was disabled by the user.

- "X-XSS-Protection" ⇒ "1; mode=block";

X-Content-Type-Options: The only defined value, "nosniff", prevents Internet Explorer and Google Chrome from MIME-sniffing a response away from the declared content-type. This also applies to Google Chrome when downloading extensions. This reduces exposure to drive-by download attacks and sites serving user uploaded content that, by clever naming, could be treated by MSIE as executable or dynamic HTML files.

- "X-Content-Type-Options" ⇒ "nosniff";

More security features are also available, by setting extra parameters in your SAGE2 configuration file:

```
        security: {
                // The SSL method to use
                secureProtocol: "TLSv1_2_method",
                // Enable HSTS: HTTP Strict Transport Security - once HTTPS, always HTTPS
                enableHSTS: true,
                // Enable CSP: Content-Security-Policy
                enableCSP: true
                // Enable HPKP: HTTP Public Key Pinning Extension
                enableHPKP: true,
        },
```

secureProtocol: String, The SSL method, e.g. SSLv3_method, is used to force a SSL version. The possible values depend on your installation of OpenSSL and version of NodeJS.
Configuration: security.secureProtocol: string (undefined by default).

- Valid values: TLSv1_2_method, SSLv23_method, ... Default: SSLv23_method (as in node v8.3)

HTTP Strict Transport Security: HSTS is an opt-in security enhancement. Once a supported browser receives this header, the browser will prevent any communications from being sent over HTTP to the specified domain and will instead send all communications over HTTPS.
Configuration: security.enableHSTS: true/false (false by default)

- "Strict-Transport-Security" ⇒ "max-age=31536000";

Content-Security-Policy: Instead of blindly trusting everything that a server delivers, Content-Security-Policy defines the HTTP header that allows you to create a whitelist of sources of trusted content, and instructs the browser to only execute or render resources from those sources. Even if an attacker can find a hole through which to inject script, the script won't match the whitelist, and therefore won't be executed. *default-src 'none'* ➔ default policy that blocks absolutely everything.
Configuration: security.enableCSP: true/false (false by default)

- "Content-Security-Policy" ⇒

  "default-src 'none';

  plugin-types image/svg+xml;

object-src ‘self’;

child-src ‘self’ blob:;

connect-src *;

font-src ‘self’ fonts.gstatic.com;

form-action ‘self’;

img-src * data:;

media-src ‘self’;

style-src ‘self’ ‘unsafe-inline’ fonts.googleapis.com;

script-src * ‘unsafe-eval’;”

HTTP PUBLIC KEY PINNING: HPKP Key pinning is a trust-on-first-use (TOFU) mechanism. The first time a browser connects to a host it lacks the information necessary to perform "pin validation", so it is not able to detect and thwart a MITM attack (man in the middle). This feature only allows detection of these kinds of attacks after the first connection.
Configuration: security.enableHPKP: true/false (false by default)

- "Public-Key-Pins" ⇒ ‘pin-sha256=”Pin1”; pin-sha256="Pin2”;max-age=2592000; includeSubDomains’
- The values of the ‘pins’ are generated with the scripts provided in the ‘keys’ folder of SAGE2: GenHPKP.bat for Windows, and GenHPKP.sh for Unix (MacOSX and Linux).
    - The scripts take two parameters: your main CA certificate and the CA certificate for 127.0.0.1 (the second pin is used as a backup).
    - Example: *./GenHPKP.sh myserver-ca.crt 127.0.0.1-ca.crt*
    - This generates two files (pin1.sha256, pin2.sha256) containing hashes/pins for your site. The pins will be added to the HTTP responses, the browser will then compare the values to the ones previously received. If different, the HTTP request will be denied
- If you change or update your site certificates, the pins should be regenerated.

Web Certificates

SAGE2 can generate so-called “self-signed certificates” to support running the secure HTTPS protocol (instead of the insecure common HTTP protocol). This requires SSL certificates to prove the identity of the server to its clients.

Self-signed Certificates

To generate self-signed certificates:

- **Windows**
  - Edit '<SAGE2_directory>\keys\GO-windows.bat'
    - Add lines with a list of hostnames for your server
    - Save file
  - Double click 'GO-windows.bat'
- **MacOSX**
  - Open the file 'GO-mac' inside the '<SAGE2_directory>/keys/' folder
    - Add additional host names for your server in the variable servers (optional)
    - Save file
  - Open Terminal
    - cd <SAGE2_directory>/keys
    - ./GO-mac
      - enter your password when asked (the keys are added into the system)
- **Linux**
  - Open the file 'GO-linux' inside the '<SAGE2_directory>/keys/' folder
    - Add additional host names for your server in the variable servers (optional)
      - for instance, add the short name and the fully qualified domain name of your server
    - Save file
  - In a Terminal
    - cd <SAGE2_directory>/keys
    - ./GO-linux

However, self-signed certificates are only valid for testing and development since they are not recognized by any trust authorities. Modern web browsers reject self-signed certificates and ask the user to approve the security exception. Some mobile browsers ignore completely such certificates.

Signed Certificates

Anyone installing SAGE2 should acquire valid SSL certificates from a reputable source. Most campuses in the USA can get certificates for free (or a modest sum) through their local campus IT department. For instance, see **https://www.incommon.org/**

> *InCommon, operated by Internet2, provides a secure and privacy-preserving trust fabric for research and higher education, and their partners, in the United States. InCommon's identity management federation serves 9 million end-users. InCommon also operates a related assurance program, and offers certificate and multifactor authentication services.*

Alternatively, web hosting companies usually sell certificates to their clients. A number of security companies also sell host certificates, requiring you to prove that you "own" the server in question.

A **requirement** for any certificate is that your host has a fully qualified domain name (FQDN): this is the complete domain name for a specific computer, or host, on the Internet. The FQDN consists of two parts: the hostname and the domain name. For example, an FQDN for a hypothetical SAGE2 server might be sage.somecollege.edu.

Host certificates are only valid for a fixed period of time (1 or 3 years, usually) and for a unique machine. Wildcard certificates supports any machine within a network subdomain (for instance, **.my_dept.my_univ.edu*) and function the same way as host certificates. Not all certificate authorities generate wildcard certificates and they are generally more expensive.

The process goes as follows:

1. you generate a request and a private key for any given server,
2. you send the request to a certificate authority,
3. you receive a signed certificate
    a. you might also receive a CA (certificate authority) certificate providing information on the chain of authority entities involved in your certificate.
4. your signed certificate and your private key are setup in SAGE2 to provide a valid secure SSL connection with the clients
    a. The CA certificate chain can be provided to speed up the validation process.

There are four files involved:

- A CSR or **Certificate Signing request** is a block of encrypted text that is generated on the server for which the certificate will be used. It contains information that will be included in your certificate, such as your organization name, common name (domain name), locality, and country. It also contains the public key that will be included in your certificate. The file uses the ".csr" extension and it looks like this:
    - `-----BEGIN CERTIFICATE REQUEST-----`
    - `MIIC4jC...`
    - `…`
    - `-----END CERTIFICATE REQUEST-----`

- A **private key** is created at the same time that you create the CSR. It usually named with the extension ".key"
    - `-----BEGIN RSA PRIVATE KEY-----`
    - `MIIEp…`
    - `...`
    - `....hw==`
    - `-----END RSA PRIVATE KEY-----`
- The **server certificate** received from your certificate authority, named with the extension ".cer", is a "X509 Certificate only, Base64 encoded" file:
    - `-----BEGIN CERTIFICATE-----`
    - `MIIF….`
    - `….`
    - `-----END CERTIFICATE-----`
- You might get a **CA certificate,** or multiple certificates (in case of a hierarchy of certificate authorities), which is a single "X509, Base64 encoded" file.
    - `-----BEGIN CERTIFICATE-----`
    - `MIIF….`
    - `….`
    - `-----END CERTIFICATE-----`
    - `-----BEGIN CERTIFICATE-----`
    - `MIIF….`
    - `….`
    - `-----END CERTIFICATE-----`
    - `-----BEGIN CERTIFICATE-----`
    - `MIIF….`
    - `….`
    - `-----END CERTIFICATE-----`

SAGE2 requires the private key and the signed certificate. Optionally, it will load the CA certificates, if available. Certificates for SAGE2 should be named very precisely in order to be loaded automatically by the server, and stored in the 'keys' folder:

- the server **private key** should be named: [*host*]-server.key
    - where [*host*] is the value associated with the key 'host' in your configuration file
    - example: if my server is called *'traoumad.evl.uic.edu'* then my key file should be:
        - *traoumad.evl.uic.edu-server.key* in the 'keys' folder
- the server **host certificate** should be named: [*host*]-server.crt
    - where [*host*] is the value associated with the key 'host' in your configuration file
    - example: if my server is called *'traoumad.evl.uic.edu'* then my key file should be:
        - *traoumad.evl.uic.edu-server.crt* in the 'keys' folder
- optionally, the **CA certificates** should be named: [*host*]-ca.crt
    - where [*host*] is the value associated with the key 'host' in your configuration file
    - example: if my server is called *'traoumad.evl.uic.edu'* then my key file should be:
        - *traoumad.evl.uic.edu-ca.crt* in the 'keys' folder
- for domain certificates:

- the domain private key should be named with a '_' and then the domain name and the extension ".key"
  - example: for the "evl.uic.edu" domain, the file is named "_.evl.uic.edu.key"
- the domain certificate should be named with a '_' and then the domain name and the extension ".crt"
  - example: for the "evl.uic.edu" domain, the file is named "_.evl.uic.edu.crt"
- the CA certificate should be named with a '_' and then the domain name and the extension "-ca.crt"
  - ex: for the "evl.uic.edu" domain, the file is named "_.evl.uic.edu-ca.crt"

You also need the same series of files for each and every hostname that your SAGE2 server uses and is accessed through. This is important for multi-home computers (often the case in clusters, or machines having connections to multiple networks). Those certificates are usually used internally and can be self-signed certificates. Common names are added in your "alternate_hosts" section of your SAGE2 configuration, such as "127.0.0.1" and "localhost".

For example, the following configuration will need a valid signed certificate for the name "iridium.evl.uic.edu" and self-signed certificates for the following names: "localhost", "127.0.0.1", "131.193.183.199", "iridium.evl.optiputer.net". These are all valid names to access this unique server from multiple network locations:

```
        name: "Cybercommons",
        host: "iridium.evl.uic.edu",
        port: 443,
        index_port: 80,
...
        alternate_hosts: [
                "localhost",
                "127.0.0.1",
                "131.193.183.199",
                "iridium.evl.optiputer.net"
        ],
...
```

The extra self-signed certificates can be generated by the commands described in the beginning of the section (scripts in the 'keys' folder).

Let's Encrypt

A third way is emerging to generate valid certificates and increase the use of valid certificates on the Internet.

Here is an excerpt from "https://letsencrypt.org/about/":

> *Let's Encrypt* is a free, automated, and open certificate authority (CA), run for the public's benefit. Let's Encrypt is a service provided by the Internet Security Research Group (ISRG).

The key principles behind *Let's Encrypt* are:

- Free: Anyone who owns a domain name can use *Let's Encrypt* to obtain a trusted certificate at zero cost.
- Automatic: Software running on a web server can interact with *Let's Encrypt* to painlessly obtain a certificate, securely configure it for use, and automatically take care of renewal.
- Secure: *Let's Encrypt* serves as a platform for advancing TLS security best practices, both on the CA side and by helping site operators properly secure their servers.
- Transparent: All certificates issued or revoked will be publicly recorded and available for anyone to inspect.
- Open: The automatic issuance and renewal protocol will be published as an open standard that others can adopt.
- Cooperative: Much like the underlying Internet protocols themselves, *Let's Encrypt* is a joint effort to benefit the community, beyond the control of any one organization

The process is still fairly manual and mostly described for the Unix operating system but it works and can generate valid signed certificates for use in SAGE2.

The process is fairly automated for existing mainstream HTTP servers (Apache, Nginx, …), but requires some automation for SAGE2. This is work-in-progress and we hope to support it in the future. The process involves having the HTTP server running and providing a generated file to the 'letsencypt.org' server to prove the server identity. The CSR request is then generated and signed automatically to provide a valid certificate. This would automatically provide a free certificate for anyone with a server and a FQDN hostname.

Audits

UCSD Campus network

From ___@eng.ucsd.edu:

> *1 - "Campus did a **Qualsys scan** of my SAGE2 install, and it looks like it is susceptible to directory traversal. Also, I have a public interface that is configured, but it is not in the SAGE2 configuration, yet it is still responding via IP address."*
>
> *2 - The IP address is not listed as an alternate host, and is there solely to allow updating of SAGE2 via npm since I could not get npm to honor the proxy config. Not a big deal, as long as the public facing interface is only there for software updates/patching.*
>
> *I'll see if I can get a specific CVE, but here is the detailed output from the scan minus the /etc/passwd file contents. Note, this interface should not be responding on 9090 at all, and I use password protection on the site as well:*
>
> *GET /..//..//..//..//..//..//..//..//etc/passwd HTTP/1.1*
> *Host: [REDACTED]*
> *Connection: Keep-Alive*
>
> *HTTP/1.1 200 OK*
> *Content-Type:*
> *X-Frame-Options: SAMEORIGIN*
> *X-XSS-Protection: 1; mode=block*
> *X-Content-Type-Options: nosniff*
> *Access-Control-Allow-Headers: Range*
> *Access-Control-Expose-Headers: Accept-Ranges, Content-Encoding, Content-Length, Content-Range*
> *Cache-Control: no-cache, no-store, must-revalidate*
> *Pragma: no-cache*
> *Expires: 0*
> *Content-Length: 2147*
> *Date: Fri, 02 Jun 2017 05:31:45 GMT*
> *Connection: keep-alive*

Air France – KLM / Group Technology Office

From: ___@klm.com

*Find attached a security scan as performed by **HP Fortify**. As you may want to have a look first, I did not include your developers. You are welcome to forward them to your team member as you see fit.*

Noted as most important findings:

- *Webserver within the Docker container: directory traversal should be configured off, e.g. to prevent access to the /etc/password file.*
- *JavaScript should prevent Cross Side Scripting (XSS) – the report shows code samples / recommendations.*
- *Weakness introduced by Self Signed Certificates: there are SSL code suites that are not safe anymore.*

There are more details and recommendations in the report. We are happy to discuss them with you and your development team. As the report already contains many details, please let us know if / when you like me to arrange such a call.

[….] However we can only proceed if the security vulnerabilities have been addressed, a standard procedure in admitting new applications.

NASA's Goddard Space Flight Center

___@nasa.gov
NASA Center for Climate Simulation
Goddard Space Flight Center

> *“Our security team has done a preliminary review of SAGE2 and I thought you’d be interested in the findings (see attached). If a conversation would be useful, let me know. We remain very interested in trying to get our proxy approach to work.”*