Content Development Best Practices

Kent Landfield

Purpose

There are many areas of content development that are problematic. Content developers vary in the quality of content they produce mainly due to not having a means for lessons learned to be propagated. This session will focus discussion on identifying best practices and how best to capture them.

Avoid Content Bloat

1. Don’t write one check per operating system. Write one check that applies to multiple operating systems. Consolidation is a good thing.

2. Use external variables. This reduces the number of checks needed to be written since a single check can be used for multiple input variables.

3. Reuse content / IDs where possible. If you have 30 checks that all look for something in the system32 directory, don’t create 30 different objects and vars that all point to system32. Reuse the one object and var 30 times.

Versioning of Content

- Update versions every time you edit anything.
- It doesn’t matter if you’re only updating the comment on a state, update that version and the version of the definition as well to ensure there are no issues with your updated check being used.
- Assure your content is updated appropriately in products

Build extensibility/reusability into your content

- This will help to minimize your overall content base, need for development of new content, and reduce the effort and time needed for maintenance tasks.
- Example would be that some settings in Windows are checked the same way; however, the reference (i.e. CCE), textual description & configurable values may vary between versions.
- Create a base Rule/Value in XCCDF which only includes information shared across the board. Then for each OS extend the base Rule/Value to incorporate unique information/values.
- If you write everything static, any errors in implementation will need to be addressed ‘n’ times instead of just once.
- Also, if different developers author the content to perform the same task they may have different approaches which may impact consistency in results. The OVAL definitions should be generic, only describing what/how the definition checks.
- This will assist in ensuring only the correct information is included in results minimizing the potential for confusion.

Include Concise Comments

- Include concise comments in your content
- This is necessary as sometimes the only way to figure out issues is to dig down into the XML
- As with source code, a little extra time upfront can help shorten the time to resolution
- Additionally, tools may use this information in some of their results so it may have added value to less technical users.

Work within the language

- Do not attempt to work around intentional restrictions of the language, no matter how good the idea may be. Your content may not function as intended with all tools.

- This stems from experience with audit configuration checks which allowed pattern matching to an enumeration which was restricted to the operations equal and not equal.

- Better approach is to define what needs changed in the spec and voice a solid argument as to why it should be changed. Convince the community to support getting the change(s) incorporated into the spec.

Careful Extending Definitions

- Do not create definitions which only extend definitions which in turn only extend definitions.
- This can make for quite the rats nest when trying to follow the XML and figure out what's really being done.
- Due to the need for tracing it may also result in circular logic which may throw a tool into an infinite loop.

Fill Text fields

- Text fields should provide for some form of human readable text as supported by the spec. Including text such as:

>desc<This checks password strength. It is always good to have extremely long complex passwords because if you can't remember it no one will figure it out.>desc<>manual_check<To check this confirm that minimum password length approaches inifinity, all characters sets are used, none of the characters determined to introduce vulnerabilities in algorithms are permitted, no dictionary searchable words are used including their reverse spelling, no palimdromes, no character substitutions, and that they are changed every other day. >/manual_check<>automated_chck<>/automated_chck<>fix_proc<Configure system to meet check requirements>/fix_proc< ;>automated_fix<>/automated_fix<>reference<>/reference<...

- This should be put into plain text, without escaped characters/tags, or XHTML. Make it more readable across the entire suite of SCAP Validated tools and if using the later will allow for easy, automated separation of the textual elements.

Assure Datatypes and Operations Match

- Make sure data types and operations match
- For example, between XCCDF and OVAL, between states and variables.
- This is just a simple QA issue which can arise from long days & nights of looking at XML and things beginning to blur together.

Consider performance impacts

- Consider the worst case scenario when using behaviors which are greedy/expensive.
- For example, resolving groups on a local system versus a million node domain has very different results.
- In the later is may result in a DoS.

Use the least version principle

- If something can be done in a lower version of a spec then write it using that lower version.
- This should allow for the largest number of tools to correctly process content.
- Be careful not to mix differing spec versions in the same XML file.
- The minimum version for the entire file should be the ceiling of the versions required to implement all content contained within.

Concise, safe pattern matches

- Use concise, safe pattern matches.
- While '.*' may be concise and easy it may also allow for the processing of non-allowed, unexpected, and/or malicious data.
- Be sure not to mix and match PCRE and POSIX patterns in OVAL.

Test, test, test!!!

- Test your content!
- Make sure your content functions correctly in all use cases, to the best of your ability.
- It's of little use if it doesn't work or function as intended.
- Test to assure content runs on the intended systems and not on unattended systems.

What I’d like to see

- We need to learn from each other. Lessons learn by one needs to be shared for the benefit of others
- Need a way to document best practices for those that follow
- Need to see what we can do to incorporate some of this into the tools of the future