Steps to Release a New Official Version of ESAPI 2.x.y.z
by Kevin W. Wall – 2021

Table of Contents

1 Assumptions

These instructions are all Linux / UNIX based and assume use of a *nix shell such as bash or ksh. It is also assumed that you have a GitHub account and some minimal familiarity with git, GitHub, and Maven. You may be able to get it to work using Cygwin, UWin, or 'git bash' from Windows, but that has not be tested. It is also assumed that you know how to carefully following instructions. :)

You must obviously have appropriate GitHub project access to the ESAPI repository (https://github.com/ESAPI/esapi-java-legacy.git) and have access to Maven Central. (Otherwise, why are you trying to do an ESAPI release?)

Finally, it is assumed that the previously created Sonatype OSS repository for ESAPI will be used, so a new repository will not need to be created.

2 One Time Initialization

2.1 Create a code signing key

Use GPG to create code signing key for signing ESAPI releases. Use RSA (option 4) for a signing key. The minimum key size for an RSA key for use with Sonatype is a 2048 bit modulus. This is the default for most GPG implementations, but you will be prompted for it. Not while many self-proclaimed pundits insist that the minimal key size should be 4096 bits,but according to NIST 2048 bits is sufficient for signing keys (see Table 2-1, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf). Note that longer keys will take longer to generate. For the comment use something like 'ESAPI signing key' (or perhaps GitHub signing key', if you intend to use the same key for multiple GitHub projects). Be sure to set a passphrase for your private key and save that passphrase somewhere secure, such as in a Password Manager.
```
$ gpg --gen-key
```

Example:
```
$ gpg --gen-key
gpg (GnuPG) 1.4.16; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory `/home/kww/.gnupg' created
gpg: new configuration file `/home/kww/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/kww/.gnupg/gpg.conf' are not yet active
during this run
```

```
gpg: keyring `/home/kww/.gnupg/secring.gpg' created
gpg: keyring `/home/kww/.gnupg/pubring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
        0 = key does not expire
     <n>  = key expires in n days
     <n>w = key expires in n weeks
     <n>m = key expires in n months
     <n>y = key expires in n years
Key is valid for? (0) 2y
Key expires at Mon 26 Apr 2021 07:38:44 PM EDT
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Kevin W. Wall
Email address: kevin.w.wall@gmail.com
Comment: Signing key for GitHub
You selected this USER-ID:
    "Kevin W. Wall (Signing key for GitHub) <kevin.w.wall@gmail.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.
```

As mentioned above, a passphrase is not strictly required, but is highly encouraged.

The gpg command will ask you to move your mouse or enter random keys on your keyboard to generate entropy to initialize the random number generator. On slow single-user systems, this can take a long time, so be patient. However, if you see something like this:

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

Not enough random bytes available.  Please do some other work to give
the OS a chance to collect more entropy! (Need 82 more bytes)
```

```
+++++

Not enough random bytes available.  Please do some other work to give
the OS a chance to collect more entropy! (Need 120 more bytes)
```

where, despite your attempts to generate entropy, gpg tells you that you need an *increasing* amount (e.g., here it goes up from 82 to 120 bytes), then it means that something on your system is consuming available system entropy faster than you are able to generate it, you would be advised to abort and start over after figuring out and (temporarily, at least) stopping whatever is consuming the entropy. Otherwise, you are going to be in for a *long* wait.

Confirm you have created the new key; look for one with the identity you just created.
```
$ gpg --list-public-keys
```

For example, here is output of mine, with my key ID highlighted:
```
$ gpg --list-public-keys --keyid-format short
/home/kww/.gnupg/pubring.gpg
---------------------------
pub   2048R/8A2A524F 2016-02-03
uid                  Kevin W. Wall (GitHub signing key)
<kevin.w.wall@gmail.com>
```

Some versions of GPG / PGP will create subkeys, which will look something like this:
```
pub   rsa4096/680D16DE 2018-09-11 [SC]
      5069A233D55A0EEB174A5FC3821ACD02680D16DE
uid           [ unknown] VeraCrypt Team (2018 - Supersedes Key
ID=0x54DDD393) <veracrypt@idrix.fr>
sub   rsa4096/26878A32 2018-09-11 [E]
sub   rsa4096/5483D029 2018-09-11 [A]
```

Note that if *your* generated listed key shows sub-keys (listed as 'sub'; see above for example), as per the Sonatype documentation, you are advised to delete the sub-keys as documented at:
https://central.sonatype.org/pages/working-with-pgp-signatures.html#delete-a-sub-key

Configure git to use your signing key. I recommend using the key id (shown in the turquoise highlighting, above) for the alias.
```
$ git config --global user.signingkey "key-short-alias-here"
```

2.2 Configure Maven's ~/.m2/settings.xml

2.2.1 Configure for ESAPI's user name for the Sonatype repo:

You must add a section to your ~/.m2/settings.xml file that as a <server> section with an <id> of 'ossrh'. It should look like this:

```
<server>
  <id>ossrh</id>
  <username>esapi-team</username>
  <password>
    [Contact Kevin Wall via an encrypted email or
    Signal to get this and for instructions of how to
    secure it.]
  </password>
</server>
```

This <server> element obviously is a child element under the <servers> element.

2.2.2 Create a <profile> element for your signing key

In the same ~/.m2/settings.xml file, add a profile like this:

```
<profile>
  <id>ossrh</id>
  <activation>
    <activeByDefault>true</activeByDefault>
  </activation>
  <properties>
    <gpg.keyname>YOUR NAME</gpg.keyname>
    <gpg.passphrase>YOUR_PRIVATE_KEY_PASSPHRASE</gpg.passphrase>
  </properties>
</profile>
```

For 'YOUR NAME', you will use the name associated with your signing key. (If you have used that name for multiple keys under ~/.gnupg, contact Kevin for additional instructions. It gets complicated, because Maven otherwise will select whatever is the 'first' key (which may be not the one you want) that matches 'YOUR NAME'.

If you leave out the <gpg.passphrase> element Maven is supposed to prompt you for your passphrase to decrypt your private key. (However, this has not been verified.) Alternately, you can create a master password and place it's hashed value in a ~/.m2/settings-security.xml file and then use it to encrypt your other passwords or other sensitive information in your ~/.m2/settings.xml file. For details on how to do this, see

https://maven.apache.org/guides/mini/guide-encryption.html
This will not be covered in further detail here.

2.3 Publish your public key so it can be used with Sonatype

Sonatype requires signed releases. So that it (and others) can verify the signature, it needs to be able to access your public key. That means that you need to may your key available to one of the key servers that it uses. To do that find your "key id" and then use that key id to upload your public key. (Note that. If applicable, you should do this after you have deleted any sub-keys.)

The URLs for the key servers are:
> hkp://pool.sks-keyservers.net
> hkp://keyserver.ubuntu.com

For example:

```
$ gpg –list-public-keys --keyid-format short
/home/kww/.gnupg/pubring.gpg
---------------------------
pub    2048R/8A2A524F 2016-02-03
uid                    Kevin W. Wall (GitHub signing key)
<kevin.w.wall@gmail.com>
$ gpg --keyserver hkp://pool.sks-keyserver.net –send-keys 8A2A524F
```

You only need to send your key to *one* of the key servers. It will be replicated to the others. (Actually, there are quite a few others, including the original MIT key server, but Dave Wichers was only able to get the above two to work.)

2.4 Create an NVD API Key that can be used with Dependency Check

At various steps throughout performing the ESAPI release, you will be instructed to run Maven in a manner that runs OWASP Dependency Check via the org.owasp:depencency-check-maven Maven plugin. (For example, running 'mvn site' will execute this.)

As of ESAPI 2.5.3.0, we have upgraded to use Dependency Check 9.0.0 (or later), because as per the Dependency Check GitHub README.md file, it states:
> **Upgrading to 9.0.0 or later is mandatory**; previous versions of dependency-check utilize the NVD data feeds which will be deprecated on Dec 15th, 2023. Versions earlier then 9.0.0 are no longer supported and could fail to work after Dec 15th, 2023.

That same README also states the strong recommendation to obtain and use an NVD API key. While it describes how to obtain the NVD API key, it doesn't describe how to use it with the Dependency Check Maven plugin. (It is documented, but assumes a lot of knowledge of the operations of Maven plugins and that you will know where to look.) So, this section describes

the one time setup for that.

NVD API keys can be obtained via https://nvd.nist.gov/developers/request-an-api-key. The API keys are considered to be per person, not per project or per organization. Fill out all parts of the form, agree to the Terms of Use by checking the box at the bottom of those terms and then submit the form. Within 5 minutes or so, you should be emailed a link that you can follow to collect your API key along with some brief instructions. Once you collect the key, I'd recommend it putting it somewhere safe, such as a Password Manager, and also place it in an exported environment variable named "NVD_API_KEY".

As of the ESAPI 2.5.3.0 release, the ESAPI pom.xml file has been modified to assume that your NVD API key is available in an exported environment variable named NVD_API_KEY.

3 Steps for each new ESAPI 2.x.y.z release

In the following 'x.y.z' should be released by the actual release details. E.g., 2.x.y.z might become '2.1.0.2' or '2.2.0.0' or whatever. (Or even something like '2.2.1.0-RC1' if you wish to make "release candidates" available for beta testing.) This should be the same as the <project><version> version # (i.e., the XPath "/project/version") in pom.xml (minus any "-SNAPSHOT" part). E.g.,

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.owasp.esapi</groupId>
    <artifactId>esapi</artifactId>
    <version>2.2.0.0-RC2</version>
    <packaging>jar</packaging>
```

Announce a "code freeze" on the 'develop' branch and that you will be not merging any further PRs until the 2.x.y.z release is made official. However, make sure that you have incorporated all the PRs that you intend to place into the 2.x.y.z release by merging them to 'develop'.

Create a release branch of the frozen 'develop' branch. We are essentially following the instructions in:
> https://nvie.com/posts/a-successful-git-branching-model/#release-branches

Start clean, based on the latest from the 'develop' branch (which is ESAPI's default). If you have an SSH key, use:

```
$ git clone git@github.com:ESAPI/esapi-java-legacy.git
```

Otherwise, use:
```
$ git clone https://github.com/ESAPI/esapi-java-legacy.git
```

We will name the (possibly temporary) release branch '2.x.y.z' where 'x.y.z' is defined as above. (Because 'develop' is our default branch for ESAPI in GitHub and our goal is to always have our 'main' (formerly 'master') branch always reflect the latest official build [i.e., the previous release].)
```
$ git checkout -b 2.x.y.z develop
```

Note that any final **emergency** bug fixes that come up may be applied to this branch, but (ideally) the 'develop' branch should remain frozen until after the release. (Also, making changes after you have created a signed tag for the release [see below] will mean that you will have to recreate that signed tag.) [**WARNING**: Do <u>NOT</u> create the branch name that you are working on and will ultimately be merging, the same as the the name for the git tagname. If you do, that will cause problems later on. For that reason, it is suggested that you name the branch of the form '2.x.y.z' and the tagname of the form 'esapi-2.x.y.z'.]

Normally, Sonatype suggests that you use the Maven 'release' plug-in and the 'release:prepare' and 'release:deploy' steps, but we were unable to get those things to correctly *sign* a tagged release in GitHub and to get the new '2.x.y.z' branch to merge to 'main', so instead, I will just be documenting the steps that we will be following manually. (Life sucks sometimes.)

3.1 Preparatory work

Do all the work necessary to prepare this release branch, e.g., editing the pom.xml to change the version # (typically by removing the "-SNAPSHOT" portion), preparing release notes to be placed in the 'documentation' directory, etc.

All of the following steps, until otherwise noted, should be done on your new release branch, '2.x.y.z'.

To do that, first ensure you are on the correct branch,
```
$ git checkout 2.x.y.z
```

replacing x.y.z as appropriate.

3.1.1 Review output from executing Maven 'versions' plugin goals

Execute the following Maven goals with the 'versions' plug-n, review the output, and analyze whether or not updates need to be made to ESAPI's pom.xml file. (Note: According to Dave

Wichers and information he's gathered from the "Maven versions forum", the 'versions' plugin will miss some things unless the '-U' (aka, '--update-snapshots') is provided.) The following checks are strongly suggested.

Checking for new versions of plugins

To get information about newer versions of plugins that you are using in your build, just invoke display-plugin-updates goal.

```
$ mvn -U versions:display-plugin-updates
```

Edit the "pom.xml" file to update all of these plugin dependencies[1] so the are the latest version reflected from the previous output. Ideally, do a safety check at this point to ensure that "mvn site" still works as expected.

Checking for new versions of dependencies

To get information about newer versions of dependencies that you are using in your build, just invoke the display-dependency-updates goal.

```
$ mvn -U versions:display-dependency-updates
```

Edit "pom.xml" to update it to the latest dependencies. Run 'mvn site')using the NVD API key, as mentioned above) to ensure that everything works as expected. This should also execute the 'animal-sniffer-enforcer-rule' plugin to see if all the ESAPI dependencies are compatible with Java 8 and all the other dependencies, but the error messages are a bit obtuse and its probably best to use that as a defense-in-depth check rather than as the sole means of validation.

Checking for new versions of specified by properties

To get information about newer versions of dependencies that you are using in your build, just invoke the display-property-updates goal. (Note that the 'fluido' version has been a bit touchy as well; run 'mvn site' (see instructions above) and look at 'target/site/index.html' in your browser to make sure everything looks good before keeping any 'fluido' version updates.)

```
$ mvn -U versions:display-property-updates
```

---

1   NOTE: Recently (after 4.0.0-M5), the updates to maven-site-plugin have been breaking 'mvn site' so before you get to far, make sure to run 'mvn site' to make sure any new update to that plugin doesn't break something.

3.1.2 Create release notes for the 2.x.y.z release

For ESAPI, until we have a process for creating an acceptable looking CHANGELOG, we are going to be manually creating release notes files. We use **DOS text format** with a '.txt' file because, well Windows notepad just doesn't handle *nix line termination (newlines) gracefully. Sigh.

To assist you in writing the new 2.x.y.z release notes, you execute the 2 scripts "createVarsFiles.sh" and "newReleaseNotes.sh" in the following order. do the following, by answering the prompts. Here the input is show in red and the example shown here is for the **2.5.1.0** release.

```
$ cd ./scripts
$ ./createVarsFiles.sh
Enter release # for NEW ESAPI version you are doing release notes for: 2.5.1.0
Enter release # for the PREVIOUS ESAPI version: 2.5.0.0
Enter (planned) release date of NEW / current version you are preparing in YYYY-MM-DD
format: 2022-11-27
Enter release date of PREVIOUS ESAPI version in YYYY-MM-DD format: 2022-07-20
You entered:
=================================================
VERSION=2.5.1.0
PREV_VERSION=2.5.0.0
YYYY_MM_DD_RELEASE_DATE=2022-11-27k
PREV_RELEASE_DATE=2022-07-20
=================================================gi

Are ALL your previous answers correct? (y|n): y
$
$ ./newReleaseNotes.sh 2.5.1.0
-rw-rw-r-- 1 kww kww 7718 Nov 26 17:54 esapi4java-core-2.5.1.0-release-notes.txt

Now move the file 'esapi4java-core-2.5.1.0-release-notes.txt' to the 'documenation/' directory
and finish editing it there. Be sure to remove all the instructional lines starting with @@@
before committing it to GitHub.
```

Move the resulting "esapi4java-core-2.x.y.z-release-notes.txt to the "documentation/" directory and follow the instructions by searching for the '@@@@' markers in the file.

Once you finish editing thew new file, you will need to add it to the GitHub repo via 'git add' / 'git commit'. Make sure that you have moved it to under ESAPI's 'documentation/' directory though before adding it to GitHub. The new file for the release notes should be called
        documentation/esapi4java-core-2.*x.y.z*-release-notes.txt

where '2.x.y.z' is the new release number. Note if you are creating release notes, but releasing a release candidate, RC#, it is advised that you omit the "RC#" portion from the release notes file name.

The contents of this file should contain details of the GitHub issues that were closed since the previous release and any important / noteworthy documentation, such as newly deprecated methods or classes, newly removed methods or classes that were previously deprecated, etc. (In sort, consider it important if it is something that will or is likely to break backward compatibility now or later.) Also, try to give proper acknowledgements to those who helped, such as those submitting significant PRs, etc. See the file 'documentation/esapi4java-core-2.2.0.0-release-notes.txt' for an example.

Then, get it committed to the `2.x.y.z` branch
```
$ git add documentation/esapi4java-core-2.x.y.z-release-notes.txt
$ git commit -m "New release notes for ESAPI 2.x.y.z" \
        documentation/esapi4java-core-2.x.y.z-release-notes.txt
```

3.1.3 Update the ESAPI pom.xml for the 2.x.y.z release

Edit the pom.xml, as appropriate. Minimally, you will want remove the '-SNAPSHOT' from the <version>, possibly replacing it with '-RC#' if you are just releasing a "release candidate" or just deleting the '-SNAPSNOT' altogether if this is to be an official release.
```
$ gvim pom.xml    # Or whatever your favorite text editor is
$ git add pom.xml
$ git commit -m "Bump release to new release number." pom.xml
```

3.1.4 Create signed tag for release
Create the signed tag via git:
```
$ git tag -s esapi-2.x.y.z -m "Signed tag for ESAPI 2.x.y.z"
```

You normally should be prompted for a passphrase for your GPG private signing key. E.g.,

```
$ git tag -s esapi-2.x.y.z -m "Signed tag for ESAPI 2.x.y.z"
You need a passphrase to unlock the secret key for
user: "Kevin W. Wall (GitHub signing key) <kevin.w.wall@gmail.com>"
2048-bit RSA key, ID 8A2A524F, created 2016-02-03
```

and the GPG agent will pop-up for you to enter your passphrase. That agent will not allow you to do keyboard input into other windows unless you hit 'Cancel', in which case you will see a prompt at the command line that looks like this:

```
gpg: problem with the agent - disabling agent use
Enter passphrase:
```

Beware however, that you will not be prompted if you have not created a passphrase to protect

your private key (ill-advised) or if you are using an agent such as gpg-agent and have selected the check-box to "Automatically unlock this key, whenever I'm logged in".

If you do not intend on memorizing your GPG passphrase, you should keep it somewhere secure though. For me, my GPG passphrases are stored in my Password Manager (PasswordSafe), with the one used for ESAPI filed under "GPG Signing Key for GitHub".

3.1.5 Verify the signed tag

It's important to make sure that the digital signature of the tag worked, so we want to verify that the tag was successfully signed. There are two ways to do this. One is to use the `git tag -v <tagname>`' and the other is to use `'git show <tagname>'`. They have different outputs that you need to look for, so we'll cover them both here. However both require access to the public signing key, so if you wish to verify a signature that someone else added, you first need to import their public signing key on your keyring. (Obviously, you will already have your own signing key pair on your own keyring or the previous step would not work, so no special additional steps are needed for the signature verification step that follows.)

To verify the tag signature using `'git show'`, use same tagname for argument; i.e.,

```
    $ git show esapi-2.x.y.z
```

This should show you the standard PGP signature block. If you do not see the

```
        -----BEGIN PGP SIGNATURE-----
        -----END PGP SIGNATURE-----
```

signature block as defined in RFC 4880 as part of the `'git show'` output, you have done something wrong.

To verify the tag signature using 'git tag -v', again use the same tagname as above for the argument. E.g.,

```
    $ git tag -v esapi-2.x.y.z
```

The last two lines of output should be from gpg and look something like this:

```
gpg: Signature made Sun 07 Apr 2019 12:26:57 AM EDT using RSA key ID 8A2A524F
gpg: Good signature from "Kevin W. Wall (GitHub signing key) <kevin.w.wall@gmail.com>"
```

If you do not see that, that means a signature was not applied. If there is no tag, the last two lines will look like:

```
        error: no signature found
        error: could not verify the tag '<tagname>'
```

IMPORTANT NOTE: If you have to make any commits after you have created the signed tag release, you will have to go back and delete the tag (via '`git tag -d tagname`'), and then recreate it by going back to the steps starting in section 3.1.4. Tweaking the pom.xml file is often needed if you run into issues that you discover while executing '`mvn site`'

Finally, we want to make sure we haven't messed anything up, so lets verify that:

```
$ mvn site # May take a while because it runs Dependency Check!
```

Make sure there are no failed JUnit tests. Also check the file 'target/dependency-check-report.html' to see if Dependency Check found any vulnerabilities. (It's easier to do that by opening the file in a web browser.) If the Dependency Check report does indicate vulnerabilities, consider updating libraries with identified CVEs by tweaking the appropriate dependencies in pom.xml and redo the verification step until everything passes. Once everything looks good, we are now ready to merge our new release branch, "2.x.y.z" to "main".

If you've made it this far, we are finally ready to push it to the official (upstream) ESAPI repository in GitHub:

```
$ git checkout 2.x.y.z
$ git push origin 2.x.y.z
```

Also, **push** this *new* signed tag to server (GitHub).

```
$ git push origin esapi-2.x.y.z
```
Alternately, push **ALL** the tags.
```
$ git push origin --tags
```

3.2 Switch back to the 'main' branch which we use for releases

Now, change back to the 'main' branch. Note we are assuming here that the 'develop' branch is "frozen", or at least we are treating it as such for now. We are now going to merge our 2.x.y.z release branch that we just prepared into 'main'.

First, let's get back to the 'main' branch that is always supposed to reflect our latest release.

```
$ git checkout main
```

Since we really don't do any development on 'main' but instead work off 'develop', this next step is just a safety check. We really don't expect to find anything that will need to be fetched and merged from our repo. If there is, stop and ask one of the ESAPI project co-leaders how to proceed.

```
$ git pull origin main     # This is just a safety check!
```

Next, merge our release branch, 2.x.y.z, to 'main'. Note: There will be at least one merge conflict in the pom.xml, for the release number, but hopefully, we already have dealt with them all by now). If so, deal with them appropriately. (Run 'git status' and it will tell you what you need to do.) The default commit message is fine to use.

```
$ git merge 2.x.y.z
```

Finally, push the updated 'main' to GitHub

```
$ git push origin main
```

4 Upload to Sonatype Nexus & Maven Central

[Many of these instructions are based on those from Dave Wichers, which he used for AntiSamy 1.5.8]

4.1 Publish the artifacts to Sonatype's OSS Repository Hosting (OSSRH) via Maven

From the 'main' branch', execute the command:
```
$ mvn deploy
```

Running 'mvn deploy' will first run 'mvn test' and then build and sign the artifacts before eventually uploading them to OSSRH Sontatype staging area.
If you get 'Access Denied' then your account doesn't have permission to publish to that particular project. If you get 'Unauthorized' that means you aren't even logging in to Sonatype successfully. (Check your ~/.m2/settings.xml file).

(Note: if you correctly set up your ~/.m2/settings.xml file first, you might be able to use:
```
$ mvn release:prepare
```
followed by
```
$ mvn release:deploy
```

to accomplish these many of these steps, but I didn't do it that way because I was unable to get the 'release:prepare' to correctly sign a tag. It claimed that it did, but didn't. Sigh.)

4.2 Login to https://oss.sonatype.org/

Visit **https://oss.sonatype.org/** in your favorite web browser and click on "Log In" in the upper right corner. The user name is 'esapi-team'. The password is the same as the credentials you used to configure ~/.m2/settings.xml under the earlier section "Configure for ESAPI's user name for the Sonatype repo" at the beginning of this document. If you don't have the password, you can get it from Kevin Wall over a secure communications channel such as Signal or encrypted email. Once you get the password, store it in a your favorite Password Manager.

4.2.1 On the Sonatype web site

Click on Staging Repositories (Under Build Promotion) and your new staging repo by entering your project name (e.g., esapi) in the staging repository search box (towards top right).

Click on the row found (assuming it was). Then, in the panel below, navigate to the Content tab, and find the release version you just published, to make sure they are all there. (You should fine the ESAPI .jar file, the pom, sources, javadoc types, along with `.asc` files for each.)

[Note: We are using the Nexus Repository Manager 2 in case you are looking for help.]

4.2.2 'Closing' the repository

In the navigation panel on the left-hand side, click on 'Staging Repositories' (under "Build Promotion") search from the form entry by typing in 'esapi' and hit enter. You should normally only see a single "hit" (unless a previous release attempt was abandoned and never released, in which case it should probably be deleted). The one that you should find should be for the artifacts just uploaded via 'mvn deploy'.

Check to see if all 20 of the files have been successfully uploaded. For each of the 8 artifacts X (where 'X' is "esapi-2.x.y.z-cyclonedx.json", "esapi-2.x.y.z-javadoc.jar", "esapi-2.x.y.z-sources.jar", "esapi-2.x.y.z.jar", esapi-2.x.y.z-jakarta-javadoc.jar", "esapi-2.x.y.z-jakarta-sources.jar", "esapi-2.x.y.z.-jakarta.jar, and esapi-2.x.y.z.pom"), there should be 4 files that get uploaded with the names:
   - X
   - X.asc
   - X.md5
   - X.sha1

Note that the 'jakarta' related artifacts will only exist for ESAPI releases after 2.5.3.0, which is when we first started creating them. For example, for the 2.3.0.0 release, one would only expect to find 5 of those 8 related artifacts in the staging repository:

nexus repository manager

Sonatype™ «	Welcome	Staging Repositories ⊠

🔄 Refresh 📋 Close 🔧 Promote 🔧 Release 🔧 Drop

☐	Repository ▲		Profile	Status

📁 orgowasp-1689

Summary	Activity	**Content**

🔄 Refresh Path Lookup: [   ]

Artifact Search ▲

[   ] 🔍

Advanced Search

Views/Repositories ▲

Central Statistics
Repositories

Build Promotion ▲

Staging Profiles
Staging Repositories
Staging Ruleset
Staging Upload

Help ▼

☐ 📁 owasp
  ☐ 📁 esapi
    ☐ 📁 esapi
      ☐ 📁 2.5.3.0
        📄 esapi-2.5.3.0-cyclonedx.json
        📄 esapi-2.5.3.0-cyclonedx.json.asc
        📄 esapi-2.5.3.0-cyclonedx.json.md5
        📄 esapi-2.5.3.0-cyclonedx.json.sha1
        📄 esapi-2.5.3.0-cyclonedx.xml
        📄 esapi-2.5.3.0-cyclonedx.xml.asc
        📄 esapi-2.5.3.0-cyclonedx.xml.md5
        📄 esapi-2.5.3.0-cyclonedx.xml.sha1
        📄 esapi-2.5.3.0-jakarta-javadoc.jar
        📄 esapi-2.5.3.0-jakarta-javadoc.jar.asc
        📄 esapi-2.5.3.0-jakarta-javadoc.jar.md5
        📄 esapi-2.5.3.0-jakarta-javadoc.jar.sha1
        📄 esapi-2.5.3.0-jakarta-sources.jar
        📄 esapi-2.5.3.0-jakarta-sources.jar.asc
        📄 esapi-2.5.3.0-jakarta-sources.jar.md5
        📄 esapi-2.5.3.0-jakarta-sources.jar.sha1
        📄 esapi-2.5.3.0-jakarta.jar
        📄 esapi-2.5.3.0-jakarta.jar.asc
        📄 esapi-2.5.3.0-jakarta.jar.md5
        📄 esapi-2.5.3.0-jakarta.jar.sha1
        📄 esapi-2.5.3.0-javadoc.jar
        📄 esapi-2.5.3.0-javadoc.jar.asc
        📄 esapi-2.5.3.0-javadoc.jar.md5
        📄 esapi-2.5.3.0-javadoc.jar.sha1
        📄 esapi-2.5.3.0-sources.jar
        📄 esapi-2.5.3.0-sources.jar.asc
        📄 esapi-2.5.3.0-sources.jar.md5
        📄 esapi-2.5.3.0-sources.jar.sha1
        📄 esapi-2.5.3.0.jar
        📄 esapi-2.5.3.0.jar.asc
        📄 esapi-2.5.3.0.jar.md5
        📄 esapi-2.5.3.0.jar.sha1
        📄 esapi-2.5.3.0.pom
        📄 esapi-2.5.3.0.pom.asc
        📄 esapi-2.5.3.0.pom.md5
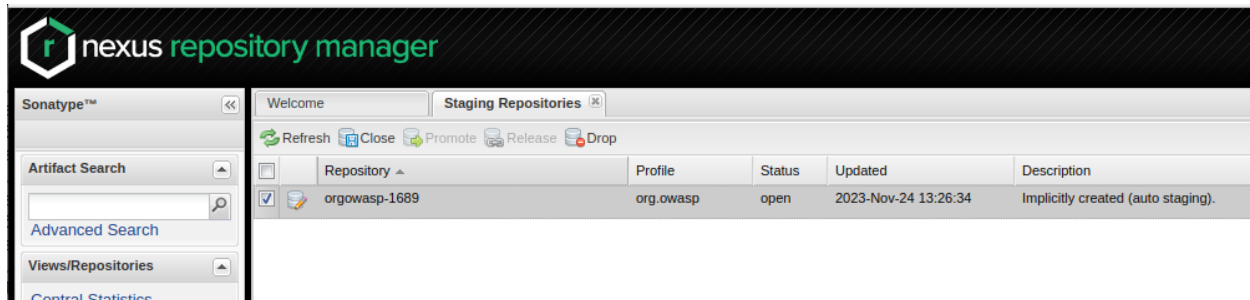   📄 esapi-2.5.3.0.pom.sha1
      📄 maven-metadata.xml
      📄 maven-metadata.xml.md5
      📄 maven-metadata.xml.sha1

Once this is confirmed, click on both of the staging repositories (so you have multiple selections) and then click on the 'Close' repository button (right under the Staging Repositories tab name). If it works, do back-flips. If it doesn't, you may have to cross your fingers and sacrifice a male unicorn and then swing it 3 times around your head in a clockwise direction at midnight during a full moon to get it to work. (Fortunately it worked for me, but I skipped the back-flip because my bad back would never forgive me. :-)

- You can monitor its staging repositories closure status via the activity tab below, by clicking on Activities-→ close:

- Refresh the Staging repository before checking each time via the Refresh button in the UI.

==IMPORTANT==: If the "close repository" successfully completes, you then click on the **Release** button *under the Staging Repositories* tab name, just to the right of the Refresh icon.

- **Note**: You might have to refresh the repository to get that Close button to "light up" so it is selectable.
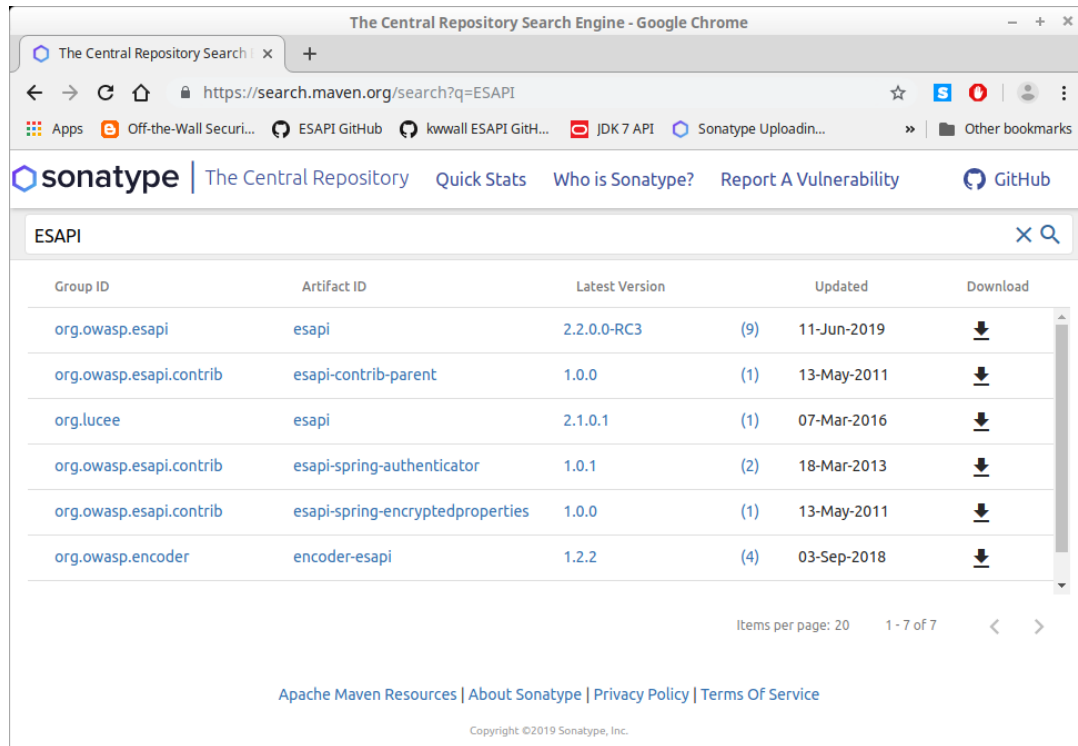
For the 'Description' that pops up, just type in "ESAPI 2.x.y.z release" where the "2.x.y.z" corresponds to the release number.

Once the release is done, Sonatype say it takes about 10 minutes to show up on the Sonatype Nexus web site for it to become available for download, but that a search for it might take 2 hours to update before you can find it that way. (Note: I was told 2 hours, but the last 2 times I did releases it took between 6 to 8 hours each time! For release 2.2.3.0, it was even worse. It took almost 12 hours for it to show up https://search.maven.org/search?q=ESAPI and about 24 hours to show up at https://mvnrepository.com/artifact/org.owasp.esapi/esapi.)

4.3 Validating the new ESAPI release has been pushed to Maven Central

The simplest way to see if the new ESAPI release has been pushed to Maven Central is to just use the URL https://search.maven.org/search?q=ESAPI and look at the results. The latest release should show up at the top.

For example, here is a screen shot showing the ESAPI 2.2.0.0-RC3 release when it was the most recent release:

5 Final release preparation for release 2.x.y.z

5.1 Create a 'Release' on GitHub

We also need to create a 'release' on GitHub so that we will have a place to download the 'configuration' jar that we will build along with its signature.

Make sure that you are logged into the GitHub web site before proceeding.

5.1.1 Create the new 2.x.y.z release

Navigate to https://github.com/ESAPI/esapi-java-legacy/releases in your web browser and select the 'Draft a new release' button on the right hand side.

For the 'Release title' choose '2.x.y.z' (expanding it to the actual release number, of course'. Do *not* preface it with 'release' or 'version' or 'ESAPI' or 'esapi-', etc.; just simply '2.x.y.z'.

For the description section ('Describe this release') write something like this:

> Release notes for ESAPI release 2.x.y.z are located at:
>
> https://github.com/ESAPI/esapi-java-legacy/blob/develop/documentation/esapi4java-core-2.x.y.z-release-notes.txt
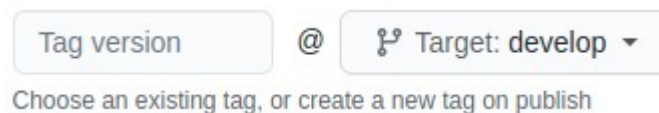>
> Note the file "esapi-2.x.y.z-configuration.jar" contains the default ESAPI configuration files under 'configuration/' (ESAPI.properties, validation.properties, etc.) and the file "esapi-2.x.y.z-configuration.jar.asc" is a GPG signature of that jar file made by <<Your Name Here>>

If there are are any applicable Security Advisories, mention them in these release notes as well. For an example, see:

> https://github.com/ESAPI/esapi-java-legacy/releases/tag/esapi-2.2.3.0

which is a little fancier than I usually do.

In the text box where it says 'Tag version', e.g.:

from the pull-down mention, change it to select 'main' instead of 'develop'. Then type in the signed tag name you gave to this release earlier, which if you were following instructions, should be named 'esapi-2.x.y.z'.

Click on '**Save draft**'. (We will return to this page to complete the 'Publish release' step momentarily.)

5.1.2   Manually create the esapi-<release>-configuration.jar file

We want the 'configuration/' directory to also be jarred up, signed, and uploaded to the GitHub release that we are preparing. That directory contains various configuration files that developers would use for production. Note that we do *not* want these files placed into the esapi-<release>.jar file (especially because in the past and we may again, use 'jarsigner' to sign the ESAPI jar) an forcing users to extract the ESAPI.properties file and then place it back into the jar would break the signature. And if the developers simply extract it, copy it elsewhere to edit and use *that* copy, all is good until their production copy gets dropped somewhere not in the application class path and then ESAPI mysteriously starts using the ESAPI.properties file that is in the ESAPI jar unbeknownst to the development team until something goes wrong. We also deliberately did not want it to end up in the esapi-<release>-sources.jar because we don't want developers to have to download the entire ESAPI source code just to get the ESAPI configuration files. So instead we so we will manually create a separate configuration jar, sign it, and upload it to to the corresponding GitHub 'release'.

First, create the jar (without a manifest file) using the following command:

```
$ jar -cMvf target/esapi-<release>-configuration.jar configuration/
```

where *<release>* is going to be replaced by your current release (e.g., 2.2.0.0.-RC2).

If you assigned a signing key, use this command to discover the key ID:
```
$ git config --global --get user.signingkey
```

and use it for *keyID* in the following command:

```
$ gpg --use-agent --local-user keyID --armor --detach-sign \
      --output target/esapi-<release>-configuration.jar.asc  \
      target/esapi-<release>-configuration.jar
```

Note that the use of '--use-agent' is optional, but recommended so you don't have to keep reentering your GPG passphrase. Check your target directory for the 2 'configuration' artifacts. E.g.,
```
    $ ls -l target/*configuration*
    -rw-r--r-- 1 kww kww 34067 Apr 21 21:08 target/esapi-2.2.0.0-RC2-
configuration.jar
    -rw-r--r-- 1 kww kww   473 Apr 21 21:12 target/esapi-2.2.0.0-RC2-
```

```
configuration.jar.asc
```

We will describe how to upload these two 'configuration' artifacts to the GitHub 'Releases' in the next section.

5.1.3 Manually upload the 'configuration' jar and it's corresponding .asc file to your 2.x.y.z GitHub 'Release'.

Go back to the GitHub web site and find your unpublished release and click on 'Edit' to edit it. (If you can't locate it, you can just use the link, https://github.com/ESAPI/esapi-java-legacy/releases/edit/esapi- 2.x.y.z (as long as you are still logged into GitHub).

In the area after the 'description' text box (which you already should have completed from above), locate the section that says 'Attach files by dragging & dropping, selecting or pasting them.' and either click on that to select the two files 'esapi-2.x.z-configuration.jar' and 'esapi-2.x.z-configuration.jar.acs' under your 'target/' directory or drag and drop them to the area that says 'Attach binaries by dropping them here or selecting them.'. (Note: If you want to use the 'drag & drop' to attach these 2 files to the new GitHub release, I have had better luck with Google Chrome than Mozilla Firefox, but YMMV. Both of them work if you want to "browse to directory".)

Finally, if this is a '**Release Candidate**' release (e.g., '2.x.y.z-RC2'), make sure you select the 'This is a pre-release' check box. If it is an official release, make sure that checkbox is unchecked.

Then click on the 'Publish release'. (Or if you forgot something and needed to go back and edit it, click on the 'Update release' button.)

5.2 Update the OWASP ESAPI wiki pages

Hang in there, you are almost done. We need to update all the ESAPI pages on the OWASP wiki that mentions a release. You can either do this directly from the page by going to the bottom of the page and clicking on the 'Edit on Github' link at the bottom, or the easier way to just 'clone' that particular GitHub repo and edit the files locally and then push them back up to the repo.

```
$ git clone git@github.com:OWASP/www-project-enterprise-security-api.git
```

The file that you will need to edit is:

 • tab_java_ee.md

That file should have a reference to the latest 2.x.y.z release.

In the 'tab_java_ee,md' file, search for 'Current release:' and 'Release notes:' and update that information (including the date) and the link to the release notes. (Important note: Refer to the release notes from the 'develop' branch so if we need to add some critical information to it such as a work around for some problem, etc., we can do so without having to put out another release. (Because the release notes are part of the ESAPI 'source' jar that gets uploaded to Maven Central and that jar is required to be signed.) Alternatively, if you know what the current release is, you can just search for it.

Add and commit the files with a comment something like "Update for ESAPI release 2.x.y.z" and then do a 'git push origin'. The wiki page usually gets the updates within a few hours, but if you don't see your updates within the following date, contact myself or Harold Blankenship.

5.3 Make a release announcement

Announce to ESAPI Users and Dev mailing lists on Google Groups (esapi-project-users@owasp.org and esapi-project-dev@owasp.org respectively) see pom.xml and to the **@owasp** Twitter feed that a new ESAPI release is available. Maybe announce on the OWASP Leaders List and OWASP Slack channel #owasp-esapi too if you are signed up for those. And if the release is a minor release and not just a patch release (e.g., 2.1, 2.2, 2.3, but not (say) 2.1.1.1 or 2.2.1.0), then consider contacting OWASP to request a notice be published in the next OWASP Connector newsletter. See https://www.owasp.org/index.php/Category:OWASP_Newsletter for further details. All release announcements should minimally direct people to reference the ESAPI release notes in the 'documentation' directory, via a GitHub URL, for additional details.

6 Prepare for 'develop' branch for the *next* ESAPI release

6.1 Merge latest changes from 'main' to 'develop' branch.

Change to the 'develop' branch and merge 'main' to it:
```
$ git checkout develop
$ git merge main
```
Address any merge conflicts that come up as needed, and then push your changes to GitHub:
```
$ git push origin
```

6.2 Edit pom.xml to bump to the *next* snapshot

NOTE: We do ***not*** want this to be in the 'main' branch since that will now reflect the latest official ESAPI release. You should be on the 'develop' branch for this.
```
$ gvim pom.xml
```
Change

        <version>2.x.y.z</version>

to whatever you want your next version to be. E..g., let's say you are currently at version 2.2.0.0 and you want your next version to be 2.2.1.0, then you would change it to:

        <version>2.2.1.0-SNAPSHOT</version>

(be sure to include the "-SNAPSHOT" part).

Add and commit pom.xml
```
$ git add pom.xml
$ git commit -m "modifying pom.xml for next planned release." pom.xml
```

Rebuild and install.
```
$ mvn clean install
```

Push 'develop' branch to GitHub
```
$ git push origin develop
```

6.3 Time to celebrate

You are finished. Go celebrate. Have a Guinness at the Foo Bar even. (ESAPI ObjFactory joke; see the [ObjFactory Javadoc](#) while you enjoy that brew for explanation. And sorry for the bad pun; not sorry.)

7 References

Sonatype OSS Repository Hosting Guide -- https://central.sonatype.org/pages/ossrh-guide.html

Sonatype OSS Repository Hosting (OSSRH) Publishing Guide –
https://central.sonatype.org/publish/publish-guide/#deployment
Generating a Signing Key using GPG -- https://help.github.com/articles/generating-a-new-gpg-key/
Releasing your Deployment from Sonatype OSSRH –
https://central.sonatype.org/pages/releasing-the-deployment.html
Uploading Components (to Sonatype) – https://help.sonatype.com/repomanager2/using-the-user-interface/uploading-components
Nexus M2Settings Maven Plugins – https://help.sonatype.com/repomanager2/managing-maven-settings/nexus-m2settings-maven-plugin