



Working Together to Build Confidence

DoD Software Fault Patterns

Dr. Nikolai Mansourov
CTO



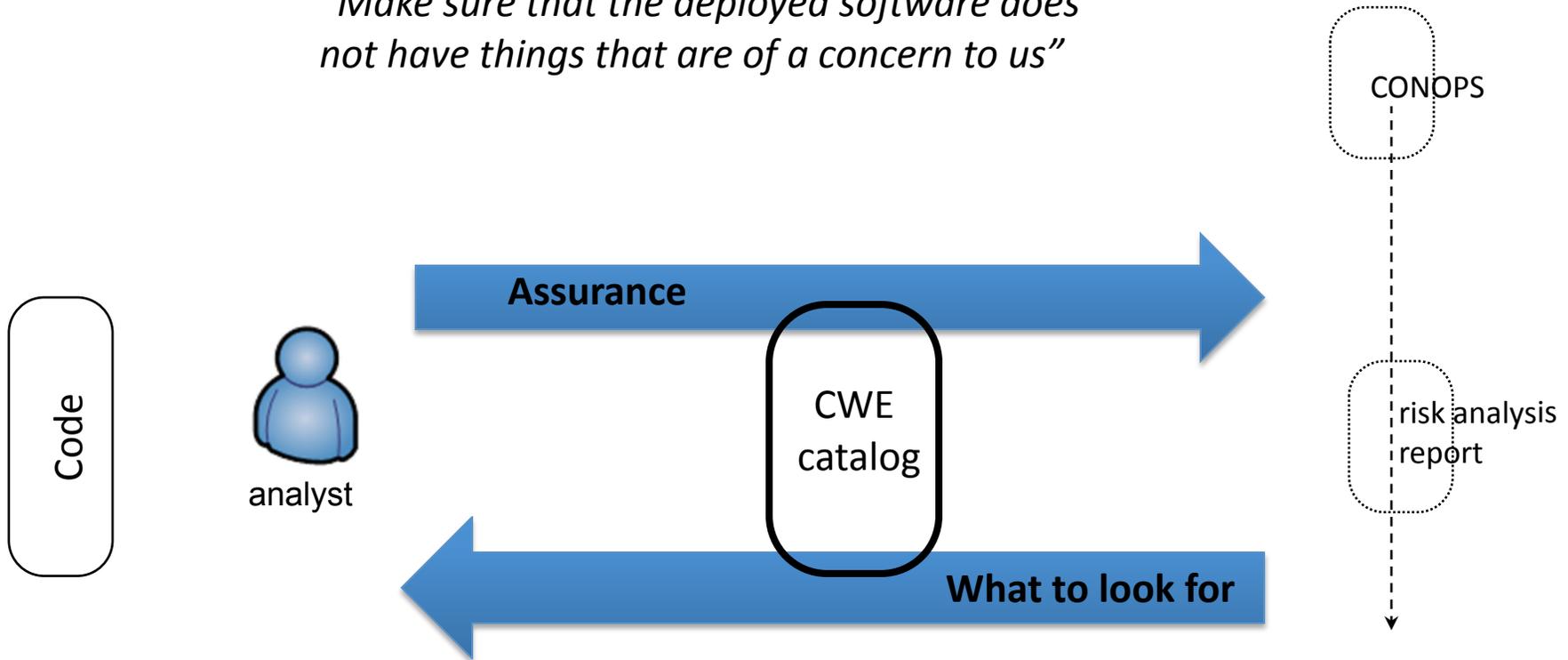
Software Fault Pattern (SFP) Research Program

- Develop a specification of software weaknesses/vulnerabilities that enables automation
 - Focus on *computation* as the viewpoint that can support automation
 - Computation is determined by system's artifacts
 - Code, data schemas, platform configuration, build scripts, etc.
 - Common, agreed upon vocabulary is defined in ISO 19506 (KDM)
 - Computation causes observable events
 - This is formalized as the Logical Weakness Model:
 - a necessary condition for a weakness
 - "condition" to confirm the weakness
 - Enables *mathematical* reasoning about vulnerability findings
 - Ensure *systematic* coverage of the "weakness space":
 - identified major areas of computations which are associated with security flaws,
 - identified common *patterns* of faulty computations
 - Aligned then with *impact* (focusing on injury, i.e. impact with a shortest causal link)

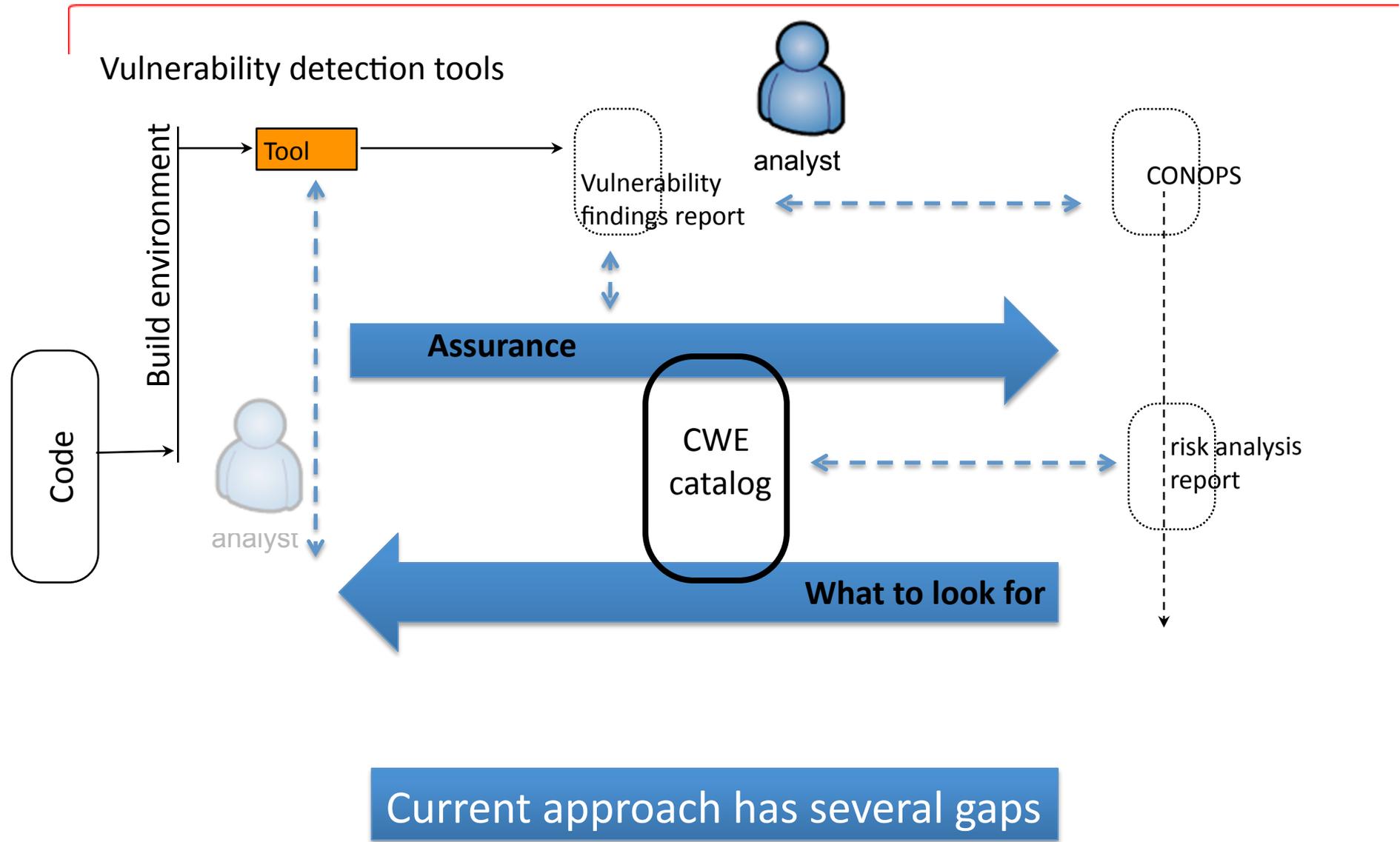


Current approach: CWE catalog

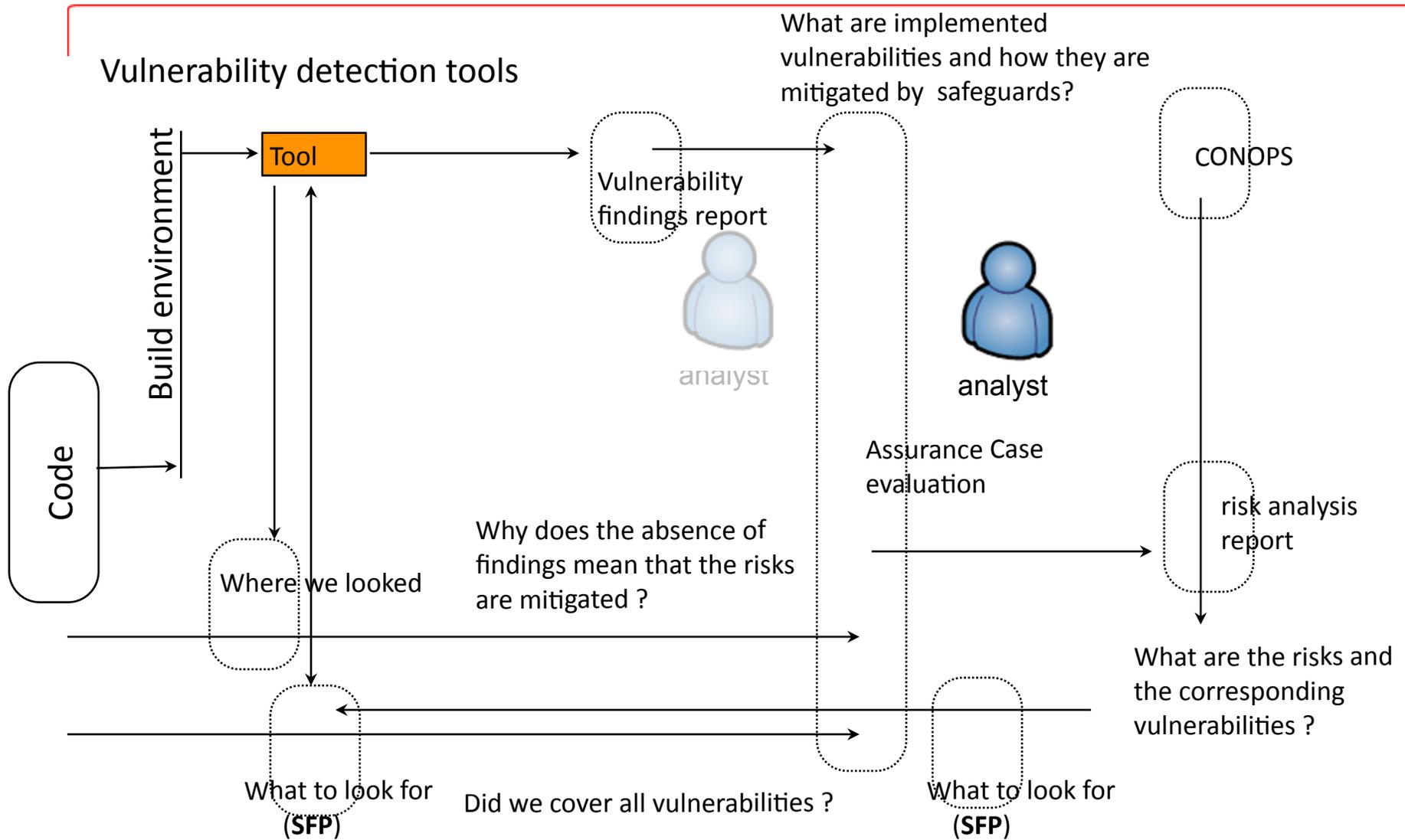
“Make sure that the deployed software does not have things that are of a concern to us”



Current automation



Better automation needs a specification



What is a Software Fault Pattern (SFP)?

- SFP is a generalized description of an identifiable family of *computations*
 - Described as patterns with an invariant core and variant parts
 - Aligned with injury
 - Aligned with operational views and risk through events
 - Fully identifiable in code (discernable)
 - Aligned with CWE
 - With formally defined characteristics

SFP approach: extending CWEs into a specification



What is a Software Fault Pattern (SFP)?

- SFP is a generalized description of an identifiable family of *computations*



- Described as patterns with an invariant core and variant parts
- Aligned with injury
- Aligned with operational views and risk through events
- Fully identifiable in code (discernable)
- Aligned with CWE
- With formally defined characteristics



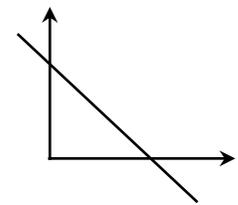
What is a pattern (that is not nebulous) ?

- Fact-oriented approach to pattern definition and discovery
- Pattern is a collection of *things* and *relationships* between these things (facts)
- Based on a well-defined vocabulary for “things” (nouns) and relationships (verbs)
- Same vocabulary is used to describe real *situations* (e.g. systems), resulting is a *factbase – discovery, phase I*
 - “real” things
 - “real” relationships
- This vocabulary is used to *define* patterns
 - Things that are “variables”
- Patterns are *matched* to the factbase (this is *discovery, phase II*)
- Vocabulary is the *conceptual commitment*

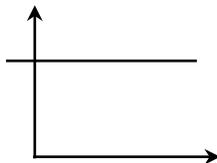
The key to defining a pattern is a vocabulary



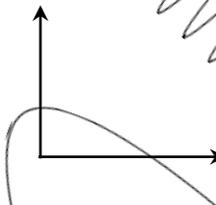
SFPs are parameterized families of computations



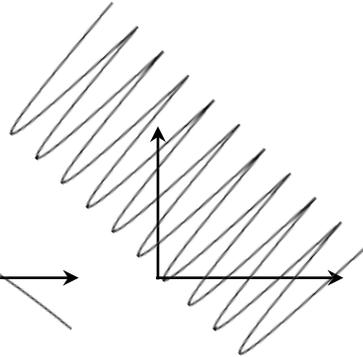
a straight line



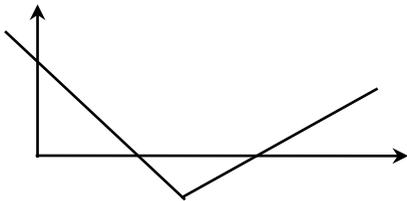
also a straight line



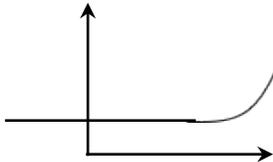
parabola



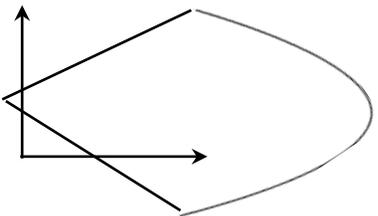
periodic line



a composite line



also a composite line



another composite line

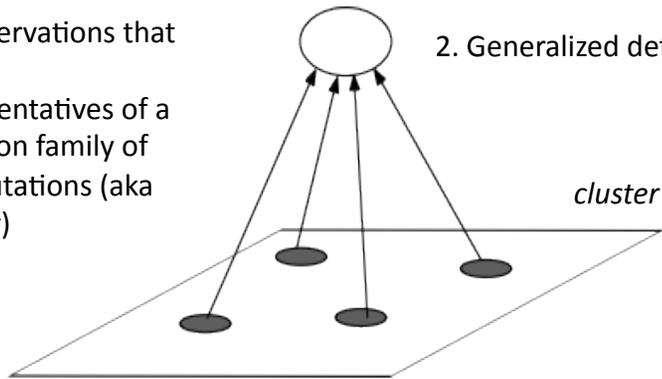
Parameterization is about creating a vocabulary of elementary “shapes” and characteristics of these shapes, focusing at the invariants and the variation points of each “shape”

CWE is predominantly a collection of observations, not a vocabulary of common “shapes”



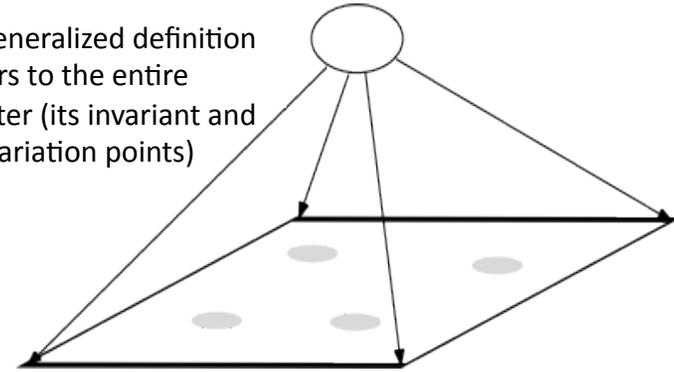
How to Identify Parameters and Why is that Important

1. Observations that are representatives of a common family of computations (aka cluster)

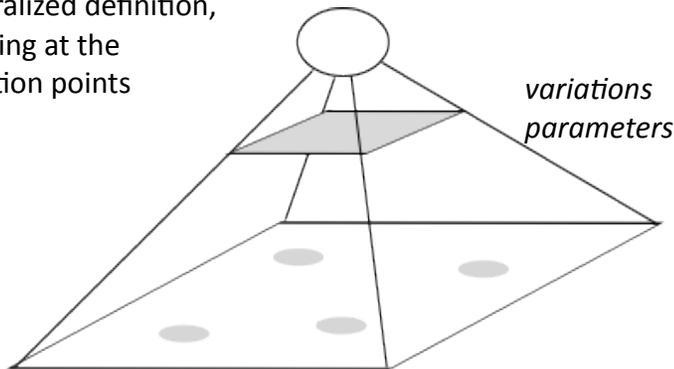


2. Generalized definition

3. generalized definition refers to the entire cluster (its invariant and its variation points)

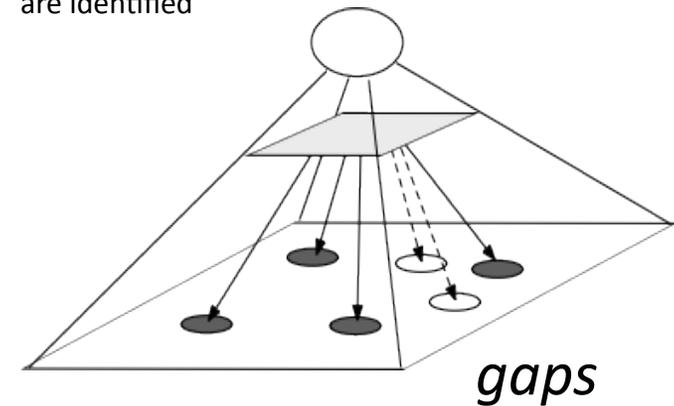


4. Parameterization introduces additional details for the generalized definition, focusing at the variation points



5. Variations are identified top-down in order to provide assurance of coverage

6. Parameters are mapped to the original observations, and gaps are identified

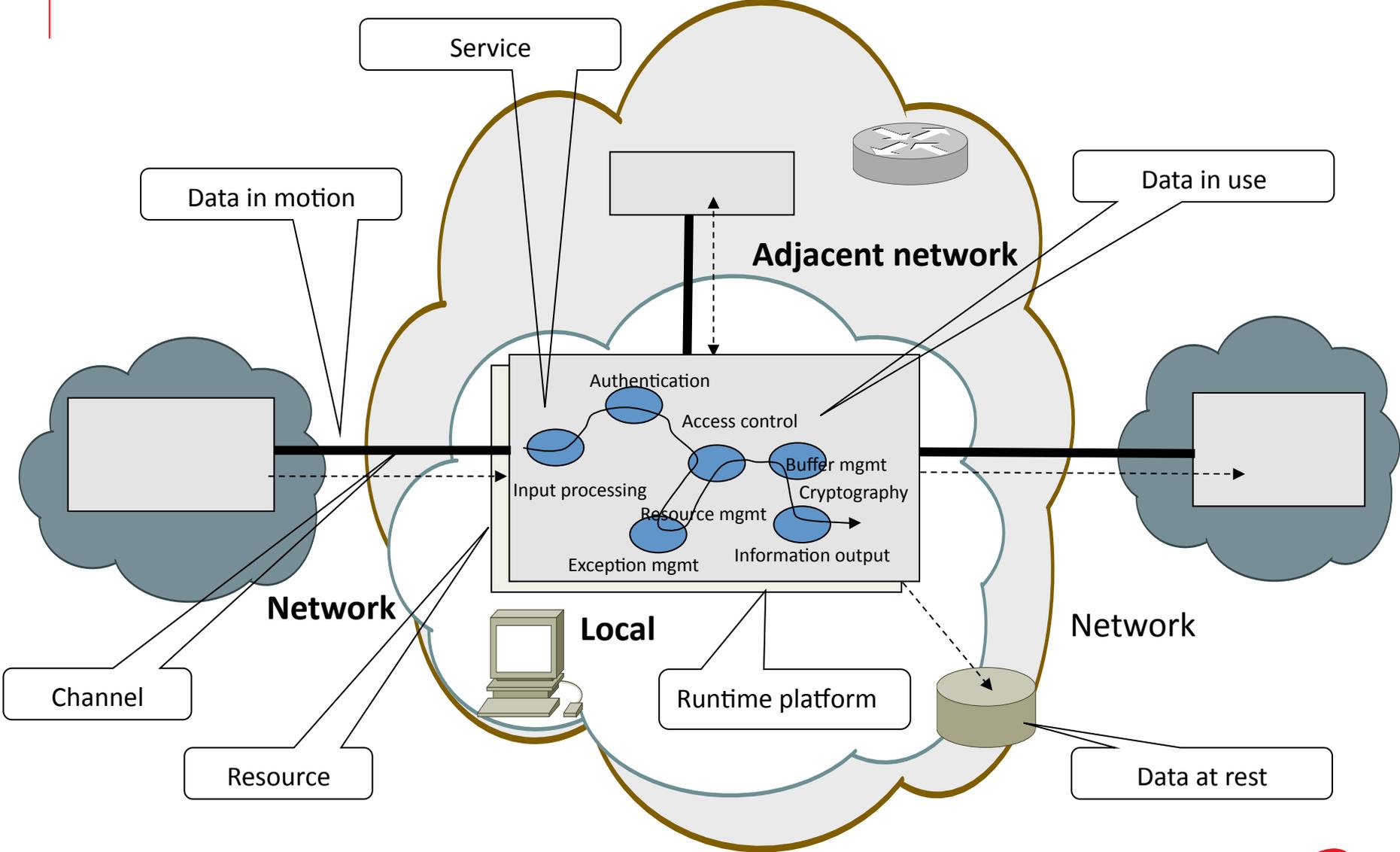


What is a Software Fault Pattern (SFP)?

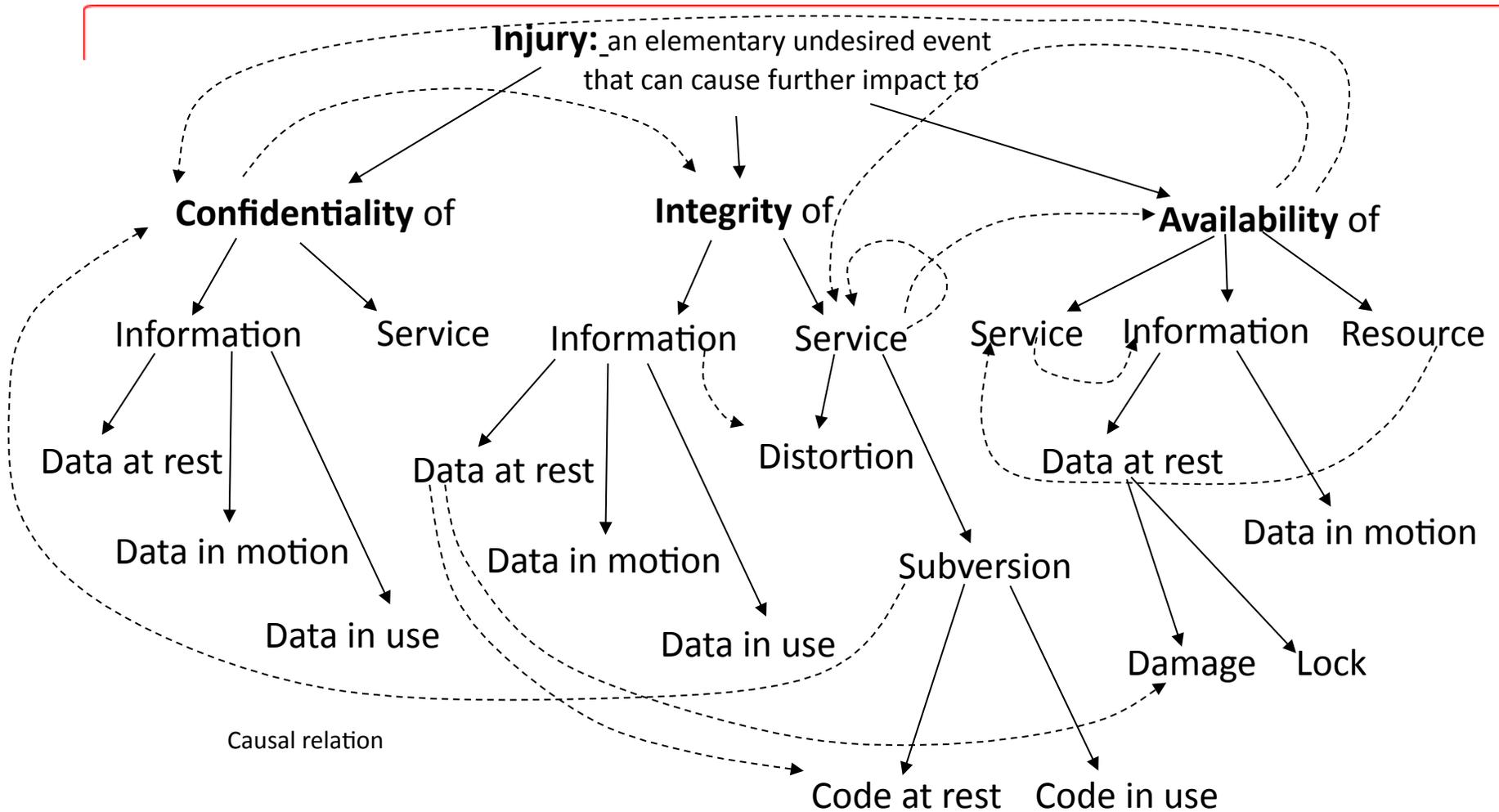
- SFP is a generalized description of an identifiable family of *computations*
 - Described as patterns with an invariant core and variant parts
 - Aligned with injury
 - Aligned with operational views and risk through events
 - Fully identifiable in code (discernable)
 - Aligned with CWE
 - With formally defined characteristics



Injury in the system context (aligned with CVSS)



Vulnerability and Injury



To guarantee the coverage of the “faulty computation” space, all injuries have to be covered

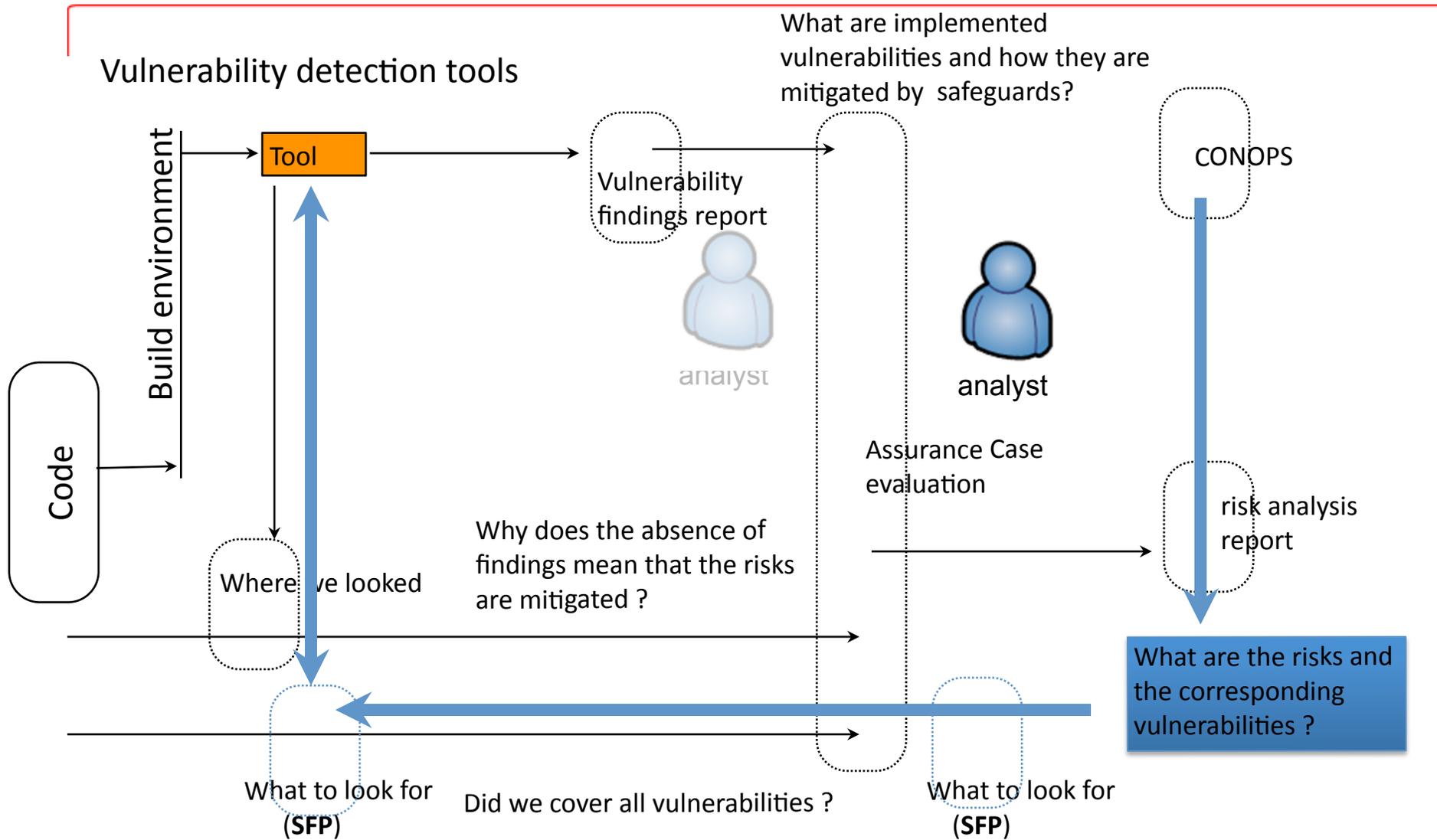


What is a Software Fault Pattern (SFP)?

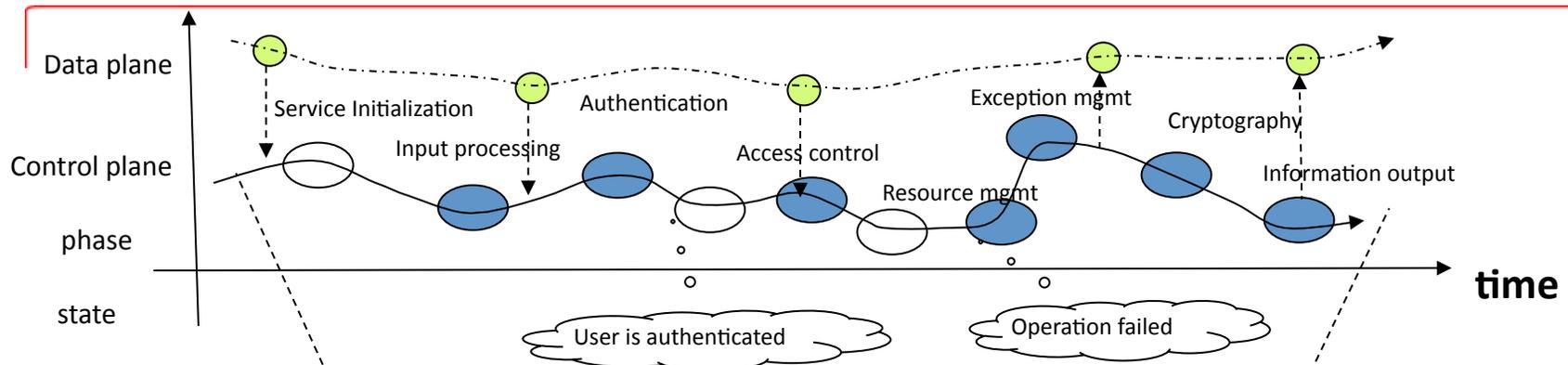
- SFP is a generalized description of an identifiable family of *computations*
 - Patterns with an invariant core and variant parts
 - Aligned with injury
 - Aligned with operational views and risk through events
 - Fully identifiable in code (discernable)
 - Aligned with CWE
 - With formally defined characteristics



Alignment with operational views and risks



Events, computation and code



Computation is an sequence of steps/events

flow pipes

“Code” provides constraints to computations and therefore determines what kind of computations can occur

Computation often performs steps that are common to large families of systems

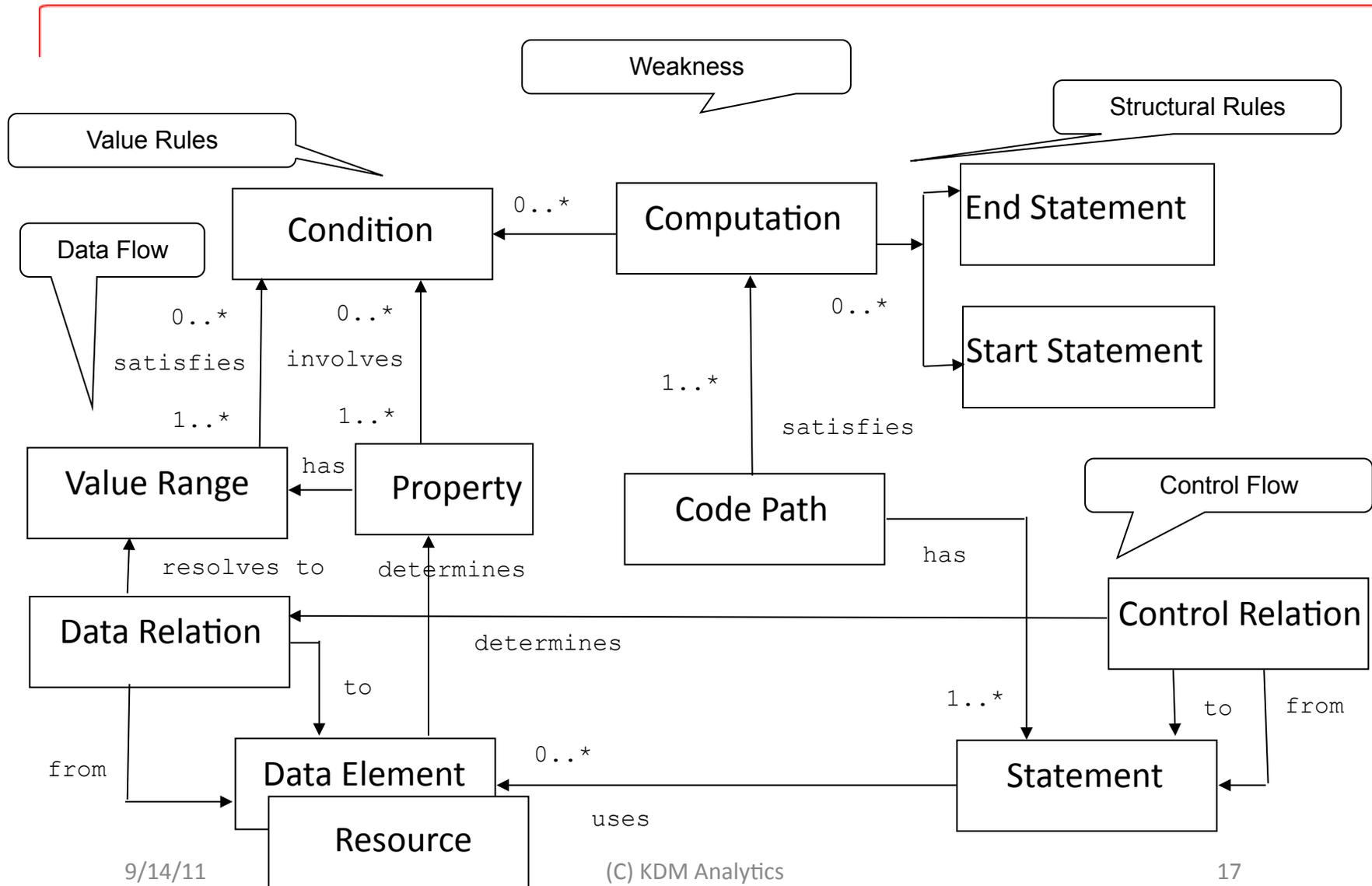


Certain “places” of “code” are indicators of particular computations

Certain “places” of “code” are necessary conditions for vulnerabilities



Weakness Logical Model



What is a Software Fault Pattern (SFP)?

- SFP is a generalized description of an identifiable family of *computations*
 - Described as patterns with an invariant core and variant parts
 - Aligned with injury
 - Aligned with operational views and risk through events
 - Fully identifiable in code (discernable)
 - Aligned with CWE
 - With formally defined characteristics



How does the new approach enables automation?



Common, agreed upon *vocabulary* for systems elements:

Pipework element

Pipe

Valve

Pump

Gauge

Meter

T-connector

Pipe is connected to pipework element

Normalized mathematical description of the system:

Valve1 is connected to pipe2;

pipe2 is connected to meter3;

Pump4 is connected to pipe5 and pipe6; etc.

Software Fault Pattern description is based on the system vocabulary

this makes all properties ***discernable***

this enables information interexchange between tools

allows mathematical reasoning about findings

allows mathematical reasoning about assurance



Not all characteristics are discernable

Discernable characteristic is a property of the computation, such as the *role* of a data element, the role of an action or of a region, which can be expressed as a statement in the vocabulary of the “code”

Is based on
----->



A non-discernable description is either ambiguous, uses ill-defined characteristics, or uses one or more non-discernable characteristics

A non-discernable description can be turned into a discernable one by:

- *Additional research to better scope*
- *More clarity and precision*
- *Additional facts*

- **Examples of non-discernable CWEs**

- **684 - Failure to Provide Specified Functionality**

- The code does not function according to its published specifications, potentially leading to incorrect usage

- **573 - Failure to Follow Specification**

- The software fails to follow the specifications for the implementation language, environment, framework, protocol, or platform

- **115 – Misinterpreted Input**

- The software misinterprets an input, whether from an attacker or another product, in a security-relevant fashion.

- **448 - Obsolete Feature in UI**

- A UI function is obsolete and the product does not warn the user.

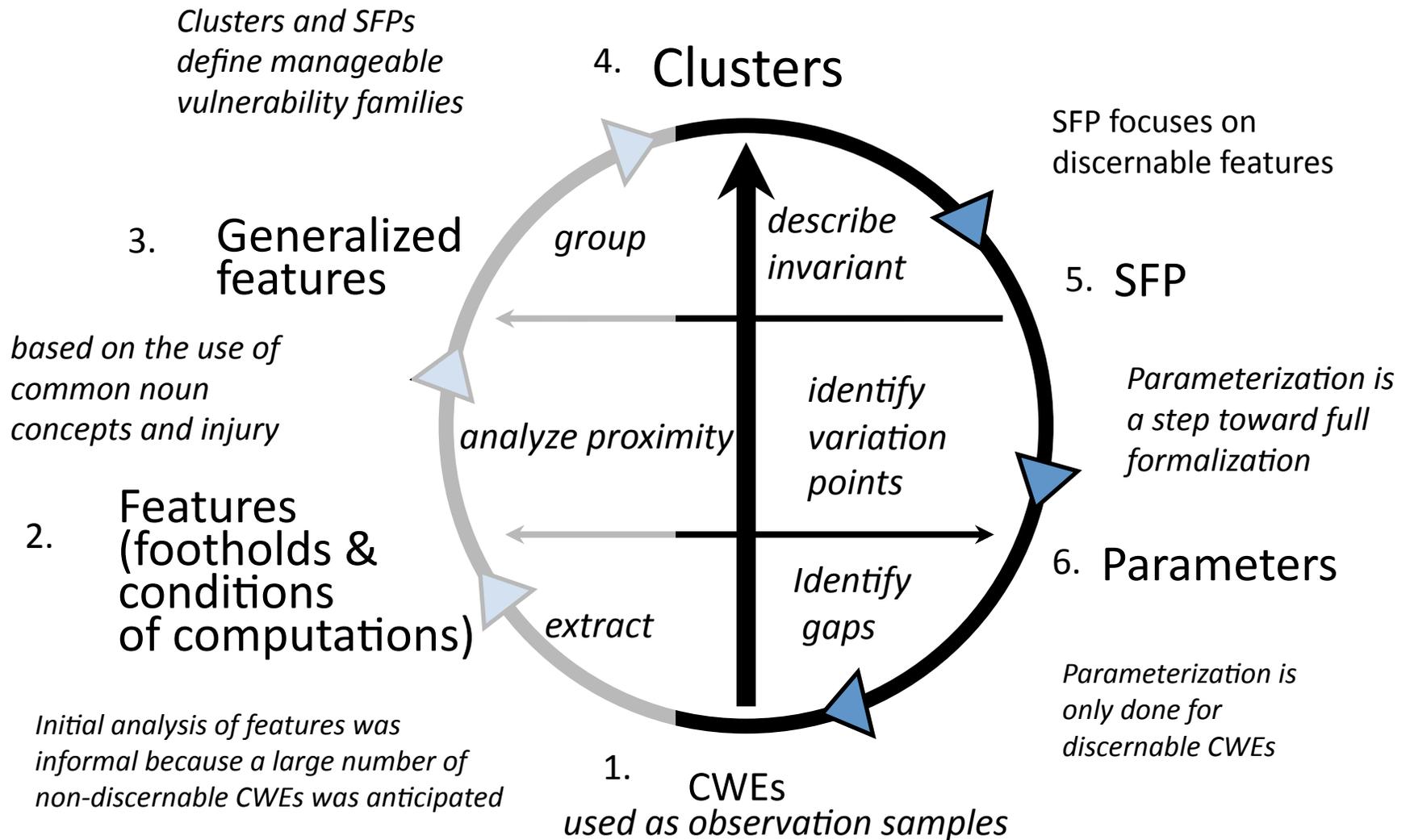


What is a Software Fault Pattern (SFP)?

- SFP is a generalized description of an identifiable family of *computations*
 - Described as patterns with an invariant core and variant parts
 - Aligned with injury
 - Aligned with operational views and risk through events
 - Fully identifiable in code (discernable)
 - Aligned with CWE
 - With formally defined characteristics



Methodology Behind Forming SFPs



How do we get there ? Methodology overview

- Bottom up process - Start with CWEs – as de-facto weakness space definition
 - We used CWE to identify common areas of computations
- Top down process - CWEs are no longer involved
 - Clusters, their characteristics – look at the nature of *all computations* in a certain area (good and bad); what are the common characteristics of these computations? Then use this a controlled vocabulary for defining weaknesses in this particular area
 - Focus at common detection (when can we distinguish a bad computation from a good computation in a given area; and how we automate this decision?)
 - Unique *foot-holds* of the computation
 - Agreed ontology between fact collection and vulnerability definition
 - Alignment with injury (defined in CVSS)



CWE enumerates various faulty computations

Home > CWE List > CWE- Individual Dictionary Definition (1.5)

Search by ID:

CWE List
Full Dictionary View
Development View
Research View
Reports

About
Sources
Process
Documents

Community
Related Activities
Discussion List
Research
CWE/SANS Top 25
CWSS

News
Calendar
Free News Letter

Compatibility
Program
Requirements
Declarations
Have a Declaration

Contact Us
Search the Site

Definition List Slice XML.zip

low')

Status: Incomplete

The input buffer is less than it can hold, or when the simplest type of program copies the buffer, it strongly suggests the use of "buffer overflow." This is a "buffer overflow", including less errors, integer overflow, and which variant is



What families of computations are covered by CWE ?



9/14/11

© KDM Analytics Inc.

25



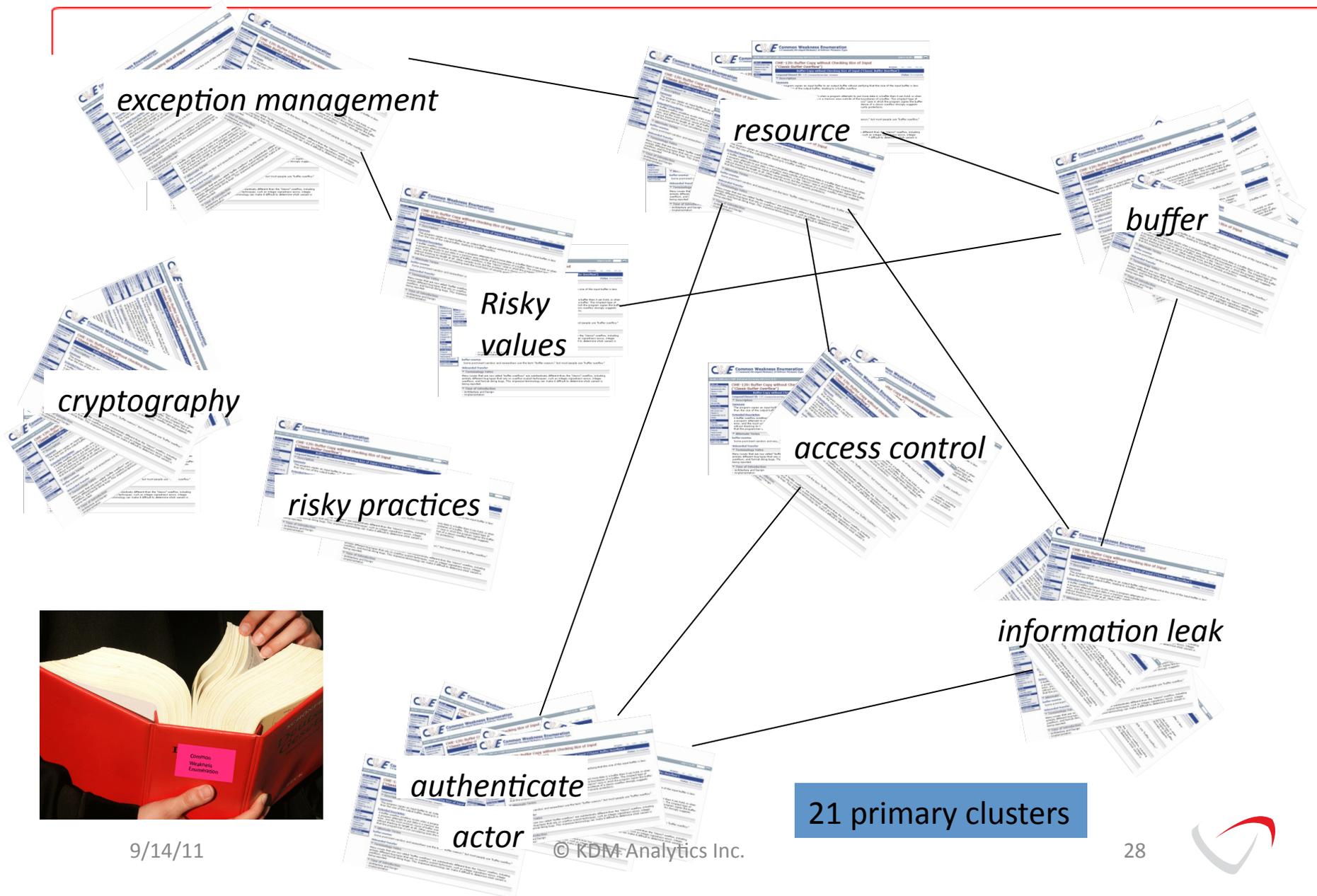
Identify common characteristics and group “close” CWEs



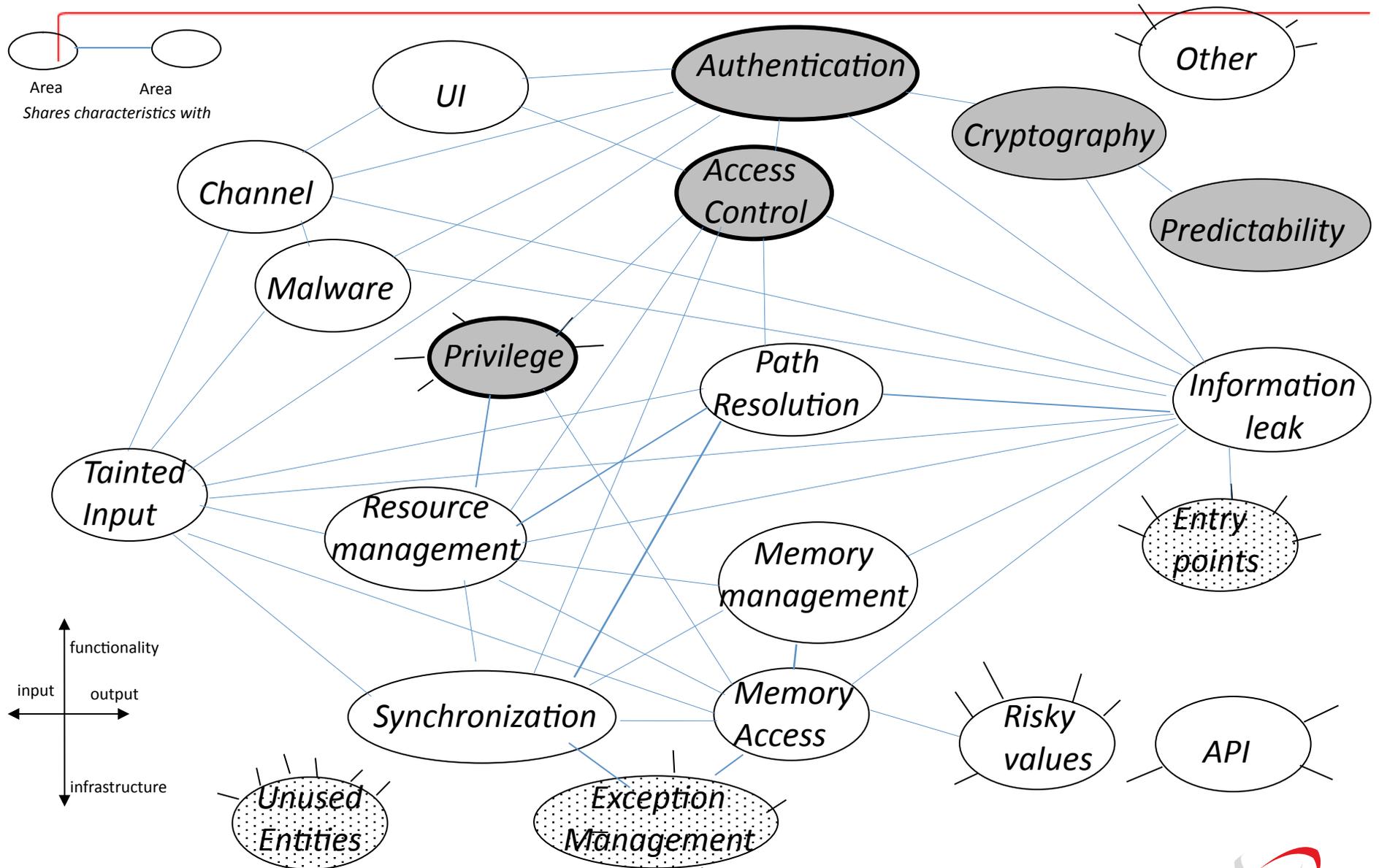
Identify common characteristics and group “close” CWEs



As the result, several clusters emerged



21 clusters and their associations

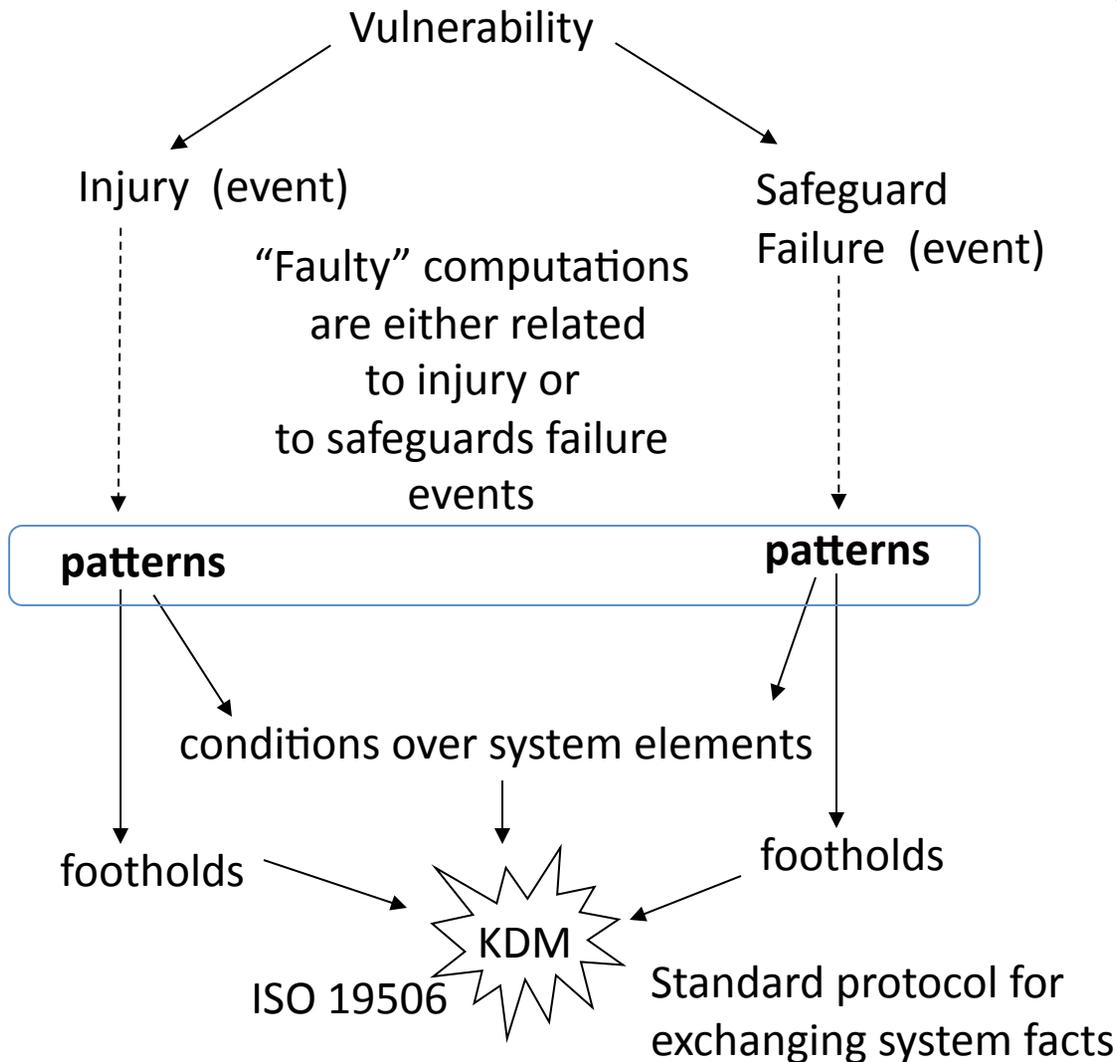


What is a Software Fault Pattern (SFP)?

- SFP is a generalized description of an identifiable family of *computations*
 - Described as patterns with an invariant core and variant parts
 - Aligned with injury
 - Aligned with operational views and risk through events
 - Fully identifiable in code (discernable)
 - Aligned with CWE
 - With formally defined characteristics



Machine-readable vulnerability patterns



Vulnerability: a bug, flaw, weakness, or exposure of an application, system, device, or service that could lead to a failure of confidentiality, integrity, or availability

Vulnerability involves an **event**

Foot-hold: a “known” construct or API call in the system’s artifacts that is *necessary* for the fault event to occur

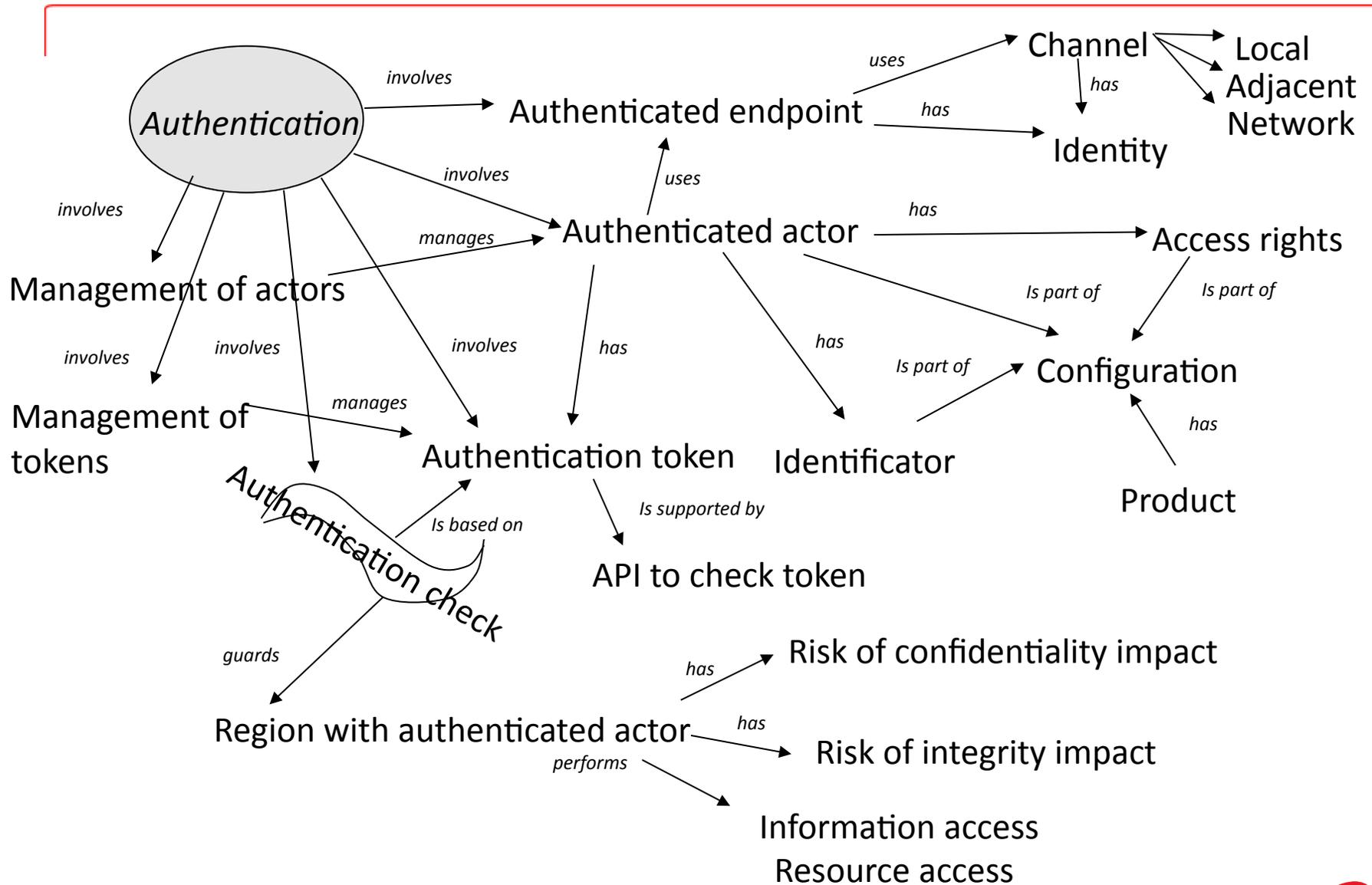


Discernable weakness description has “foot-holds”

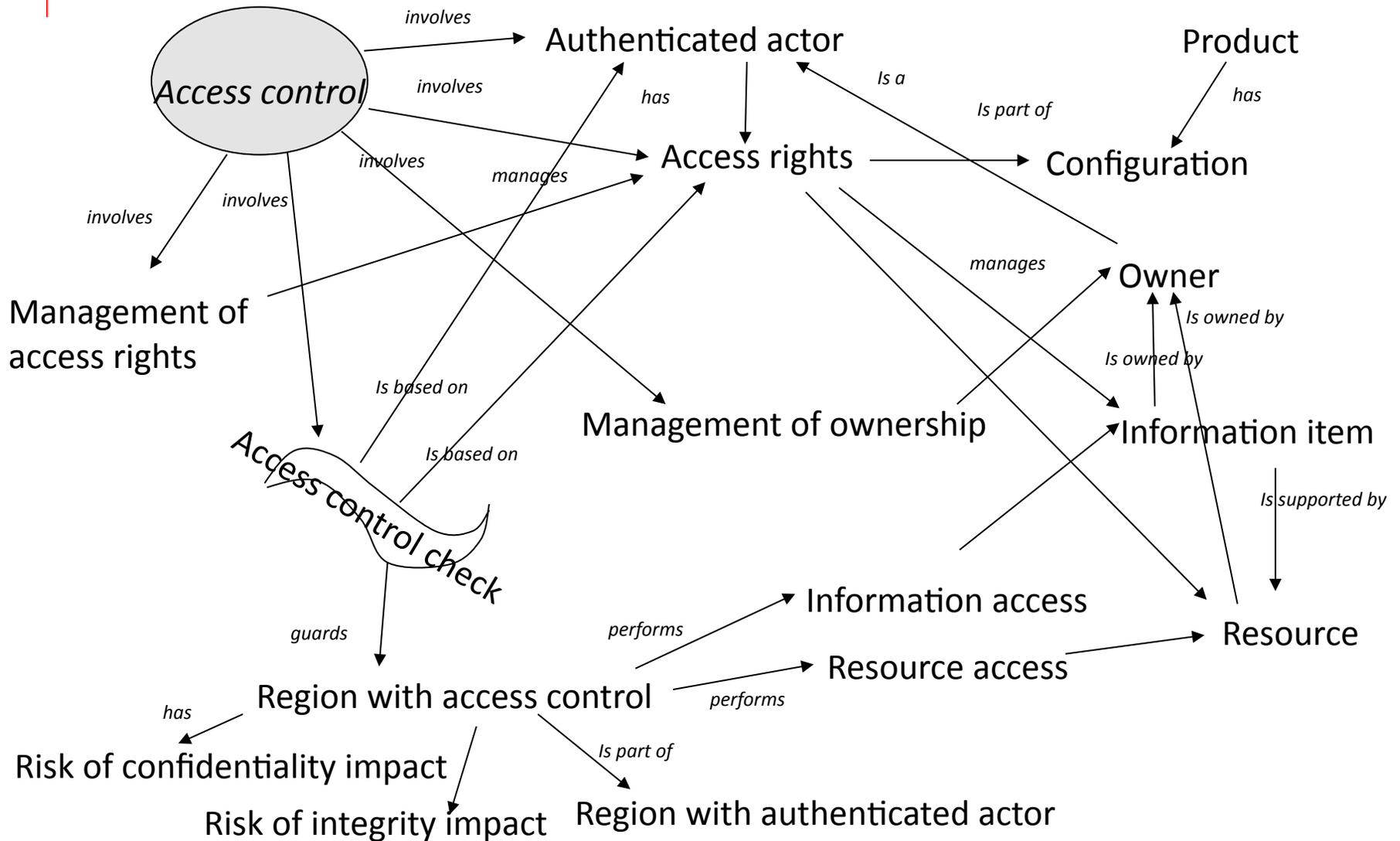
- “Foot-hold” – a tangible “place” of the computation that is a necessary for the computation to result in injury
- Classification of the “foot-holds”
 - API calls
 - Entry points
 - Programming language constructs
- Main “foot-holds”
 - Input port (exploitable vulnerability)
 - Output port (confidentiality impact)
 - Places where resources are modified (integrity impact)
 - Places where code can be modified (integrity impact)
 - Conditions (key to determine data constraints and properties)
 - Certain programmatic constructs (availability impact)



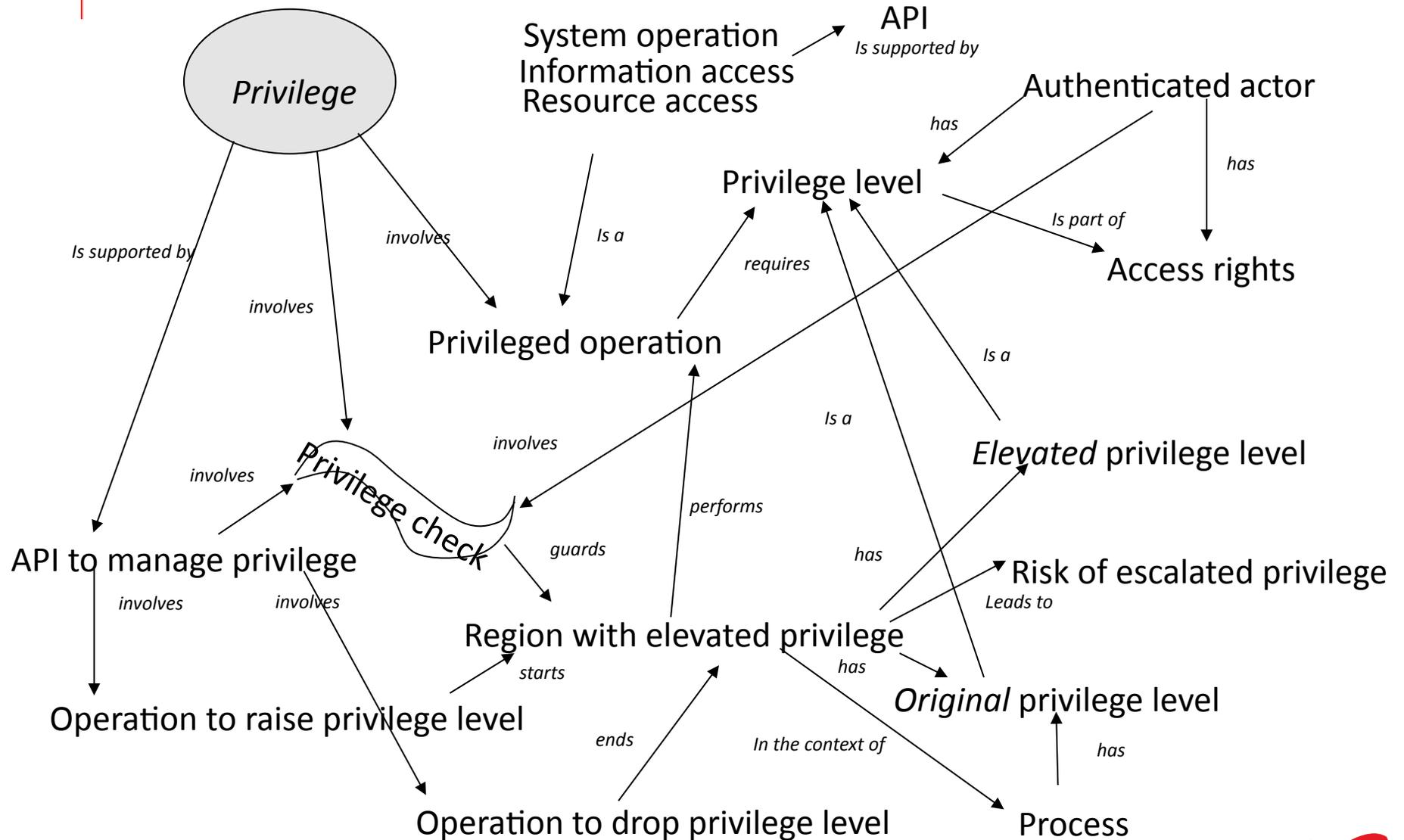
Examination of "Authentication" cluster



Examination of the "Access Control" cluster



Examination of the "Privilege" cluster



SFP EXAMPLES



Extracting and Generalizing SFP Features

CWE 194 Unexpected sign extension

The software performs an operation on a number that causes it to be sign extended when it is transformed into a larger data type. When the original number is negative, this can produce unexpected values that lead to resultant weaknesses.

Features are normalized and use standard vocabulary of noun and verb concepts

- computation involves data element DE1 of data type T1
- data type T1 is signed
- computation involves cast of DE1 to data type T2
- data type T2 is signed
- T2 is larger than T1
- value of DE1 is negative

primitive noun concepts

- ActionElement AE1 (cast)
- data element DE1
- data type T1
- data type T2

foothold
cast of DE1 to data type T

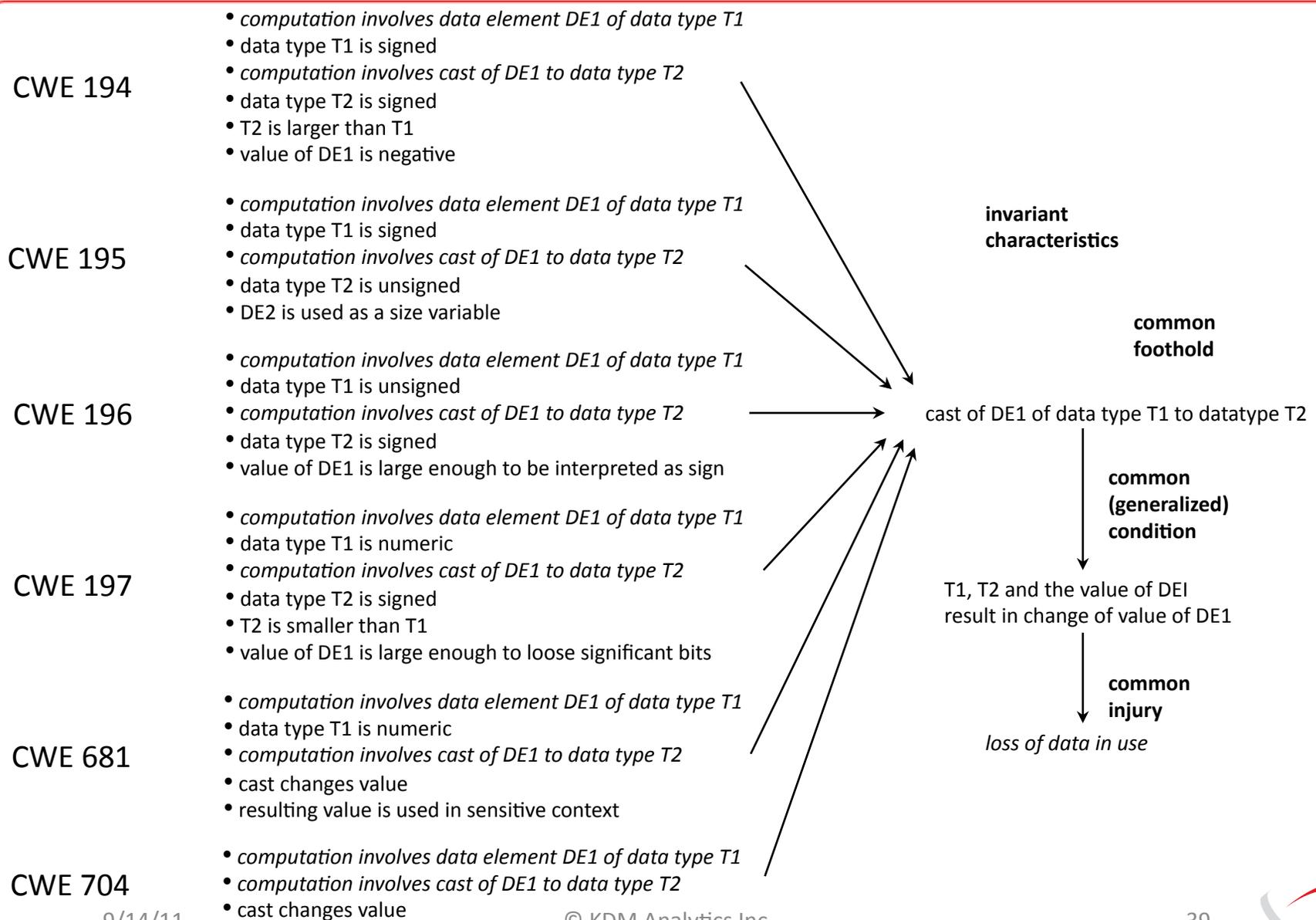
injury
Loss of data in use

- condition**
- data type T1 is signed
 - data type T2 is signed
 - T2 is larger than T1
 - value of DE1 is negative

this is an issue because under certain circumstances the cast operation violates a naive assumption that the value remains unchanged; this is a minor injury in itself, but it can be combined with other issues when the changed value flows into another region, e.g. when intersected with user access & unauthorized user or with resource control, authentication, buffer access or resource access



Focusing on Invariants



Example of formalized content

Unsafe Type Conversion

A weakness where *the code path* has:

- an *end statement* that performs cast of data value of datatype1 to datatype2 where cast operation modifies the data value



Bottom Up Identification of Variation Points

- *computation involves data element DE1 of data type T1*
- data type T1 is signed
- *computation involves cast of DE1 to data type T2*
- data type T2 is signed
- T2 is larger than T1
- value of DE1 is negative
- *computation involves data element DE1 of data type T1*
- data type T1 is signed
- *computation involves cast of DE1 to data type T2*
- data type T2 is unsigned
- DE2 is used as a size variable
- *computation involves data element DE1 of data type T1*
- data type T1 is unsigned
- *computation involves cast of DE1 to data type T2*
- data type T2 is signed
- value of DE1 is large enough to be interpreted as sign
- *computation involves data element DE1 of data type T1*
- data type T1 is numeric
- *computation involves cast of DE1 to data type T2*
- data type T2 is signed
- T2 is smaller than T1
- value of DE1 is large enough to loose significant bits
- *computation involves data element DE1 of data type T1*
- data type T1 is numeric
- *computation involves cast of DE1 to data type T2*
- cast changes value
- resulting value is used in sensitive context
- *computation involves data element DE1 of data type T1*
- *computation involves cast of DE1 to data type T2*
- cast changes value

extracted parameters

- • data type T1 is signed
- • data type T1 is unsigned
- • data type T2 is signed
- • data type T2 is unsigned
- • data type T1 is larger than data type T2
- • data type T is smaller than data type T2
- • value of DE is negative
-
- • value of DE is large enough to loose significant digits in in T
- • value of DE is used in sensitive context

This is a bottom-up approach that does not assure coverage



Top Down Identification of Variation Points

Unsafe Type Conversion

common foothold	common generalized condition
cast of DE1 of data type T1 to datatype T2	T1,T2, and value of DE1 results in change to value of DE1
common injury <i>loss of data in use, Loss of availability of service</i>	<i>because under certain circumstances the cast operation violates a naive assumption that the value remains unchanged;</i>

CWE 194
CWE 195
CWE 196
CWE 197
CWE 681
CWE 704

variations:

- value changes sign
- value is truncated
- value is enlarged

This is a top-down approach that does assure coverage

Extracted Parameters

datatype T1 (source)	datatype T2 (target)	relation between T1 and T2	data element DE1 (input)
<ul style="list-style-type: none"> • data type T1 is signed • data type T1 is unsigned 	<ul style="list-style-type: none"> • data type T2 is signed • data type T2 is unsigned 	<ul style="list-style-type: none"> • data type T1 is larger than data type T2 • data type T1 is smaller than data type T2 	<ul style="list-style-type: none"> • value of DE1 is negative • value of DE1 is large enough to be interpreted as sign in T2 • value of DE1 is large enough to loose significant digits in in T2



Parameterization example

Unsafe Type Conversion

A weakness where the code path has:

- an end statement that performs cast of data value of datatype1 to datatype2 where cast operation modifies the data value

SFP Parameters	Variation on injury			Source Data Type		Target Data Type		Source Data Value				Target Data Size<> Source Data Size	
	value changes sign	value truncates	value enlarges	signed	unsigned	signed	unsigned	positive	negative	larger than max datatype2	sensitive	smaller	larger
CWE													
194 - Unexpected Sign Extension	√		√	√			√		√				
195 - Signed to Unsigned Conversion Error	√		√	√			√		√				√
196 - Unsigned to Signed Conversion Error	√	√			√	√		√		√			
197 - Numeric Truncation Error		√								√		√	
681 - Incorrect Conversion between Numeric Types	√										√	√	
704 - Incorrect Type Conversion or Cast	√	√	√										

Now we can use variations and parameters to identify gaps in existing CWEs



Further generalization (description of a larger family of computations)

Unsafe Type Conversion

common foothold	common generalized condition	
cast of DE1 of data type T1 to datatype T2	T1,T2, and value of DE1 results in change to value of DEI	CWE 194 CWE 195 CWE 196 CWE 197 CWE 681 CWE 704
common injury <i>loss of data in use, Loss of availability of service</i>	<i>because under certain circumstances the cast operation violates a naive assumption that the value remains unchanged;</i>	

Other computations that violate naive assumptions about the resulting value (SFPs are numbered as per Phase I result)

SFP Wrap around error

SFP Incorrect pointer scaling

SFP Use of uninitialized variable

SFP Divide by zero

SFP Suspicious condition

SFP Incorrect parameters to an API

SFP Incorrect operation of Non-Serializable Object

SFP Faulty pointer use

SFP Faulty pointer creation

Family: “Identifiable glitch in computation” SFP-1

common foothold	common generalized condition
identifiable operation that under certain circumstances results in unexpected change of data	data is inappropriate for the operation

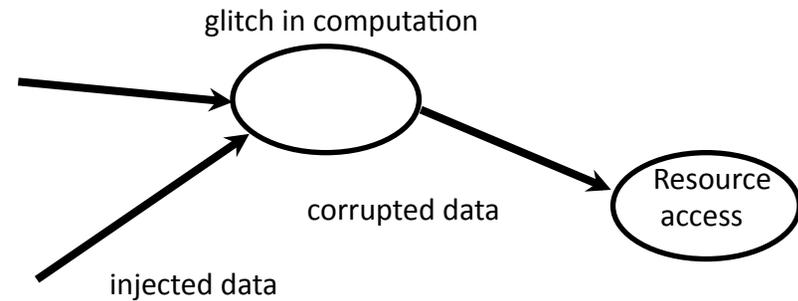
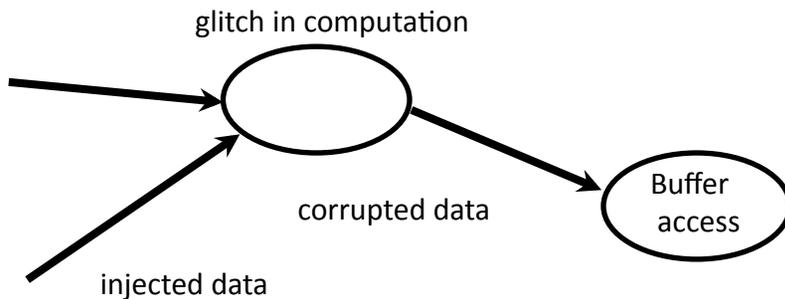
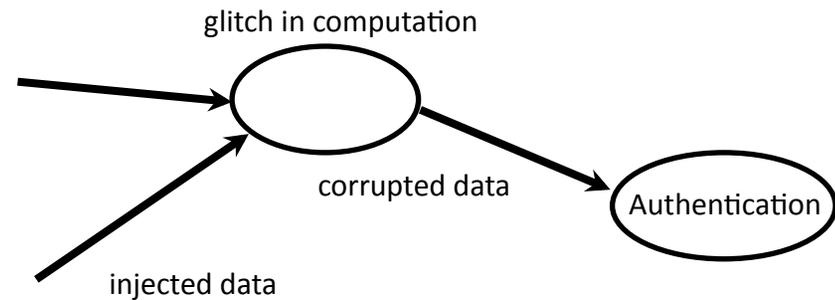
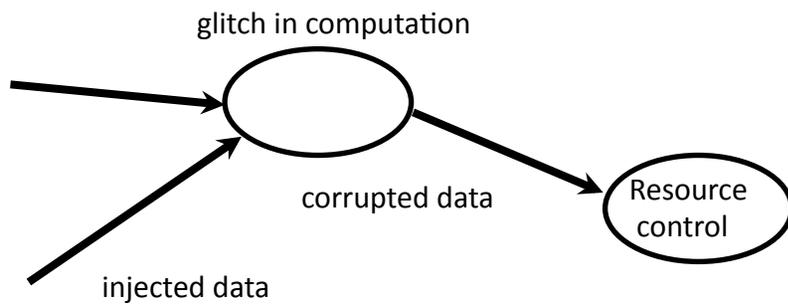
common parameters:

- operation (syntactic pattern)
- type of data (integer, boolean, etc.)
- what condition of data leads to a glitch
- type of glitch (how does the value change, e.g. overflow, underflow, loss, exception, etc.)



Overcoming Usual Difficulties in Categorization

Vulnerabilities that are compositions of several elementary “shapes”



incorrect pointer scaling -> faulty pointer use

incorrect buffer length computation -> faulty buffer access



SFP-8 Faulty Buffer Access

SFP8

Faulty Buffer Access

A weakness where the code path has all of the following:

- an end statement that performs a Buffer Access Operation and where exactly one of the following is true:

- the access position of the Buffer Access Operation is outside of the buffer or

- the access position of the Buffer Access Operation is inside the buffer and the size of the data being accessed is greater than the remaining size of the buffer at the access position

Where Buffer Access Operation is a statement that performs access to a data item of a certain size at access position. The access position of a Buffer Access Operation is related to a certain buffer and can be either inside the buffer or outside of the buffer.



SFP-8 Parameters and CWE mapping

Parameters	Buffer location			Access kind		Access position in relation to the buffer		Access position defined by (this parameter is not necessary)	
	heap	stack	data segment	write	read	inside the buffer	outside the buffer	Array with index	pointer
Values									
CWE									
118 - Improper Access of Indexable Resource								√	
119 - Failure to Constrain Operations within the boundaries of a memory buffer									
121 - Stack Overflow		√		√		√			
122: Heap Overflow	√			√		√			
123: Write-what-where Condition				√					
124: Buffer Under-write				√			√		
125: Out-of-bounds read					√				
126: Buffer Over-read					√	√	√		
127: Buffer Under-read					√				
129: Unchecked array indexing								√	
120 - Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')				√		√			



Improved Reporting Based on Injury

Parameter S	Buffer			Access		Access Position contained		Access Position is defined by	
	Heap	Stack	Data segment	write	read	In the buffer	Outside the buffer	Array with index	pointer
Priority									
P1		✓		✓		any		any	
P2	✓			✓		any		any	
P3	any				✓	any		any	

Priority reporting is based on parameters and can be structured around vectors of attack and impact



SFP: WHERE WE ARE AND WHAT NEXT



- 21 primary clusters
 - Cover 630 CWEs
- 62 secondary clusters
 - Contain both discernable as well as non-discernable CWEs
- 36 software fault patterns
 - Cover 310 discernable CWEs
 - Each SFP has
 - Foot-hold
 - Conditions
 - Parameters
 - Sample values of parameters
 - Injuries
 - CWE mapping



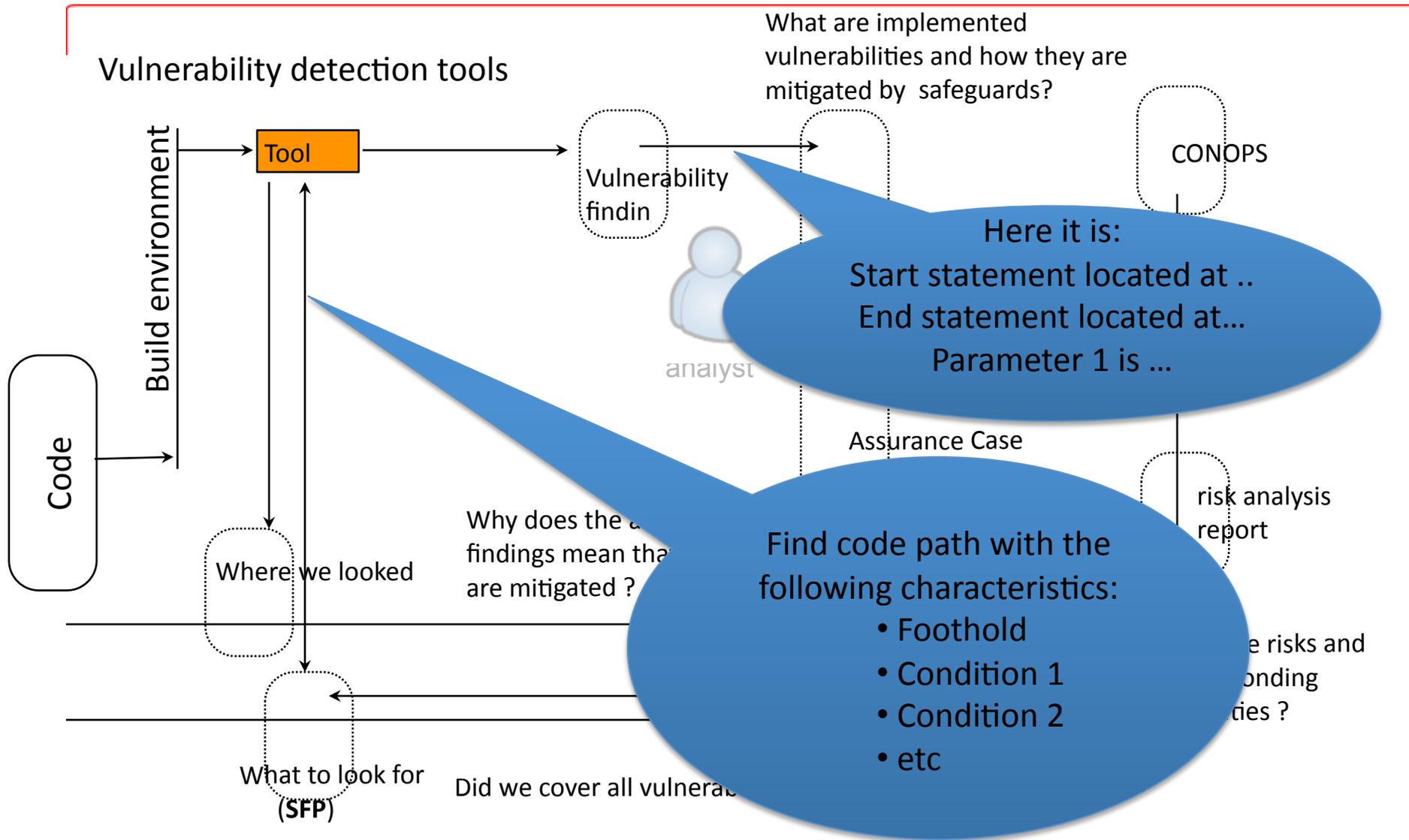
- There are non-discernable CWEs
 - Ill-defined code weaknesses
 - Design weaknesses
 - Architecture weaknesses
 - etc.
- Full formalization of SFPs
- More parameter values
- Address gaps in CWEs



***SFP DEFINES AN INTERFACE TO
AUTOMATED DETECTION TOOLS***



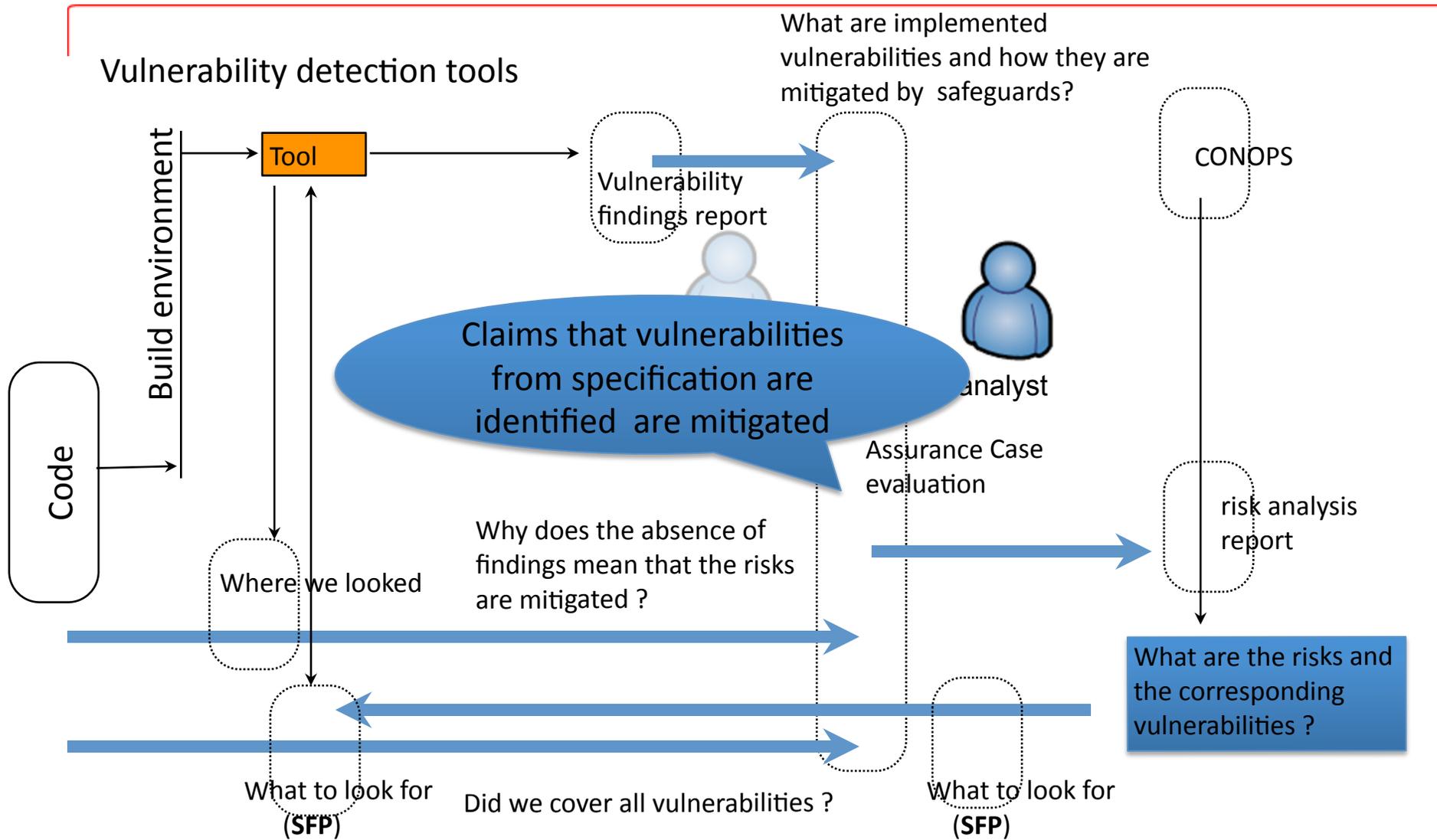
SFP defines an interface to the detection tool



SFP AND CLAIMS TO ASSURANCE CASE



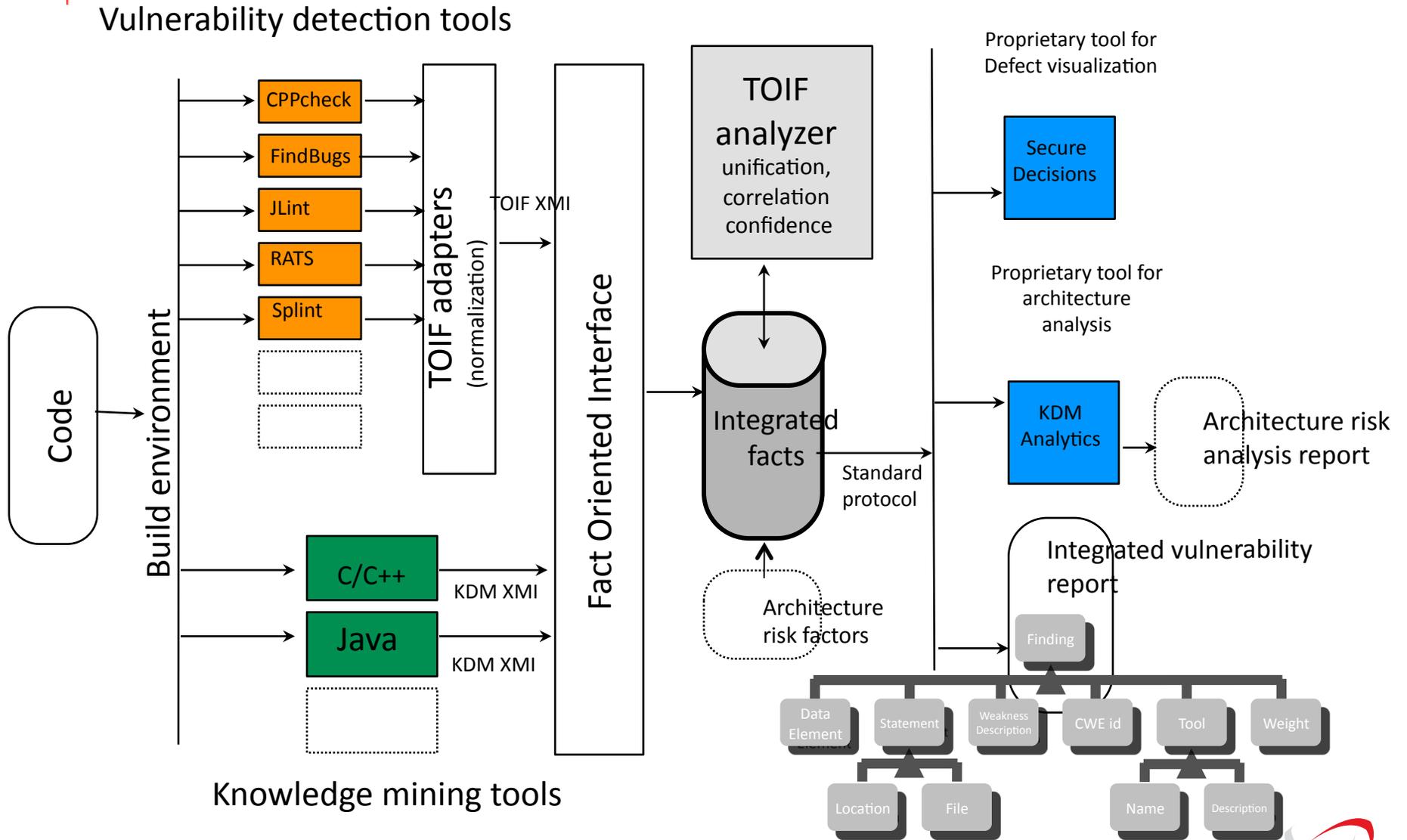
Mathematical reasoning about claims



SFP AND INTEGRATION OF EXISTING TOOLS



DHS TOIF Architecture



SUMMARY



- Existing classifications (Landwehr, CWE, etc.) lack some key considerations
 - They do not restrict the features that are input to categories
 - They do not focus on the features that are identifiable in artifacts, like code
 - They do not consider normalization of feature descriptions
 - They are not aligned with injury
 - They do not consider common vocabulary for feature descriptions
- Benefits of SFP approach
 - Manageable catalog with a small number of categories
 - Normalization allows comparison, generalization, etc.
 - Aligned with injury - easy to report and manage;
 - Static analysis contributes to traditional risk analysis and system/mission assurance
 - Helps identify gaps



Benefits of Parameterized SFPs

- SFPs extend the CWE catalog into a specification
- SFPs allow mathematical reasoning about vulnerabilities
- SFPs make analysis systematic
- SFPs facilitate management of findings
- SFPs facilitate interface between stakeholders and static analysis tool vendors

