Thanks Anca
so as Anca mentioned we have these

various OSCAL models and ... Trestle
provides a set of capabilities to help

you edit, create and manipulate these
models and it helps you validate whether

the format
of the generated OSCAL document is

correct or not whether all the
constraints are satisfied as per the

and since these models are

interconnected for example a profile
refers to a catalog, component definition

refers to a catalog or profile, we also
validate based on those references

whether the set of controls that are
g Got used they are they are correct

or not or if there are any
inconsistencies over this all those
referential integrity constraints and all are verified over there pressure

also helps to integrate with various third-party tools and assessment tools

which require to convert the data from their native format to OSCAL and vice versa.

So we provide capabilities for format conversion you know and that's what we are showing at the bottom where these

OSCAL models are then pushed into different tools they work with that and then the results and all gets pushed back to the different entities

next slide please so uh present basically uses uh the defsec of patterns to help uh build the compliance artifacts
as code so as you know index deflector
ops all the content is managed and git
repository's code which helps provide
visibility across different teams since
the content different contents can be
owned by different teams they can be a
separate catalog owner profiler ssp
owner and component definition owner
having all the content and get and
provides the visibility to different
teams to all this content
it also helps to automatically validate
and verify these content whenever some
updates are pushed to this content we
use stressful command line interface
whenever any changes happen to to
validate whether these contents are as
per the OSCAL schema
and also all this uh editing and updates
are logged into it so that we can have
change tracking and approval processes over there to

to so that only the approved content gets updated and

we keep track of who is changing whatever content so so trestle fits very well in this whole def sec of pipeline based thing uh working with git to help enable the

the manipulation of oscar documents as code and get repositories

next slide please so this slide basically shows the uh high level

architecture of trestle and its various parts at the core we have the uh

various oscar models converted into programming language classes so we use

python so all these oscar models have a representation as python classes and
what it helps us is it helps to validate all the constraints on the more schema

so whenever we are creating and editing it and shows that the content is always compliant with the

oskar schema on top of that we have an adapter which helps us to read the oscar documents into into these OSCAL Python objects and convert it back to OSCAL
docs the base on top of this base we have our

set of tools to enable editing and authoring content authoring and transformation so tressal basically uses an opinionated directory structure

where all the oscal models and contents are stored and on top of that we have a

set of cli command line interface to help you import an existing oscal
content into tresel repository
edit and create new content merge the

content back and validate it and also so
we have command line interface for that

and we also have an sdk so that we can
use it programmatically

um since editing this auskal content in
json files directly is not very

easy for humans so what pressure also
provides is a set of markdown based

command line interface
what we allow is we convert this oscar
documents into markdown uh documents
where users can go and edit the content

especially the textual part of the
content where user have to provide all

this information and then we can convert
it back to oscar so so this enables

editing of all this content in get
repository in markdown format and then
automatic conversion back into the OSCAL format

we have a set of tasks and transformers for converting content in other formats

like spreadsheets and tool specific formats and all into oscar to bring this external content into tressel and we also have a set of plugins to extend the functionality for example if we need to do validation of fedramp sps and all we have built a separate uh repository where we have that code and that that can be attached to pressure using a plug-in mechanism and then we have various applications for one application is for managing the regulatory content and all and the approval workflows we have a specialized ssp creation workflows
in git repository and we also have different format conversions to and from Moscow such as the spreadsheet for document and other native formats and all next slide please

so this this

shows an example of using tetracell CLI for example there is a new user who's not very familiar with OSCAR objects they need help to manage these OSCAR files and given these OSCAL files can be pretty huge what preSell allows this to break these big OSCAL files into smaller files so here we are showing an example where the nosql catalog which is having multiple groups is divided into separate files one per group which users can go and edit and then we can
merge this content back into one single oscal file so using the split and merge 

functionality it helps easy editing of the auscal content directly using the 

command line interface 

uh next slide please 

the next requirement is around the for the using tresellis and sdk here we 

provide a set of utilities to transform data from a non-oscar format 

to austral and vice versa because there are different tools they that generate 

their compliance related artifacts in their own format and to really make 

oscar usable for them we need a set of utilities and tools which will help them 

convert their set of a set of content into oscar format and vice versa and 

this sdk actually forms the basis of a
security and compliance center in IBM Cloud where we need to convert data from various assessment tool formats such as Stanium and OpenShift compliance operators which generate data in their own native format into OSCAL and so on and these tasks and transformers also provide capabilities to convert other kind of content in different formats such as Excel sheets, XML content, and so on into OSCAL format. A couple of links are available here where you can go and look at how these transformations can be done. We have a very detailed documentation of giving step-by-step instructions on how to use the transformation, and we also have a set of pre-canned demos which we have built.
demonstrating different capabilities of
tresel so these are available

publicly and one can go and look at each
of the demos and see how different
capabilities of pressure can be used
next slide please.

So

as i mentioned right we need support for
uh authoring different kinds of content

uh because editing oscillation is not
very

very easy we so Trestle provides this
content authoring capability where we
can convert content both from OSCAL
format into

into markdown and back. And also Trestle
supports

content authoring for even non-oscars
content for example
a system architecture design decisions and all which can be captured in a markdown format but we need to enforce some of some templates over there and uh so that the content is uh authored in a in a specified format and trestle can enforce those content on that and we have a demo of uh the link is given here uh on how to use the content authoring capability of festival over there and we we will also show in in the next few slides how we are using uh tesla to actually on uh author rascal content next slide please so just to add before we move forward so we have the set of pocs with various personas and we
immediately realized that depending on the persona the set of uh skills right where we're very uh um on a wide spectrum right the uh the farther we go from the you know policy as code and from for from code developers uh the uh ability to to work with uh raw json you know decreased and people are more familiar with uh you know text or spreadsheets and so on so the what you see next is a reflection of our you know a few uh pocs with the individuals right that we worked with it can be that uh in you know your
particular context different type of skills are required so what we present is by no means a limitation of what trestle can do it is just a reflection of our pocs and if additional type of uh interfaces are needed uh to for those uh personas to to author that that can threshold can be expanded to support those yeah the the last piece here is the agile logic especially the SSP part and since SSP are very complex documents and they can be multiple editors who provide different kinds of contents to the ssps what we support is we convert the ssp document to a markdown file per control where different editors can go and
provide their content in terms of control responses and all, and then these markdown contents can be assembled back into a single ssp and this since we have this command line interface it can be used with the ci cd pipelines or get workflows and also that whenever some content changes the city pipeline execute the required commands to convert this content back into austral validate it so that ensure that it is following the proper schema and all and then this final document ssp document will show we also create a word document out of from this combined ssp document which can then be given to auditors to verify whether the control implementation for the overall system is correct or not
so next we are going to show for each of the as i mentioned for each

of the different users whether it is control owners such as regulators and

OSCAL profile owners services owner and all how they can use stressel to create the content in different formats and then convert it

back to oscar the next slide please so here what we are showing is how a catalog can be created in the markdown

and then it can be converted into oscillator so what we provide here is a capability where an auscal catalog can

be it can be broken into separate markdown files one per control users can go and edit the content over
here in terms of they can specify the control statement

they can specify the guidance even the parameters and and the default

default values and all can be specified specific properties can also be added so there

are two two kinds of content here one is the standard markdown content that we

are seeing where we specify in text format and to specify the structured

content like parameters and properties we use the yaml um format that we put it

as a yaml header at top of the markdown content because it's much more easier to

capture this content in

uh

okay so yes the structured content is more easy to capture in yammer format than
and what we do is once all this content is available in the markdown file we can easily convert that using the trestle content authoring command line interface into the oscar js and so if you see on this right hand side the text is a bit small sorry for that but what we see is the parameters we create a control uh SC-7 here we set the parameters and the values over here we set the sort id property in the properties of that control the actual text of the control statement goes into parts and statements and subparts of that so we create the proper hierarchical uh format of the statement so if a statement has subparts like abcd and they are further subdivided into one two three four and all all that
hierarchical hierarchical structure

would be maintained in the oscar js and
when we convert this markdown into

hospital the guidance and all would be
added as additional parts over there and

when we when we create the oscar all the
markdown files

for all the controls would get merged we
also have a proper directory structure

to capture the group information so
each group would have a directory and

then uh inside that we'll have one
markdown file for control and based on

that directory organization when we run
the cli command this single oscal json

file which is compliant with the
oscillator schema will be created

the next slide please
similarly we have the capability to edit

the profiles so here what we do is in
addition to just the content that needs to be edited in the profile we also provide the content from the catalog so for example the control statement which is coming is coming from the catalog from the catalog this would help the profile editor to to look at the content so that when the editor needs to set the values of the different parameters for this control or specify additional guidance they can look at the control statement so that the appropriate responses or the guidance can be added over there rather than opening a separate window and looking at the control statement over there all the content is available easily available over here for reference and we can cut we can convert this uh off mark down to
oscar json format so here uh we add all the controls in the list of control for this specific control the parameter settings would go into the modify set parameters uh part of the profile the additional contents the editable content which is added here in terms of additional guidance or additional parts which would go into the control that is added as an alter statement over here and the respective content would be added with the right ids and properties and all and the content would be added over here next slide please now after this we have the component definition and there are two kinds of information as anchor mentioned earlier in the component definition right one is the mapping from the technical rules to
the catalog controls now this kind of

00:15:48,079 --> 00:15:53,040
information is most easily captured in a
spreadsheet rather than specifying that

00:15:53,040 --> 00:15:57,279
in the markdown style it can be done in
magdon also but given that there will be

00:15:57,279 --> 00:16:01,360
lots of rules that are getting mapped to
different controls

00:16:01,360 --> 00:16:05,519
we found at least in our experience that
users are more comfortable using the

00:16:05,519 --> 00:16:09,040
spreadsheet and they also need to
specify the corresponding parameters and

00:16:09,040 --> 00:16:16,240
value over here so what we do is we take
the this information of mapping from

00:16:16,240 --> 00:16:20,399
rules to different controls for our
component in the spreadsheet format and

00:16:20,399 --> 00:16:24,000
we convert that into an OSCAL component
definition

00:16:24,000 --> 00:16:28,160
and again the text is not very visible
over here but what we have done is we

00:16:28,160 --> 00:16:33,199
have created a component inside this
component definition for this vpc

00:16:33,199 --> 00:16:36,320
component that is shown here in the spreadsheet.

00:16:36,320 --> 00:16:41,279
For that component there are three rules which are mapping to SC-7 so what we do

00:16:41,279 --> 00:16:46,880
is in the implemented requirements we have SC-7 control id under that we are

00:16:46,880 --> 00:16:51,600
we are capturing this rules in the form of properties as Anca described.

00:16:51,600 --> 00:16:55,120
Currently in OSCAL we don't have the support of

00:16:55,120 --> 00:17:00,000
capturing these technical rules and the respected respective parameters and

00:17:00,000 --> 00:17:04,880
their values so we are using properties over here to attach these rules with

00:17:04,880 --> 00:17:09,679
this control implementation inside the component definition but once we have

00:17:09,679 --> 00:17:14,079
these capabilities to represent rules and rule parameters properly component

00:17:14,079 --> 00:17:18,559
definition we will switch to the new
format once that is available

232
00:17:18,559 --> 00:17:22,559
but using the properties we are able to
associate these technical rules and

233
00:17:22,559 --> 00:17:27,679
their parameters with the the
implemented controls like sc-7 which

234
00:17:27,679 --> 00:17:31,679
which we are showing over here
in the component definition so this is

235
00:17:31,679 --> 00:17:35,200
one part of the information where we are
capturing the structured information in

236
00:17:35,200 --> 00:17:40,160
the spreadsheet and next please
the other set of information

237
00:17:40,160 --> 00:17:44,559
is the control responses in terms of the
text how a

238
00:17:44,559 --> 00:17:49,520
given service implements the given
control this information is more uh

239
00:17:49,520 --> 00:17:54,160
easily captured in the markdown format
as we have done for catalogs and profile

240
00:17:54,160 --> 00:17:58,559
so for this piece of information we
capture the control implementation

241
00:17:58,559 --> 00:18:02,160
information in this background file as
we are as we are seeing over here

242
00:18:02,160 --> 00:18:07,600
implementation a b and c where
the component definition owner can go

243
00:18:07,600 --> 00:18:12,559
and provide the control responses
we show the content from the catalogs

244
00:18:12,559 --> 00:18:17,200
and profiles and also the set of rules
that we have mapped from the spreadsheet

245
00:18:17,200 --> 00:18:21,200
into the component differentiate the top
so that the

246
00:18:21,200 --> 00:18:25,520
user who is editing this markdown for
markdown file has all the content

247
00:18:25,520 --> 00:18:30,160
available at one place to write the
appropriate control responses and then

248
00:18:30,160 --> 00:18:34,640
this gets converted into the austral
component definition format so

249
00:18:34,640 --> 00:18:38,559
we add these control responses to the
existing component definition that was

250
00:18:38,559 --> 00:18:43,200
created in the last step from
spreadsheet and it would go into the

251
00:18:43,200 --> 00:18:47,600
description of these various statement
implementations as we are seeing on the.

00:18:47,600 --> 00:18:51,840
right hand side left
we have statement a and and there we in

00:18:51,840 --> 00:18:55,280
the description we have put this control
responses how this specific component

00:18:55,280 --> 00:19:00,000
implements this control.
Next slide please

00:19:00,000 --> 00:19:06,320
We also have the validation tools that
specify the mapping of technical rules

00:19:06,320 --> 00:19:11,600
to the checks that are implemented in
policy as code so again that information

00:19:11,600 --> 00:19:14,960
is captured in the spreadsheet format
and here we are showing the

00:19:14,960 --> 00:19:17,840
rules and the corresponding check ids
here

00:19:17,840 --> 00:19:21,200
in addition to the control to which this
rule belongs

00:19:21,200 --> 00:19:24,400
and
we can convert this excel sheet

00:19:24,400 --> 00:19:27,840
to an OSCAL
component definition for this validation

00:19:27.840 --> 00:19:31.760
component so here if we see there is a
validation component for tool chain and

00:19:31.760 --> 00:19:36.480
for this sc-7 control we have now two
properties one is the rule id that that

00:19:36.480 --> 00:19:40.559
is captured and we have the second
property which captures the check

00:19:40.559 --> 00:19:44.000
associated with that rule so both the
information is captured through

00:19:44.000 --> 00:19:48.880
properties currently and as we have
support for specifying checks and rules

00:19:48.880 --> 00:19:54.160
in component definition and then we'll
we will use those new constructs

00:19:54.160 --> 00:19:59.039
once that become available
and then next slide please so finally

00:19:59.039 --> 00:20:04.720
once we have all this information
just just one comment uh so now as you

00:20:04.720 --> 00:20:10.080
as vikas mentioned right in vpc we have
rc7 three rules and as we associate

00:20:10.080 --> 00:20:14.880
those three rules with the three
corresponding checks um under you know

00:20:14,880 --> 00:20:20,400
the the properties we use
and and their version or their you know

00:20:20,400 --> 00:20:24,000
the
relationship between the parameters now

00:20:24,000 --> 00:20:29,600
within one property entry we we
um

00:20:29,600 --> 00:20:33,840
with the properties we have now declared
multiple properties that we need to link

00:20:33,840 --> 00:20:37,440
together the right rule is the right
check the right version is the right

00:20:37,440 --> 00:20:41,760
version for hulu is rice version for
check so we repurpose here

00:20:41,760 --> 00:20:46,720
the remarks uh entry uh to provide the
key

00:20:46,720 --> 00:20:50,559
of the grouping right so uh because a
lot of people have asked in the past how

00:20:50,559 --> 00:20:55,919
how we do that and uh i know oscar is
working in the

00:20:55,919 --> 00:21:01,200
next release on providing support for
grouping natively so then we'll be able to use remarks uh properly and move to the grouping when that would be available

Thanks Anca for specifying that yeah so so yeah we are currently using remarks field for grouping related properties together and once we have the proper construct we will be using that now once we have all this content available the final step is to generate the ssp

now there are two parts here right one is we have to assemble all the responses that were added to the component destinations into the SSP plus there are some additional content which which are specified in SSP one is
for the overall system the the the system

owner may want to specify how the overall system implements are given

control so that information can be added to this markdown file in the ssp repo

for this control and also we can capture some of the properties like control

origination what is the implementation status what are the roles and all

in the ml header and all this gets converted into the json ssp so so as we

are seeing on the right hand side we have

multiple components one per component definition that is coming from the

component definition OSCAL file which gets added over here and the

corresponding control responses are provided

plus we we have this system which is the
overall system for which the control responses are captured in this markdown file and that also gets added as a separate component for the overall system where this control response is captured over there so in this way this ssp is almost automatically generated in the sense that all the content is already available at different places and we just need to assemble that into this SSP there is some additional content that needs to be provided which is not captured as part of the markdown for example the system characteristics and all now these inventory information all this information can be injected into this oscar json
either if that content is available in

markdown or JSON file in some other
places we have the capability to add

this content into this final uh oscar
ssp that that that gets generated

and once we have this final OSCAL json
what we do is we have capability to

convert it
this into a word document so that

we have all the responses with all the
uh with all the architecture

system characteristics and everything uh
captured in the word document which can

then be said to auditors so we have this
capability of converting oscar.json to

what document also
already implemented yes

going to the next slide
so

just one more comment based on the
questions so um

00:23:47,919 --> 00:23:52,400
we are not forcing the people to use
markdown both the markdown and the JSON

00:23:52,400 --> 00:23:57,440
are available for uh for editing right
uh the the markdown is just the support

00:23:57,440 --> 00:24:02,240
for people that don't feel comfortable
with raw json uh but if there are

00:24:02,240 --> 00:24:06,480
developers in the component in the
service vendors that want to declare

00:24:06,480 --> 00:24:09,440
directly in the json they can go and and
uh

00:24:09,440 --> 00:24:14,000
directly populate the the json
yeah thanks thank you for adding so what

00:24:14,000 --> 00:24:17,840
we are showing here is if you see these
two arrows right whatever content we

00:24:17,840 --> 00:24:21,840
edit in the markdown format we can
convert back to oscar json format and if

00:24:21,840 --> 00:24:25,279
somebody is directly editing the json
for example this control responses are

00:24:25,279 --> 00:24:29,520
directly provided in the json we can
also convert that back into the markdown

00:24:29,520 --> 00:24:33,520
format so that users who are
more uh

00:24:33,520 --> 00:24:37,360
familiar with markdown and they find
json hard to edit they can go and update

00:24:37,360 --> 00:24:41,360
in the markdown and then we can convert
vice versa from one format to the other

00:24:41,360 --> 00:24:44,480
in our experience what we have
felt is

00:24:44,480 --> 00:24:49,039
people like uh cso team and control
owners and all they're more they're not

00:24:49,039 --> 00:24:53,760
very comfortable with directly editing
json so they want a format which is more

00:24:53,760 --> 00:24:56,799
human friendly and that's why we have
markdown but

00:24:56,799 --> 00:25:00,880
that doesn't stop us to stop anyone from
directly editing the json that people

00:25:00,880 --> 00:25:04,799
that functionality is
always available yeah

00:25:04,799 --> 00:25:09,679
now this uh slide basically shows the
organization of wages reports as we have shown we have a catalog repo where control owners regulators and cso and all can go and edit the control so we can have some existing catalog like this 800-53 plus organizations can define their own catalogs also with additional controls that can be created in this catalog repo either directly in json or in the markdown format and converted back into one format and the other then we have a separate trip of profile repo where additional guidance and parameters and all can be set for these controls and similarly we have the guidance repo where the how the
how this control should be implemented

by various services and all are captured
what are the evidence to be provided so

that can be in the form of a separate
guidance report it can be part of the

profile also we have kept these two
separate

but this guidance support is actually an
oscar profile object only where this

additional content gets added and this
guidance can be published in

some website and all where users can go
and look at it so we also have a

capability to convert that into excel
and uh

html format over here then we have the
component definition repo where

content can be edited in spreadsheet and
markdown format and converted back to

the austral and then finally we have the
ssp repo where

we finally generate the SSP in the word document format what we have done is we have linked all these report together in the sense that suppose a catalog repo updates some some catalog or controls in this repo what can be done is

once the catalog gets updated we notify the downstream repo which are dependent on that catalog so in this case we will notify the profile repo uh with what has changed in the catalogs and we directly open a pr over here with the appropriate changes so that the profile owner can look at the changes in the catalogs and then they can merge that changes or include the changes in the profile repo similarly if
the profile owner updates the profile

00:27:16,559 --> 00:27:21,039
we notify all the downstream repo that
is the guidance component definition and

00:27:21,039 --> 00:27:27,120
ssp repo about this change using the git
pr mechanism so that so that whenever an

00:27:27,120 --> 00:27:31,440
upstream content
changes the downstream content which are

00:27:31,440 --> 00:27:34,960
reports which are dependent on that
content are not notified they can look

00:27:34,960 --> 00:27:38,399
at that content and and if they find
that reasonable then that content would

00:27:38,399 --> 00:27:44,320
get merged in that so this way
we have set up this end to end pipeline

00:27:44,320 --> 00:27:49,279
so that whenever any content changes
using the ci cd mechanism the content

00:27:49,279 --> 00:27:53,279
would automatically be converted to
auscal the pipelines would be triggered

00:27:53,279 --> 00:27:56,960
so that the next set of reports are
updated over

00:27:56,960 --> 00:28:01,200
are notified and whenever those reports
get updated the lower reports get

00:28:01,200 --> 00:28:04,320
updated
and finally the SSP can be automatically

00:28:04,320 --> 00:28:09,760
built using this whole uh end-to-end
pipeline across the ripples

00:28:10,240 --> 00:28:14,399
okay thanks thank you

00:28:14,399 --> 00:28:20,159
yes so uh the the spirit here is
continuous compliance right so uh the uh

00:28:20,159 --> 00:28:24,399
assumption is that automation is in
place in order to um

00:28:24,399 --> 00:28:29,679
to collect the evidence to do the
assessments and um we want to maintain

00:28:29,679 --> 00:28:33,679
the documentation that is associated
with that constantly up to date so that

00:28:33,679 --> 00:28:37,360
that's the the reason for the
implementation is pipeline and the

00:28:37,360 --> 00:28:41,360
uh the way that because show the
structure now um

00:28:41,360 --> 00:28:45,840
what we've seen so far these are um
libraries right we don't have any
environment we may have reference architectures that are uh uh described

like in a terraform file format or other

other type of ci cd artifacts and we can you know um

associate an ssp with with that um type of reference architecture

artifact but now if we want to move from this

library context to an actual environment context

what we are doing is we are looking at the

grc tooling right as an uh hierarchical

architecture and i'll have a complex live but i'll help you to traverse it

so we populate we expect that the artifacts the compliances code that we
have seen are populated for all the controls in a particular regulation

independent of whether they are going to be uh

automated or not for collecting the posture so this information would be consumed from the repository by grc in the case of ibm we have open pages and we developed an oscar interface for open pages

um and now the um

the this uh depending on the scope of the various tools the appropriate information will be provided so in the context of the ibm security and compliance center because we are a generic orchestrator all the
technical controls details catalog
profile uh component definition will be populated there if you are talking about an um an orchestrator like the openshift
uh acm or acs we would assume that only the controls relevant to the uh that particular context will be populated now um in order to provide this information in a standard way we have developed an um exchange protocol to allow the exchange of information for that you know library content that we have just seen but also uh to allow uh alignment to a standard for the results for the interlock with the various tools that provide validation by policy validation points so if you are looking at this generic orchestrator that is working with red hat tools three ibm uh
specialized orchestrator is tenu
cavionics they all have their native

formats to to provide the results so we
use the exchange protocol to um

align everything with with OSCAL the the
lower we go into this into the stack the

more specialized we become and
eventually we hit the point where we

have just custom interfaces but as we
are at the uh generic uh orchestrator

level we found it very useful to to be
able to aggregate um across the uh the

videos tools and provide posture in a
unified way so with that i will stop

sharing and I let
Lou take over to present the details of

the exchange protocol API