# Title

I'm Charles Wilson, Staff Product Cybersecurity Architect at Torc Robotics.

This presentation will cover an overview of threat model creation.

# Training Path

This diagram shows the overall AVCDL training path.

If you're taking this training, it's assumed that you've already completed the AVCDL overview training,

and additionally, that you've completed the

* requirements taxonomy
* security requirements
* secure design principles
* apply secure design

and

* threat prioritization plan training.

Additionally, you should have completed the threat modeling overview training.

This training will cover threat model creation.

# Introduction

Before getting into threat model creation, let's briefly review where it fits in the overall threat modeling process.

# Threat Modeling Process

Here’s the threat modeling process with its three individual activities,

threat model creation,

threat model analysis,

and threat candidate triage.

Let’s consider the model creation activity.

# Model Creation Activity

As the first activity in the threat modeling process threat model creation is arguably the most important.

The system design is considered by the development and threat modeling SMEs.

This results in the creation of a threat model.

It’s important to consider that the system design is not just a singular document.

It’s anything that can contribute to the creation of the threat model.

This includes things like the results of the attack surface analysis, previous threat models, or any other information that the development SME can provide.

It's important to note here that the development SME or SMEs are not an optional participant in this activity.

This is because it is not the place of the threat modeling SME to guess as to the intentions and designs of the development team.

The threat modeling SME's purpose here is to facilitate the acquisition of information and to eventually create a representation of the system which is usable for the process of threat modeling.

[break]

If we consider the NIST threat modeling lifecycle, we see that we have

* diagramming
* identification
* investigation
* mitigation

and

* validation activities.

[break]

Threat model creation is represented by the diagram portion of this.

That's all we're trying to accomplish here.

We're not looking into any deeper meaning behind what the various entities contained within the threat model are.

We're merely attempting to identify what constitutes the system.

This information will be used in the later analysis and triage activities.

# Idealized DFD

Here's our idealized DFD.

The DFD symbology used has been discussed in the threat modeling overview material.

On the left is the resource that we are trying to ensure is secure.

On the right is the interactor attempting to access the resource.

When we're doing the diagramming, it is useful to show the boundary arcs such that the focus is on the object that we are attempting to secure.

[pause]

Every resource should have a single process that owns it.

This is for various reasons, including avoiding race conditions, and the simplification of controls.

The more processes you have attempting to access a resource, the more complicated both your functional and cybersecurity implementations will be.

[pause]

Rather than using bidirectional arrows for the data flows, all data flows are shown as unidirectional.

This eliminates any confusion as to what we're discussing when talking about a particular data flow.

[pause]

This representation is not without its issues.

If we were just using this as something to reason on in a non-cybersecurity fashion, it might be sufficient.

But let's look at some problems that exist in this particular diagram.

[pause]

The first is that we have these two processes that are both named process.

This causes issues in that we have no way to distinguish which process it is that we're talking about.

Is it the process that's managing the resources or is the one handling the interaction?

A second issue that we have is that in order to make a distinction in communication between the Interaction managing process and the resource managing process, we have to put additional notation on the data flow.

In this case, we can't just say that we have a process-to-process data flow.

We have to say process-to-process and also specify that we've got a request.

We need to have unambiguous names for the data flows, the resources, the processes, and the boundaries we're using, because when issues are created, we want unambiguous places for the development team to look when they're considering the issues.

So, let's look at a slightly updated version of this.

# Preferred DFD

This is roughly the same diagram, but we've given names to the processes which make it very clear what each does.

We have the resource manager and the interaction manager.

Now, the fact that they're called managers is arbitrary. We've simply given them unambiguous names.

Additionally, all of our data flows have been uniquely named in the form, source to destination, so that when we're talking about issues it’s very clear which data flow is being referred to.

Now, you might be wondering what if I have multiple protocols being used going from a source to a destination.

That's when you would have the additional decoration, as seen in the previous diagram, with parentheticals that allow us to show that a data flow is using a particular protocol.

This allows us to disambiguate when using multiple protocols over the same channel.

# Example

That's enough of the theoretical aspects.

Let's work our way through an example that shows various aspects of how the model creation activity is undertaken.

# Block View

Here's the example system that was shown in the threat modeling overview training.

Let's get into how we're going to translate this system into a DFD suitable for threat model.

# Level 1 DFD

This is the level one DFD that was shown in the overview.

And let's look at it in a bit more detail.

All of the processes shown have been directly translated for the block view.

You'll notice that the names are all fairly unambiguous.

We have core service, web server, web browser, console interface, and terminal.

Likewise, the interactor is fairly straightforward in terms of its translation to just administrator.

Now, we don't know that this is an administrator.

They're just an external user, but for the purposes of this example, we'll call them an administrator.

One of the first things of note is the translation of our data stores.

Because we have only one DFD symbol that we use for all resources, there is an adornment on that symbol that shows us at a very high level, whether we're dealing with structured or unstructured data.

Specifically, the file icon versus the database icon that we're using here, which in the block view, was very obvious that there was a distinction, and we're going to put that here.

It's important to note that your DFD creation utility may not be able to put this information there.

And so, an alternative here would be to have some textual notation that you add to the diagram, because free text tends to be allowed in just about every system.

There are two boundaries, and those are the core service boundary and the machine boundary.

The direction of the arc shows that the most important thing in this element is the core service.

This diagram doesn't have a boundary between the resource and the process.

This is a simplification, and the reason for it is that there is implicit trust between the core service, the console interface, and their respective resources (data stores in this case). So, we don't bother putting that there.

As mentioned, having unique process and resource names gives us the ability to have data flows, which are also unique when we're using the convention of source to destination.

We don't have a call out here for the protocol that's being used from the web browser to the web server because only a single protocol is being used between them.

[pause]

We recognize that this can’t be all that's there within the actual system.

We don't just have free floating processes without some kind of operating environment, which provides us the structure that we use in order to make the connections between things a process and resource, or between a process and the outside world.

So, let's expand on this.

# Level 2 DFD

We need to create another DFD that adds in the entities that've been left out.

In this case, we can see that we're interfacing to the filesystem.

So, we're going to add the filesystem process, both for the core services configuration data, as well as the console interface configuration data.

Similarly, the core services data and console interface data are both databases, and so, we're going to add in a database manager.

[pause]

Communication between the core service and console interface, and the core service and the web server is accomplished via interprocess communication.

And so, we're going to add in interprocess communication management.

[pause]

We know that the web server is going to go across the network to get out to the web browser, and so, we're going to add the network manager.

[pause]

Finally, terminal communication is achieved via a serial driver and so we add it in.

That takes care of the processes, but we've now introduced another set of boundaries.

We have operating system boundaries between the web server, core service, and console interface; and the interprocess communications manager and filesystem.

And we have a boundary between the database manager and the core service and console interface.

[pause]

Within this system a single database manager is being used for all of the databases being managed.

Note that this does not tell us anything about whether the databases shown are separate files or merely tables within a single database file.

None of that is relevant at this level.

[pause]

We've dropped the core service boundary as the OS boundary is more significant.

[pause]

And we have a whole new series of data flows, and since we have decent process naming, we have unique data flows between the various entities.

You'll note that there is replication of process entities for both the database manager and the operating system processes.

The reason for doing it this way instead of having one process entity with data flows going off in all directions, is that it’s difficult for a person to reason on the diagram.

From the standpoint of analysis, there's no particular advantage to use one approach over another.

However, from the standpoint of being able to clearly show what's happening end-to-end of a workflow, this type of duplication of process nodes is more useful.

[pause]

As we look at this diagram, even with these changes, we still haven’t identified all the processes that need to be in place.

So, let's repeat this exercise.

# Level 3 DFD

Here is the level 3 DFD.

The introduction of the database and network managers exposes the need for additional interfacing processes.

The database manager needs to access the filesystem in order to get to the database.

And so, we've inserted the filesystem, and also an operating system boundary.

Likewise, the network manager doesn't connect directly to the network, it requires a network driver.

[pause]

The reason we don’t need a boundary between the network manager and the network driver is that both are owned by the operating system.

The database manager, on the other hand, is not, which is why we have a database boundary rather than an OS boundary there.

[pause]

Just by having the boundaries, we can make certain inferences about the processes themselves.

We'll address how to make that more obvious in a bit.

An important thing to note here is just as with the operating system having its process nodes replicated, the same is done for the database manager nodes.

Likewise, the boundaries all have unique names, which means that if you see a boundary in one section labeled OS boundary, and also another section, then you know that if we were to create an unfolded form of the DFD, that boundary would be contiguous.

However, there is no motivation for us to do that as it would only complicate the diagram. As it stands, this diagram is still comprehensible from the standpoint of the information that we got from development.

As you can see, we still have these colored regions, which we took from the system diagram, and all the entities are logically placed within the context of each of these.

Now let's look at how we're going to go about taking this information and documenting it.

# Documenting

Now, it is not the case that you can just take your three levels of DFD, and you do need all three of them, and call it a day.

As a side note, your actual system may require more or less of these base DFDs.

[pause]

So, you do need all those diagrams, but they are insufficient for the purposes of threat modeling, but we’ll address that later.

[pause]

The tools that you're using to create these DFDs could be anything that's out there that's available and able to give you this representational form.

But there's additional information we need to capture.

We're going to cover that next.

[pause]

The remainder of this presentation is going to be based on the least common denominator of tooling.

We’re going to use an Excel workbook with multiple sheets to capture various pieces of information.

If you happen to be using a tool that allows you to capture the same type of information, that's fine.

This information will be carried into the analysis activity, so you should be able to see as we evolve this material, why these choices were made.

# Level 1 DFD - IDs

So, back to the level one DFD, we're going to apply IDs to everything.

We have the five entity types, and they are:

* resources
* processes
* interactors
* boundaries

and

* data flows

For the purposes of this example, we're going to use the following ID prefixes:

* ESR example system resource
* ESP example system process
* ESI example system interactor
* ESB example system boundary

and

* ESD example system data flow

This is for convenience.

If you use a tool such as, the Microsoft Threat Modeling Tool, every entity is going to be assigned a GUID.

However, it does have the drawback of being fairly opaque when it comes to human comprehensibility.

So, for the sake of being able to show this in a way which is easier for a person to look at, we’re going to use the unique identifier prefixes as mentioned.

And as you can see each data flow, process, boundary, interactor, and resource has been given an identifier. There's no necessity to put them in a particular order or to have them be sequential.

# Level 2 DFD - IDs

Similarly, with level 2, we have more processes and an additional boundary that we need to include.

# Level 3 DFD - IDs

And by the time we finished IDing all the level 3 entities, we're up to a fairly good number that we need to keep track of.

# Initial Entries

Here is the initial set of entries from this exercise.

There is a template provided with the AVCDL material, and a link to it should be attached to this presentation’s description.

There are individual sheets for each of the different entity types: resources, interactors, boundaries, processes, and data flows.

The first few columns of which are shown here.

You would have the ID, name, and a description field.

If you're using the recommended naming convention for data flows, it's not really required to put in a description because the name tells you that it's going from a source to a destination, whereas for processes, name may not be sufficient to convey what it’s for.

# DFD Entity Attributes

Here’s the DFD entity attributes diagram and attribute lists from the Threat Modeling Overview video.

We’ll get into the specifics on each of these, what the attributes are, and why we care about them.

# Resource Attributes 1

For our purposes, we’ll consider resources as data-at-rest.

This is a fairly gross simplification because a resource could be an external data source.

However, from the perspective of the DFD and the threat model, we look at it as if the resource is a static object.

[pause]

Within the space of the cybersecurity requirements taxonomy, resources are in the at-rest group of assets.

There are six buckets for resources:

* executables
* configuration data
* unstructured data
* databases
* credentials

and

* logs

These in and of themselves are artificial divisions, but what they represent are specific entities that we wish to be able to make various cybersecurity property assertions about and establish requirements for.

[pause]

Within the context of threat modeling, we have the following attributes.

If the resource has a unique ID, then there is some mechanism that allows us to know that this resource is the resource that we want.

This is most important when you're dealing with executables.

The reason for that is that if you only use a mechanism such as the file name, there's nothing preventing another file from being named the same.

The problem from a cybersecurity standpoint is that we have issues of file paths being a chunk of metadata that tells us where to look for things.

And if all we do is look at the file and don't care about the path, this leaves us open to a whole host of issues.

[pause]

Next, we have whether the resource is logically read-only.

For something like an analog data stream, obviously it's a read-only entity.

However, if we're dealing with base configuration information or other information which has no need for right access, then we want to be able to assert that it is read-only as a layer of protection.

[pause]

Whether or not the resource is structured is important because that informs us as to whether or not we're dealing with a database.

[pause]

It's important to know whether or not the resource contains either PII or credentials.

Both are important and significant.

We deal with them separately, but fundamentally, they are a characteristic of data within the data store.

[pause]

Whether or not the data is encrypted.

Note that the mechanism of encryption is not specified here.

We only need to know that this layer of protection is present.

[pause]

Whether the data is integrity checked.

Integrity checking both establishes that the data has not been tampered with, and also that it hasn't been corrupted accidentally.

[pause]

Finally, the resource type is identified.

It can be analog; something not subject to cybersecurity consideration.

It can be an executable. This covers all executables, and it doesn't matter whether we're dealing with drivers, services or application binaries.

There is distinction between a compiled executable, interpreted code, or something which exists as either firmware or microcode.

These are sets of instructions which the processor uses in order to perform operations on data.

It can be configuration data which represents metadata used to establish the behavior of the processes.

The final two types are logs.

One is data, and one is audit.

There's an important distinction there.

Data logs are generic information collected for the purposes of the system itself.

Audit logs are there for the purpose of tracking specific system state changes which are, in the case of cybersecurity, relevant cybersecurity events that occur in the system.

# Resource Attributes 2

In the case of our example what we have is a set of four resources, two pairs, two for the core service, two for the console interface.

One of each is a flat file that represents our configuration data.

The other is a database, which is our live data.

[pause]

As we can see from the tabular form of the data, some of the information that we record is fairly obvious from the DFD itself.

And that's great. If certain data can be derived, for instance, whether it's read-only, whether it's structured, the resource type, then feel free to prefill those particular things before getting into the question and answer with the development SME. Keep in mind that there are things that are going to require their input and “I don't know” should not be an unexpected response.

Remember that guessing is never an option when you're doing this type of cybersecurity analysis.

If you can't assert that something is the case, then it probably isn't.

If they don't know whether something is encrypted, don't presume that it is.

Things will need to be verified, and there needs to be a positive assertion that something is the case.

# Process Attributes 1

For process attributes, the applicable area in the requirements taxonomy space is within the at-rest assets, specifically executables.

Now, obviously, when they're being executed, they're not at-rest.

However, they don't represent the externalizable data.

What we have here is a particular bucket that we're looking at, and what we care about are the cybersecurity properties that are applicable to it.

Within the context of threat modeling, we're going to care about whether it has a unique ID. We’ve covered unique IDs within the context of resources.

Within the context of executables, we care about these as they're running, as opposed to when we're looking at them as a resource.

So, there's some means that we can use to uniquely identify the process.

[pause]

There are a pair of attributes that we have, that indicate whether the process is authenticated and authorized.

Authenticated being that it is who it says it is.

So, we can look at the process and we say, yes, this process is a valid process that’s recognized.

The second is that the process has authorization to perform the operation it's attempting.

On some systems, these authorizations are known as entitlements.

So, for instance, you may have a process that is responsible solely for managing a resource.

That process should not be authorized to access the network.

There's no necessity for it to have that type of capability.

[pause]

There are three types of processes

The first are drivers.

These are your lowest level process. They're typically in the kernel space and are responsible for interfacing to some chunk of reality.

These are things like serial drivers, network drivers, disk drivers, things of that nature.

Second have services.

These are special processes which sit within the running system.

They tend to be at a higher level of execution than normal processes.

And they do things that are outside the scope of an individual normal user space executable.

For instance, the printer or database manager could be considered a service.

It just kind of sits there, it waits, you talk to it, and things happen.

It tends to have specialized interfaces in order to communicate with it.

Lastly, we have normal user-mode processes.

[pause]

In addition to type processes fall into one of three classes.

One is the operating system processes.

I'm not going to make any distinction between elements of the operating system, whether that be the kernel itself, versus operating system utilities that sit on top of that, or services or drivers or whatever have you.

This is an ownership and trust concept.

Next, we have third-party processes.

This is the class of all process not owned by us or the operating system.

So, it could be open source.

It could be a supplier component.

It’s not relevant, just at a lower level of trust than ourselves.

Next, we have self.

It's like everything that we own.

Finally, we have external.

These are processes outside the element which it is communicating with.

Now, you could create much higher levels of granularity, but for practical purposes, this is sufficient for our purposes.

# Process Attributes 2

# If we consider our example and focus on the processes, we can see that there are eleven of them.

# The process types are fairly obvious from the DFD.

# For the remainder of the attributes, we're going to have to ask questions, but we can make a couple of assertions.

# One is that the operating system processes are always going to considered authorized, and authenticated, and their process class is OS.

# Another is that the element’s processes considered to be authorized and authenticated within the context of their intended use.

# As for the other attributes, we need more information.

# It's important to note that it's not always the case that we’ll get this data right.

# So, don't consider the threat model to be a read-only document.

# It is going to be revised at some point.

[pause]

When you're dealing with external entities, treat them as if they don't have unique IDs. Since they're external to the system, there is no control over these things, and they may or may not be authorized and authenticated.

# Interactor Attributes 1

# For interactors we only have a type.

# If it's a person, we know what people are, we have certain expectations.

# For instance, there should be interfaces that front-end them.

# Otherwise, what we have is analog data that we're going to do A-to-D conversion on.

# There are no cybersecurity controls that can be applied here.

# The responsibility lies with functional code to ensure that the data is reasonable.

# Interactor Attributes 2

Our example has one interactor.

It's an administrator, as noted before, this could just be labeled as user, but it's labeled as administrator, and the type is person.

# Boundary Attributes 1

Having different types of boundaries is useful from the standpoint of threat modeling, but not necessary in the context of the DFD.

This is because we capture this information within the processes and data flows.

We add this attribute because it helps us to reason on the DFD non-programmatically.

[pause]

Let’s consider the boundary types.

Machine boundaries exist between the element’s hardware and the outside world.

Network boundaries exist between processes using networking as a communication mechanism.

Interprocess boundaries exist between processes using operating system mediated, process-to-process communication.

Operating system boundaries exist between operating system processes and other entities.

Resource boundaries exist between a process and a resource it manages.

External interactor boundaries are fairly arbitrary.

One could argue that you don't have an external interaction unless you are crossing a machine boundary type.

The same thing could be argued for networking, but it's not necessarily the case.

You could use networking between processes, even though inter-process communication may exist within the same machine.

# Boundary Attributes 2

Within our example, we have four boundaries, the core services boundary, which is an inter-process boundary.

[pause]

We have the machine boundary between our element and the outside world.

We've subsumed the network boundary here for the purposes of convenience, because we also have serial as well as networked communication.

And so, we put a machine boundary there, since it serves the same purpose.

[pause]

We have an operating system boundary between the core service and the filesystem, and also the core service and the Interprocess communications system.

[pause]

And finally, we have an interprocess boundary between the element’s processes and database manager.

# Data Flow Attributes 1

Finally, we have data flow attributes.

Arguably, these are the most important because this is where most of the action happens.

In our cybersecurity requirements taxonomy space, this is the data-in-motion asset region with requirements at the network and protocol layers.

Attributes are related to the cybersecurity properties.

[pause]

The first attribute to consider is whether the data flow requires an acknowledgement.

Although this may seem trivial, it's important to properly manage protocolled communication.

Failing to consider requirement acknowledgement can lead you into all sorts of issues.

[pause]

Next, we have sequence numbers.

This is not an issue for standard protocols because they all use sequence number for managing out-of-order and duplicate messages.

It is a concern with custom protocols, since it leaves the system open to a host of attacks.

This is a design deficiency which can become fairly important as things scale.

[pause]

The next pair we have is authenticated source and destination.

These are foundational to secure communications.

[pause]

Next is whether or not it's safety critical.

Safety criticality is not the only thing that we have that we could consider.

We could say that this is cybersecurity critical.

However, we tend to think about things which are safety critical as being the highest level of import when it comes to the controls we have to have in place.

[pause]

Next is the integrity check which has three options.

The first is no check.

The second is a simple integrity check, such as a CRC.

And the third is a cryptographic integrity check.

[pause]

Finally, is encryption, also having three options.

The first option is no encryption.

The second is symmetric encryption.

And last is asymmetric encryption.

[pause]

Obviously, protocols that are running on data flows can be layered, and one must consider each of these separately.

If you have data flows between two processes that exist at different layers, you need to show them separately in order to reason on them appropriately.

# Data Flow Attributes 2

As one might expect, in our example, data flows represent the majority of items.

Not called out as an attribute is a protocol associated with every data flow.

We have function calls, IPC, HTTP, and HCI (human-computer interface).

[pause]

And then we have a whole host of attributes that we've shown here.

You'll note that there are blanks.

This is information that we were not able to get.

Data flows is where this is probably going to occur most often.

[break]

In the next set, we have items 28 through 54.

Looking to items 39 and 40, we can see that we have that browser to network manager, and network manager to browser, which have TCP/IP flows running below of the HTTP flows.

This is the case described earlier, where you have different protocols with different mechanisms that you need to account for.

[pause]

Our last protocol here is serial I/O.

Now, the template does not provide a set of these because it's impossible to know what all of them are.

And so, they are just left for the cybersecurity SME to put in place.

# OS-centric DFDs

One might think that by the time that we get to something like a level 3 DFD that we're pretty much done, especially once we've filled in all of the various attributes for things.

But actually, this is not the case.

[pause]

From the standpoint of threat modeling, what we've done is collected all of the data.

However, that data is not organized in a way which makes it easy for us to do an analysis.

Yes, we could look at the entity-to-entity flows, but as has been shown, there’s no way to talk about a particular workflow.

All we’ve done is show everything in the element and how it’s connected.

[pause]

We have not contextualized it.

That's what we're going to do next.

We'll start by creating OS-centric DFDs.

# OS – Filesystem

Starting with the filesystem component of the operating system, we're going to show all of the touch points that the system has with it.

This does two things.

One is it gives us a base to be able to look at cybersecurity controls which may not be within the context of threat modeling but are appropriate to implementation.

This allows us to reason on those and say, yes, we've got these various controls in place.

The other thing is that this allows us to double check that we haven't missed anything that we're going to need to be able to reason on the element.

[pause]

You'll notice that this diagram is fairly straightforward, and we can reason on this directly.

Obviously, since this is the filesystem that we're talking about, this is a highly trusted component, and so, all the communications going to it through these data flows is trusted.

[pause]

A minor note.

This DFD shows our color coding for levels of trust, green being the operating system, blue being ourselves, yellow being third party, and red being external (not shown here).

# OS – IPC Manager

The next component within the operating system that we'll consider is the interprocess communications manager.

Here we have the core service, the console interface, and the web server.

# OS – Serial Driver

We have the serial driver, and you'll note on the left, the console interface in blue, and the terminal in red, indicating that it is external to our system.

Note that we're not using any boundaries here because the color coding does that indication for us.

# OS – Network Manager

With the network manager, we only have the web server talking to it.

Note that we're not showing the network driver on the other side, because it's also an operating system component, and we don't care about operating system to operating system components from the standpoint of our analysis.

# OS – Network Driver

Finally, we have the network driver component, and it's talking to the web browser.

It’s worth noting that in some operation systems, the network manager and driver may be combined.

[pause]

So, with this, we have covered all of the portions of the most trusted part of the system.

We can look at those, emplace controls, and then not have to worry about that layer in the context of us doing analysis for the element itself.

# Networking DFDs

The next set of DFDs to look at are those related to networking as a whole.

This is an especially important, because so much of what is done within a systems deal with communications between various elements and systems relies on the network stack.

It's also one of those areas where people don't give sufficient attention.

So, it's worthwhile to take some time to diagram specifically.

# Networking

In the case of our example, we only have one network path that we go through, that connects our web server and our web browser.

It traverses both the network manager and eventually the network driver.

We’re showing not only the trust levels, but also the boundaries, just for emphasis.

# Workflow DFDs

The final and arguably the most important set of DFDs we're going to look at are the workflows.

These are the operational flows that occur in the element itself.

These are the ones of primary interest because issues discovered in threat modeling needed to be mitigated, and if you give the person who needs to do this, huge diagrams containing a ton of superfluous information, it makes it that much more difficult to perform the mitigation.

And so, these diagrams are small by intent.

If an overly complicated DFD is needed to represent a workflow this is a good indication that the workflow is itself overly complex and should to be simplified.

# Initialization – Core Service

Our first workflow is the initialization of the core service.

It's a fairly trivial one where we have the core service itself, and its two data sources.

Note that there’s a block of text to provide a little bit of elaboration, and this is in the DFD file itself.

The reason for this is that it allows us to have information which makes it easier to interpret what the DFD is doing in certain situations.

It's not mandatory, and yes, it could have references to external documents, but where it's lightweight, there's no reason not to include it.

# Initialization – Console Interface

A second workflow is a similar one, and this time it's for the console interface.

# Core Service Configuration

In our third workflow, we're showing how the console interface is used to configure the core service.

# Normal Operation

And finally, we show normal operation.

This is a perfect example of how little we require in order to convey a particular workflow.

# Recap

We’ve seen how to translate a block diagram of the element into a series of DFD views.

We’ve covered what the attributes for each DFD entity type are and how they are motivated.

We’ve worked through a realistic example system.

We’ve motivated why having a series of simple targeted views is superior to a single big bang DFD.

[pause]

That covers the basics of how you would create a DFD-based threat model.

This is not to say that all questions could possibly be answered.

However, the materials that have been provided should cover most things.

# Example Materials

The materials that were used to cover this example are in this location shown here.

It's important to note that this includes the DFDs, as well as the threat model, and model creation workbook that goes along with it.

# AVCDL on GitHub

All AVCDL materials, both in source and distribution forms, are available on our GitHub site, as shown here.

Because of the size of the repository, it's recommended that you either clone the repository or download a ZIP archive of it, if you're not familiar with using git.

Instructions for downloading a ZIP archive are linked to on the repository’s front page.

# AVCDL on YouTube

A set of training videos have been created to cover various aspects of the AVCDL that follow the training path found in the AVCDL primary document.

It's important to note that all AVCDL materials except for the introductory blog posts are intended for product cybersecurity engineering practitioners.

# Next Steps

The next trainings in the threat modeling sequence are threat model analysis and threat candidate triage.

Once you've completed all of the trainings related to threat modeling, you can proceed to one of the other trainings at this level.

These include:

* attack surface analysis
* penetration testing

and

* vulnerability identification

Additionally, if you've completed secure coding and its prerequisites, you can also proceed to:

* static analysis
* dynamic analysis
* fuzz testing

and

* secure code review

# References

Here are references to the source material used in the creation of this presentation.

They'll also be included in the video description.

Additionally, this presentation’s source material will be provided on the AVCDL GitHub repository.