



OWASP 2021
>irtual
APPSEC

PRESENTED BY: KEVIN W. WALL
Senior Application Security Engineer
Guaranteed Rate


OWASP ESAPI – A Retrospective: The Good, the Bad, & the Ugly

usa.globalappsec.org


Pencils down

- This talk is licensed under:
Public presentations Copyright © 2021 -- Kevin W. Wall – All Rights Reserved. Released under Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License (CC BY-NC-SA 3.0 US).
- Slide deck will be uploaded to:
<https://github.com/kwwall/presentations>
- Speaking for myself and not OWASP, not current or previous employers
- You can contact me via:
 - Email at: kevin.w.wall@gmail.com
 - Twitter or the OWASP Slack: [@KevinWWall](#)


Objectives of this talk

- Library development is different than typical product development.
 - Early success may be a problem in the long haul.
 - FOSS is different than business development.
- 


Why I pretend to know what I'm talking about

- 20+ years systems programming, 20+ years in AppSec.
 - Previously developed several proprietary systems libraries, including one for AppSec.
 - Involved in ESAPI since June, 2009.
 - ESAPI dates back to at least Sept 2007 as per mailing list archives
 - Have been ESAPI co-lead since 2011.
 - Buzzword free talk. No mention of blockchain, AI, or quantum computing!
- 
- A decorative blue wavy line at the bottom of the slide, consisting of a solid blue area at the very bottom and a lighter blue area above it, separated by a white wavy line.

The Approach

- I will discuss 3 perspectives of “Lessons Learned”:
 - Process
 - People
 - Technical
 - For each perspective, I will have 4 sections:
 - Overview – what this perspective is about
 - The Good – What was done right
 - The Bad – What was done wrong
 - The Ugly – Neither good or bad, but things like ugly hacks that worked, etc.
- 
- A decorative blue wavy bar at the bottom of the slide, transitioning from a darker blue on the left to a lighter blue on the right.


Process perspective: Overview

- Libraries and applications are different.
 - Design
 - Testing
 - Documentation
 - Client lock-in
 - Security libraries vs other, non-security libraries
 - Not eating our own AppSec dog food
 - Effects of later process change
- 
- A decorative blue wavy bar at the bottom of the slide, consisting of two overlapping curved shapes in different shades of blue.


Process perspective: The Good

- Things started small, with simple expectations
- Original vision was a library of interfaces only


Process perspective: The Bad

- No clear plan for release migration from 1.x to 2.x
 - Failure to eat our own AppSec dog food
 - No plan for becoming a victim of early success
- 
- A decorative blue wavy bar at the bottom of the slide, consisting of a solid blue area on the left and a lighter blue area on the right, separated by a white wavy line.

Process: The Bad: Release migration

- Did we have enough feedback from 1.x to do this?
 - Allow more time for release cycles for libraries than applications
 - Can't use libraries out-of-the-box; need to write code to give feedback.
 - Semantic versioning changes to major # usually implies interface changes, which there were little.
- 


Process: The Bad: Ignoring AppSec best practices

- Could be perceived as “do as I say, not as I do”
 - No threat modeling
 - No formal code reviews (until very late); some informal SAST
 - Most unit tests only checked “sunny day” paths
- 

Process: The Bad: Early success

- The Encoder and Validator became very popular and dominated a lot of the requests and questions in the mailing lists.

Process perspective: The Ugly


- Little consideration that library dev differs from application dev
 - E.g., Agile emphasizes constant refactoring but has limited applicability to libraries once you have lots of clients. ***Can't break client code!***
 - How do you security test a security library?
- 
- A decorative blue wavy bar at the bottom of the slide, transitioning from a darker blue on the left to a lighter blue on the right.

People perspective: Overview


- How the people involved with ESAPI affected the process and product
- Succession of leadership



People perspective: The Good

- Top notch AppSec people worked on this and we had good leadership (Jeff Williams, then Jim Manico) ensuring it kept running smoothly.
 - Lots of initial enthusiasm up through the 2.0 GA release.
 - Technical disagreements handled professionally and transparently on a public mailing list.
- 


People perspective: The Bad

- Failure to realize:
 - Good technical leaders in the business world do not translate to the FOSS world.
 - Picking your next successor by suddenly stepping down and putting the 2 most involved committers as co-leaders is not a good strategy for success.
- 

People perspective: The Ugly




People perspective: The Ugly

- After 2.0 GA, pretty much all the other help disappeared, leaving Chris Schmidt and I to handle about 98% of the work.
 - The previous regime did not pass on any details of how to actually do a release.
 - Chris had to leave after a few years because of other commitments.
- 
- A decorative blue wavy bar at the bottom of the slide, consisting of two overlapping curved shapes in different shades of blue.


Technical / architectural perspective: Overview

- A discussion of technical decisions around architecture and design and what I learned.

Technical / architectural perspective: The Good

- It started with interfaces.
 - It was quickly realized that some implementations were needed, only if rudimentary.
 - Property driven approach to select implementation.
- 


Technical / architectural perspective: The Bad

- Monolithic approach was a bad idea.
 - Singletons were a terrible idea.
 - Some implementations were too successful (e.g., output encoders).
 - As a result, ESAPI quickly became thought of as a reference implementation rather than a set of interfaces.
 - Many individual interfaces of components comprising a single security control became bloated.
 - Uncontrolled growth of ESAPI properties.
- 

Technical: The Bad: Monolithic approach

- ESAPI 2.x has 12 direct compile-time dependencies, which pulls in 14 additional transitive compile time dependencies.
 - If you want to use something like the ESAPI Encryptor, which requires no external dependencies if you use SunJCE and JUL for logging, you still pull in all 26 dependencies, even though you need zero.
- Has caused problems with vulnerable dependencies that are past end-of-life, such as Log4j 1.x
 - We can't completely eliminate log4j dependency (despite now having SLF4J support) without breaking backwards compatibility.

Technical: The Bad: Bloated interfaces

- The Encoder interface has 22 methods, *BUT*
 - The HttpUtilities interface has **60 methods**, many of which are very nuanced.
- 

Technical: The Bad: Uncontrolled property growth


- Adding a new property required this in the DefaultSecurityConfiguration class. E.g.,
 - A new public static final String to represent the property:

```
public static final String FORCE_HTTPONLYCOOKIES = "HttpUtilities.ForceHttpOnlyCookies";
```


- A new public method to access the property value:

```
/**  
 * {@inheritDoc}  
 */  
public boolean getForceHttpOnlyCookies() {  
    return getESAPIProperty( FORCE_HTTPONLYCOOKIES, true );  
}
```


Technical / architectural perspective: The Ugly

- Radical dependence on ESAPI properties and logging and exception handling soon made every component tightly coupled to SecurityConfiguration, Logger, and EnterpriseSecurityException. And transitively, to User (for Logging).
 - Too many major, ill-timed changes *at once*:
 - Changing from Google Code & Subversion to GitHub & Git
 - Changing from Ant to Maven
 - We trusted the NSA for a code review.
 - Important(?) open issues never addressed
 - Rules for locating ESAPI.properties too complicated
- 

Technical: The Ugly: Issues never addressed

- If you look at the undone tasks designated by TODO, FIXME, OPENISSUE, etc. comment tags, we see:
 - TODO: 62 total; 43 in src/main, 19 in src/test
 - FIXME: 9 total; 5 in src/main, 4 in src/test
 - OPENISSUE: 4 total; all in src/main
 - DISCUSS: 37 total; 26 in src/main, 11 in src/test
- 

Technical: The Ugly: Finding ESAPI.properties

- One of the most frequently asked question on Stack Overflow.
 - Documented in DefaultSecurityConfiguration, not SecurityConfiguration.
 - Rules *roughly* are:
 - 1) Inside a directory set with a call to `SecurityConfiguration.setResourceDirectory("full-path-dirname")`.
 - 2) Inside the `System.getProperty("org.owasp.esapi.resources")` directory.
 - 3) Inside the `System.getProperty("user.home") + "/esapi"` directory.
 - 4) The first "esapi" directory on the classpath, for several different possible class loaders.
- 

Conclusions

- Monolithic libraries are a thing from the past.
- ESAPI 3 (aka 3SAPI to us ESAPI devs) will be done very differently. (But that's another talk.)



Questions

?



OWASP 2021
>irtual
APPSEC

THANK YOU!