1
00:00:11,000 --> 00:00:21,000
At Rise8 software development right now, we're working with the DOD and the VA.

2
00:00:19,000 --> 00:00:29,000
We've also worked with other government organizations, mainly developing paths to production, shipping software to production and actually

3
00:00:29,000 --> 00:00:39,000
notably recently worked with the VA to establish their first continuous ATO. So we have a deep understanding of the challenges

4
00:00:39,000 --> 00:00:49,000
associated with getting new capabilities to production. Notably the process and really executing RMF in general.

5
00:00:48,000 --> 00:00:59,000
So we've been developing Tracer as a continuous RMF platform internally to rise in order to accelerate this ATO process.

6
00:00:59,000 --> 00:01:09,000
and really enables the delivery of new software capabilities faster than currently possible. And really, you know, the reason why we're here is that we see

7
00:01:09,000 --> 00:01:19,000
OSCAL as a key enabling technology or function sort of behind the scenes to the success of Tracer and in really helping Tracer do what

8
00:01:19,000 --> 00:01:28,000
we wanted to be able to do. So over the next few minutes, we're going to dive into how a tracer leverages OSCAL. So let's

9
00:01:29,000 --> 00:01:38,000
dive into it. So I know most people here are very familiar with OSCAL. We don't have to talk about I talk about it too.

10
00:01:38,000 --> 00:01:48,000
much. So let's focus here on the implications, like why is OSCAL

important within the context of Tracer? So currently

11
00:01:48,000 --> 00:01:58,000
, the process relies heavily on basically Microsoft Office documents
like Word Excel. There's a ton

12
00:01:58,000 --> 00:02:08,000
of manual review and upkeep of these documents. These documents like
transparency, search ability. They're difficult to update

13
00:02:08,000 --> 00:02:17,000
, really. They're once they're finalized, they're set in stone,
they're version management issues. And really, all of this drives

14
00:02:17,000 --> 00:02:27,000
what at times could be a 12 to 18 month authorization process. And
because of that, you know, countless company hours

15
00:02:27,000 --> 00:02:37,000
spent managing this process and can cost upwards of millions of
dollars in order for, you know, an authorization to be put together
and push it across the finish line.

16
00:02:37,000 --> 00:02:47,000
So in then because of that, because it's so difficult, a lot of times
these these authorizations just aren't updated

17
00:02:47,000 --> 00:02:57,000
. And because the authorizations can't be updated, the software can't
be updated. And you see these sort of huge Leviathan pushes

18
00:02:58,000 --> 00:03:07,000
to get the initial software deployed, the authorization done, and then
it's just kind of left there. And so you can point to countless
examples across the

19
00:03:07,000 --> 00:03:17,000
government of of of antiquated software that has real mission impacts
no matter the department that you're in. And

20

```
00:03:17,000 --> 00:03:26,000
in a lot of this can be traced back to really the bureaucracy around
the authorization process. So what we want to do is we see

21
00:03:27,000 --> 00:03:36,000
OSCAL as a key function for eliminating a lot of that oil and opacity
within that process. We want to create a system that puts compliance
and security

22
00:03:35,000 --> 00:03:45,000
data at the center, not documents in order to create a transparent
process, you know, remove the barriers to updating authorizations

23
00:03:46,000 --> 00:03:56,000
and pave the way for more frequent delivery of software. So, so right
at the bottom of this slide, you'll see a small snippet of the asphalt
control

24
00:03:55,000 --> 00:04:05,000
format. This is what we're pulling in directly from the next repo to
populate tracer. And of course, we're talking about 800-53 controls.

25
00:04:06,000 --> 00:04:16,000
So let's dive a little bit deeper into what that looks like for
tracers specifically.

26
00:04:15,000 --> 00:04:25,000
So right, how Tracer uses OSCAL. So the the app itself ingests the
entire eight hundred fifty three control catalog

27
00:04:25,000 --> 00:04:35,000
from the next repo and then is able to manage and display them in the
user interface. We will take a look at some screenshots from Tracer

28
00:04:36,000 --> 00:04:46,000
will also walk through a quick demo to sort of see it in action. But
this is a bit of a little primer, so we know it's going on behind the
scenes

29
00:04:45,000 --> 00:04:55,000
. So we pull in the control catalog and and then Tracer does a lot of
```

the heavy lifting when it comes to managing

30
00:04:56,000 --> 00:05:05,000
the control catalog, system creation, tailoring. And then users go in
and can do all this sort of implementation assessment documentation

31
00:05:05,000 --> 00:05:15,000
and then ultimately generate artifacts or reports that can either be
manually extracted in the form of, say, like a CSV

32
00:05:15,000 --> 00:05:25,000
or in the actual format in X somehow or JSON format that can then be
shared with other programs that are able to

33
00:05:25,000 --> 00:05:34,000
consume it. So, you know, we've heard or I'm sure the community is
aware of Exacta, their ability to consume off

34
00:05:34,000 --> 00:05:43,000
OSCAdata on the Fed ramp office is very forward leaning in their
ability to consume OSCAL data

35
00:05:44,000 --> 00:05:53,000
. So just as a little aside, we want Tracer to be fully self-contained
for the purposes of executing autonomous.

36
00:05:53,000 --> 00:06:03,000
But we also know that, you know, the the RMF process is sprawling.
There are many stakeholders, many different systems

37
00:06:03,000 --> 00:06:12,000
that are involved, so we know it needs to be able to be. The data
curated within tracer needs to be easily shareable, removable with
external services and

38
00:06:12,000 --> 00:06:22,000
also is perfect for doing that. So really, all of the the not only the
controls, but then the body of evidence that's managed within tracer.
All of that

39

00:06:22,000 --> 00:06:32,000
is kind of is packaged up and behind the scenes is managed in the ask out format so that it can be easily. It can easily be tracer

40
00:06:32,000 --> 00:06:41,000
and move into external services. So let's dive into our first control. I'm sure everybody's favorite control is

41
00:06:41,000 --> 00:06:51,000
one. So here we see the really a big portion, a small portion of the control

42
00:06:51,000 --> 00:07:01,000
in our format for AC once again pulled straight from the the in this repository. And then on the right, we have a small screen shot of the tracer

43
00:07:01,000 --> 00:07:11,000
user interface. We're going to zoom in on the tracer UI in the next slide. I'm sure it's impossible to read right now, but so we take that controlling

44
00:07:11,000 --> 00:07:21,000
twitch directly from the this repo in the scout format that is then displayed in the tracer UI, where users can actually

45
00:07:21,000 --> 00:07:31,000
can't access it. You know, then then below you'll see the implementation, this sort of documentation phase where, you know, really, depending you know,

46
00:07:31,000 --> 00:07:40,000
engineers, developers, the people responsible for implementing the control, they input their evidence here. You have your then third party

47
00:07:40,000 --> 00:07:50,000
assessor who can come in and do the assessment, and all of that data is is is collected in one place and

48
00:07:51,000 --> 00:08:00,000

and again, it's all saved in that OSCAL format for the next step. So let's zoom in and see what that looks like. It's a little bit clear of

49
00:08:00,000 --> 00:08:10,000
a picture here. On top you have the actual control language, you know, your assessment objectives, you know, the control discussion

50
00:08:10,000 --> 00:08:20,000
, you know, ultimately, you have your implementation parameters here as well. And then below this is where you'll see the evidence added by developers

51
00:08:19,000 --> 00:08:29,000
assessed by assessors and emerged into this overall body of evidence that will ultimately go into the system security plan.

52
00:08:29,000 --> 00:08:39,000
So moving on, we then see again, this is just like a the control language is then merged

53
00:08:39,000 --> 00:08:49,000
with that body of evidence in the scout format that will then go through to shape the Boscalid System Security Plan

54
00:08:49,000 --> 00:08:58,000
. So what tracer does? It has another interesting facet where we we look at controls really from

55
00:08:58,000 --> 00:09:08,000
two directions. One is from the system perspective in the other one is from the component perspective. So let's let's really just talk about the system perspective

56
00:09:08,000 --> 00:09:18,000
first. Any given control can be broken out into policy, infrastructure, platform or application

57
00:09:17,000 --> 00:09:28,000
controls. So, you know, choose your control. It can be labeled as one or really any combination of the above. And what that means is that when we establish

58
00:09:27,000 --> 00:09:38,000
these components within a system, if a component is, say, a policy
component, that component is, then response

59
00:09:38,000 --> 00:09:48,000
able for all controls that are labeled policy. So going back to this
example, you know you have your control here

60
00:09:48,000 --> 00:09:58,000
. Let's just say it must be implemented across all of the layers of
the system, stack policy, infrastructure, platform and application

61
00:09:58,000 --> 00:10:07,000
that we looking at it. From a system perspective, we can do an overall
kind of holistic system control

62
00:10:08,000 --> 00:10:17,000
assessment, but then you can also dive into how this control was
implemented at each of these given layers. This is supported in the

63
00:10:17,000 --> 00:10:21,000
the system security plan format

64
00:10:21,000 --> 00:10:31,000
, so you get a holistic, system centric perspective, but you can still
dive in to see how each control affects

65
00:10:31,000 --> 00:10:41,000
or is affected by an individual component. And the converse is true as
well. When you you can dive into an examination

66
00:10:41,000 --> 00:10:51,000
of a given component. We know exactly which controls that component is
responsible for. So you can you can almost

67
00:10:51,000 --> 00:11:01,000
take that component out of context and do an individual component
assessment versus really having to assess the entire stack at any
given time. And that

```
68
00:11:01,000 --> 00:11:11,000
has a couple of different implications that we'll talk about a little
bit later on and sort of like where we see Tracer going in the future.
But at this point, this is how

69
00:11:10,000 --> 00:11:15,000
we're we are developing a very clear

70
00:11:15,000 --> 00:11:25,000
breakdown and a very transparent method for sharing really system
control implementation evidence through a modern

71
00:11:25,000 --> 00:11:28,000
through a modern

72
00:11:28,000 --> 00:11:32,000
software system

73
00:11:32,000 --> 00:11:42,000
. A lot of what's coming online right now is cloud based. And, you
know, BWC, GCP, Azure

74
00:11:42,000 --> 00:11:52,000
, all those services, they handle so much of the the control baseline.
And then you can you can go on down

75
00:11:52,000 --> 00:12:02,000
into with your your platform services. A lot of these is commercially
available services, really, they are components ized. And so

76
00:12:02,000 --> 00:12:12,000
the way that controls are answered really do. You can't really answer
it completely from from

77
00:12:12,000 --> 00:12:22,000
one perspective, from the system perspective, you really do need the
sort of stratified component structure in order to really accurately
and transparent
```

78
00:12:22,000 --> 00:12:32,000
answer these controls. So let's now take a look at tracer in action. Um, I'm just going to sort

79
00:12:33,000 --> 00:12:42,000
of let this demo play for a few minutes. I'm just going to sort of prime you all just so you know what you're looking at. We're going to be landing here on the systems

80
00:12:42,000 --> 00:12:51,000
page where you can manage multiple systems. It's not just one system in my system, it's really like authorization so you can manage

81
00:12:51,000 --> 00:13:01,000
multiple authorizations. And here we're sort of jumping right into an established system. But in order to establish

82
00:13:01,000 --> 00:13:11,000
a new system, you select your baseline or your your framework of choice. Right now we support eight hundred and

83
00:13:10,000 --> 00:13:20,000
fifty three, four and five. We also have a API overlay and we're soon going to be able to

84
00:13:20,000 --> 00:13:30,000
support the federal framework and baselines as well. And all of these controls are automatically imported via ask from again the

85
00:13:30,000 --> 00:13:40,000
next repositories. So behind the scenes, when you're establishing a system, you select your framework, you select your your categorization, you know, low, moderate or high

86
00:13:40,000 --> 00:13:49,000
. And Tracer just goes and grabs all of those controls and populates them in your system for you, of course. Then you move on to the tailoring where you

87
00:13:49,000 --> 00:13:59,000

can customize your control baseline to your specific system's needs, and then you go through the rest of the process of implementation assessment

88
00:13:59,000 --> 00:14:09,000
and beyond. So hopefully you'll be able to hear the audio from this. Let me know if you can't, and I'll just sort of do a talk track

89
00:14:09,000 --> 00:14:19,000
over it, and I think the audio will be coming through, so you might need to to speak.

90
00:14:20,000 --> 00:14:29,000
Let's see. What can you let me know if you've tried or not. You're looking at the system state because either the lighthouse system is our system

91
00:14:29,000 --> 00:14:39,000
, all the consoles. Sorry, there are several people that have died in particular. Now you can see the entire console list of the system in

92
00:14:38,000 --> 00:14:44,000
over on the left to see how the controls are broken down by all Typekit streaming here

93
00:14:45,000 --> 00:14:54,000
. Resistance, breathing because those are automatically imported in this suppository and it'll make your selection immediately. Once we get the controls

94
00:14:54,000 --> 00:15:04,000
through the system, you can use one organization infrastructure platform applications where any company really uses just

95
00:15:04,000 --> 00:15:07,000
no structures of older people.

96
00:15:08,000 --> 00:15:18,000
establishing this comparison structure. Let's use this system here and all the consoles above it. You're also welcome here. The consoles

97
00:15:16,000 --> 00:15:26,000
, as we can see here at the bottom of the stack, the application here is responsible for one hundred and forty eight hundred and seventy four

98
00:15:26,000 --> 00:15:33,000
. This means that I'm working. The engine already goes activist consoles and

99
00:15:33,000 --> 00:15:43,000
another system is established where consoles let's onboard a new application. You can see all

100
00:15:42,000 --> 00:15:52,000
the individual components already established here. The system. Let's I'll fill out all the required information and we will establish inheritance

101
00:15:58,000 --> 00:16:07,000
selection already established within the data warehouse system and as a you

102
00:16:07,000 --> 00:16:11,000
, the console and you components

103
00:16:12,000 --> 00:16:20,000
once done accidentally overall, because obviously it's already been imperative

104
00:16:25,000 --> 00:16:33,000
. Let's get into this new components you'll be able to do right away with the series. Consoles aren't here because of

105
00:16:33,000 --> 00:16:43,000
zero inheritance tax. The check sales and the German agency with five seats in Congress looks like they can talk

106
00:16:43,000 --> 00:16:53,000
. Developers will come here. It's authentic implementation details. So as assessors, you're checking to make sure everything looks as

realtors can talk

107
00:16:51,000 --> 00:16:54,000
in text snippets

108
00:16:55,000 --> 00:17:04,000
as part of their documentation. In this example is the tax records, because there are also spaces for notes in an activity law that developers

109
00:17:03,000 --> 00:17:09,000
and assessors back and forth during the implementation process. We basically have

110
00:17:10,000 --> 00:17:20,000
an automatically generated console report for this new app that not only information that we just added, but also we consult with pixels and documentation pre

111
00:17:19,000 --> 00:17:28,000
so quick little inject. Here we're going to take a look at what a couple of these formats could look like. Right now, we're looking

112
00:17:29,000 --> 00:17:38,000
at the OSCAL XML, of course, as well as csv. So in just a minute, I'll put through and show you examples of what that could be.

113
00:17:39,000 --> 00:17:48,000
This allows us to realistically assess with tax rates and also what to system. You can check back in our simplified see how

114
00:17:49,000 --> 00:17:59,000
it looks in the system perspective because here we responsible for business, all their individual statuses. See here that we apply for a

115
00:17:59,000 --> 00:18:08,000
new application because the markets, along with the other application in this system, the overall implementation of this or in the system is

116
00:18:08,000 --> 00:18:18,000

. Lastly, what is your system like this when your vulnerability
exists? These results are displayed on this. Some

117
00:18:18,000 --> 00:18:28,000
bullet. You have to go to another product to access the latest there.
Bring it all together used to onboard new capabilities. Visualize

118
00:18:26,000 --> 00:18:31,000
your data in its lifecycle. Provides information on

119
00:18:32,000 --> 00:18:41,000
. All right. So there's a there's a lot that goes beyond what is
explicitly required for a traditional system

120
00:18:41,000 --> 00:18:51,000
security plan. As you can see, there is activity logs in the back and
forth between developers and assessors as

121
00:18:51,000 --> 00:19:01,000
a control is going through its initial implementation, or even as the
control evolves over time, as the system or that component evolves
over time,

122
00:19:00,000 --> 00:19:10,000
So the traditional SSP formats, it's really only a subset of the data
that that tracer can

123
00:19:11,000 --> 00:19:21,000
produce. And curators of that being said, of course, we know the SSP
is the industry standard. So the the additional

124
00:19:21,000 --> 00:19:31,000
data that is managed within the application can either be, of course,
just live with tracer or managed in like a separate artifact

125
00:19:31,000 --> 00:19:41,000
. But really, what we're trying to get at is is producing this
platform that goes beyond the initial implementation

126
00:19:41,000 --> 00:19:51,000

and assessment of a given system and moving into a continuous compliance and continuous RMF type paradigm. And so

127
00:19:51,000 --> 00:20:01,000
we know that we're going to have to move beyond really the SSP format, which is what I mentioned before, is is focusing on data, not documents and not

128
00:20:01,000 --> 00:20:11,000
not trashing that space, but really not again, not focusing on SSP being the finish line. It's

129
00:20:10,000 --> 00:20:18,000
really just a a checkpoint along the way. You're right

130
00:20:18,000 --> 00:20:27,000
. OK. So this is a little screenshot of the XML SSP exports that came

131
00:20:28,000 --> 00:20:38,000
out of tracer on this. And again, I think I should have mentioned this upfront. All of this is mocked. None of this is real data on a lot of this. This is all

132
00:20:37,000 --> 00:20:47,000
just for demo purposes only. So even though we were using the VA like house as an example of the information within

133
00:20:47,000 --> 00:20:57,000
tracer in the demo, that's all notional. Same thing for these exports. This is all notional. So this this should

134
00:20:57,000 --> 00:21:06,000
be consumable by all other systems that can consume Moscow. It conforms to the Pascal framework. However, we know that not everything

135
00:21:06,000 --> 00:21:15,000
is there. So a lot and sometimes it will need to be human readable, especially for, you know,

136
00:21:15,000 --> 00:21:25,000

Ayo's or just authorizing bodies that aren't as up on OSCAL still need
more of your traditional kind of CSV or Excel spreadsheet

137
00:21:25,000 --> 00:21:35,000
output. What we're still doing here is supporting the paradigm that
we've established within tracer say for your given control

138
00:21:35,000 --> 00:21:44,000
. You have the overall compliance status of that control from a system
perspective. You can then see which components

139
00:21:44,000 --> 00:21:54,000
support that control. So electricity component one in component two,
they might have their independent compliance status. There's also a

140
00:21:53,000 --> 00:22:03,000
potential inheritance factor for these different components and then
the the actual evidence or implementation details for those individual

141
00:22:04,000 --> 00:22:14,000
components. So this this is just again a more transparent way of
looking at how control is implemented from a system perspective.

142
00:22:14,000 --> 00:22:23,000
Right. So what are the key benefits here? So using ASCO in Tracer
makes it way easier to establish your system

143
00:22:23,000 --> 00:22:33,000
. It really is just a matter of a few clicks to import the control
baseline from Ribeau in the ask out format

144
00:22:33,000 --> 00:22:43,000
. Then a few more minutes of your control selection or tailoring to
really tailor to your exact needs. Beyond that, modifying existing
packages

145
00:22:43,000 --> 00:22:53,000
can be done really easily because this is all in digital format. So
you can bring in controls, you can remove controls again in just a
couple of clicks. And then

```
146
00:22:52,000 --> 00:23:00,000
once all of the the implementation and assessment is done, you can
automatically generate these speeds

147
00:23:01,000 --> 00:23:10,000
. Also, if you have a machine to machine connection, say with with
events or exacta that can be done automatically any time new

148
00:23:11,000 --> 00:23:20,000
data is introduced into the system, you can see there's an update to
how a component addresses a control. All of that can be shipped
automatically to your

149
00:23:20,000 --> 00:23:30,000
your sort of documentation source of choice. So this makes it so you
don't have to wade through huge documents

150
00:23:31,000 --> 00:23:40,000
or spreadsheets. And again, because it's it's data driven you, it's
really easy to tell what has changed with, you know, within a given
time frame.

151
00:23:40,000 --> 00:23:50,000
or with a given with a given update. So what is next? I think there
was a question

152
00:23:50,000 --> 00:24:00,000
in the chat that this directly is related to is establishing the
concept of free floating components. I'm using

153
00:23:59,000 --> 00:24:03,000
OSCAL that can be imported into systems, you know,

154
00:24:04,000 --> 00:24:13,000
a great example is IWC has their OSCAL SSP. I believe GCP just, you
know, push something out. Same thing

155
00:24:14,000 --> 00:24:23,000
goes for other cloud service providers or any other service, you know,
say, you know, vanilla Kubernetes or keyCode, any other service that
```

156
00:24:23,000 --> 00:24:33,000
you want or, you know, that is commonly used establishing their own like Pascal formatted SSP or

157
00:24:32,000 --> 00:24:42,000
sub ASP that can be automatically imported into Tracer. Not only can we bring in the controls that those components

158
00:24:43,000 --> 00:24:52,000
satisfy, but we can also bring in the implementation details. And really, what this means is that building entire packages could be cut down to minutes

159
00:24:52,000 --> 00:25:02,000
. So really, it's it's kind of like you can do a drag and drop kind of thing, right? Because we've established

160
00:25:02,000 --> 00:25:12,000
this hierarchy within this system. You know, you can you can just sort of select your stack and we can bring in all of the data again via OSCAL

161
00:25:11,000 --> 00:25:21,000
and then cut the package creation, not just establishing the system baseline, but actually creating the package down to just minutes. Additionally,

162
00:25:22,000 --> 00:25:31,000
within the master component is updated. Say you know, eight of us updates their SSP. Those updates could then be automatically rippled into everybody

163
00:25:32,000 --> 00:25:41,000
else's package that is using that service. So what this means is that authorisations can really just focus on the small, idiosyncratic

164
00:25:41,000 --> 00:25:51,000
elements of the system that makes that system unique, not reassessing the entire monolith. Really, the entire

165
00:25:51,000 --> 00:26:01,000
time. So what this means is that new capabilities will make it to users much faster

166
00:26:01,000 --> 00:26:11,000
, both in the initial authorization stage as well as keeping software and systems up to date. It'll open the door to more commercial companies who

167
00:26:11,000 --> 00:26:20,000
right now a lot of the industry just doesn't know how to work with, can't work with the government because it's bureaucratic

168
00:26:20,000 --> 00:26:30,000
boundary is just so high. Removing those barriers will open the door for more for more industry to want to work with the government, giving government customers

169
00:26:30,000 --> 00:26:40,000
more choice, more more competition, better prices and of course, across the board making software more relevant

170
00:26:40,000 --> 00:26:50,000
, more frequent updates. And just like really providing a better, more effective software for all of all of our

171
00:26:49,000 --> 00:26:59,000
, you know, our war fighters, our service providers, everybody in the government. So yeah, that's where we want to go in the future. So that's the end of

172
00:26:59,000 --> 00:27:09,000
our slides, our content we would love to hear on thoughts or questions that people might have from the community. And we've got a few people on the

173
00:27:09,000 --> 00:27:16,000
team here that should be able to answer questions as well. So you have opened it up to the floor