Working Together to Build Confidence

Software Fault Patterns:

Towards Formal Compliance Points for CWE

Dr. Nikolai Mansourov
CTO

What is a formal compliance point ?

- Example: "Chair"
  - "Chair is a piece of furniture that has a square horizontal surface, four legs and a backrest and that is used for sitting down".
    - General concept: "piece of furniture"
    - Characteristics:
      - Has a leg
      - Has a surface that is horizontal
      - Has a surface that is square
      - Is used for sitting down
  - Characteristics are used to discern individual things and for making unambiguous statements
    - A sofa is not a chair
    - A table is not a chair
    - A conference chair is not a chair
  - Really, a well-defined "bin"

chair(X):-
∃pieceOfFurniture(X) &
hasLegs(X,4) &
∃surface(Y) &
hasSurface(X,Y) &
isSquare(Y) &
isHorizontal(Y),
isUsedForSittingDown(X)
.

A chair shall have 4 legs, …

Why do we need formal compliance points for CWEs ?

NIST SAMATE SRD ID=14

CWE 121 Stack- based Buffer Overflow

```
/* Stack Overflow */
#define BUFSIZE 256
int main(int argc, char **argv) {
char buf[BUFSIZE];
strcpy(buf, argv[1]);
}
```

NIST SAMATE SRD ID=866

CWE 251 Often Misused: String Management

```
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 10

int main(int argc, char *argv[])
{
        const char myLongString[] = "This is a long string...";
        char str[MAX_SIZE];
        // Often Misused String Management:
        // Buffer overflow with strcpy function
        strcpy(str, myLongString);
        return 0;
}
```

Often Misused: String Management

Category ID: 251 *(Category)* **Status:** Incomplete

Description

Description Summary

Functions that manipulate strings encourage buffer overflows.

Applicable Platforms

Languages

C

C++

Demonstrative Examples

Example 1

Windows provides the _mbs family of functions to perform various operations on multibyte strings. When these functions are passed a malformed multibyte string, such as a string containing a valid leading byte followed by a single null byte, they can read or write past the end of the string buffer causing a buffer overflow. The following functions all pose a risk of buffer overflow: _mbsinc _mbsdec _mbsncat _mbsncpy _mbsnextc _mbsnset _mbsrev _mbsset _mbsstr _mbstok _mbccpy _mbslen

Stack-based Buffer Overflow

Weakness ID: 121 *(Weakness Variant)* **Status:** Draft

Description

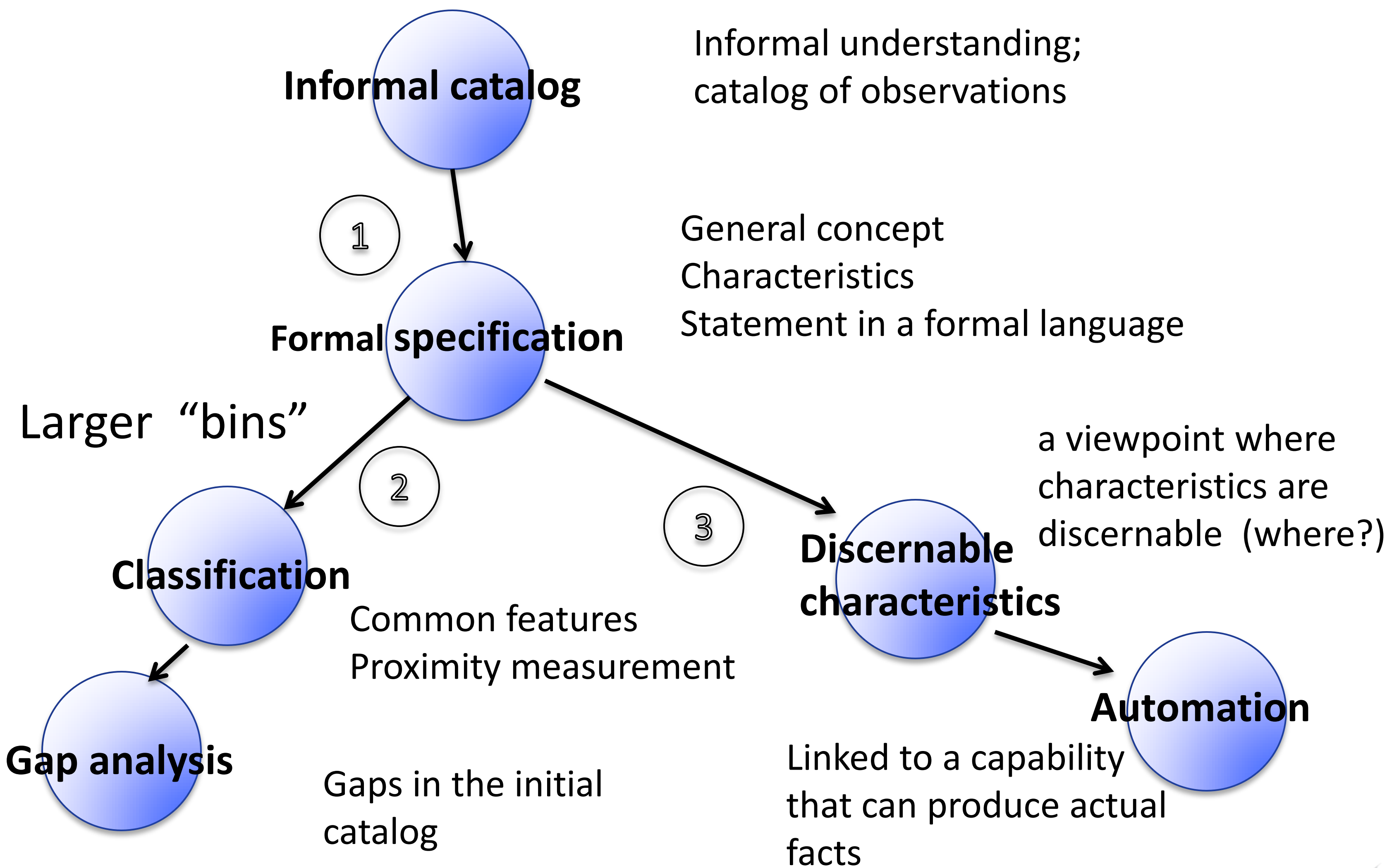
Description Summary

A stack-based buffer overflow condition is a condition where the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function).

Alternate Terms

Stack Overflow: "Stack Overflow" is often used to mean the same thing as stack-based buffer overflow, however it is also used on occasion to mean stack exhaustion, usually a result from an excessively recursive function call. Due to the ambiguity of the term, use of stack overflow to describe either circumstance is discouraged.

Formal compliance, larger bins, and maybe automation



Software Fault Pattern (SFP) Research Program

- Develop a *formal specification* of software weaknesses/vulnerabilities that enables automation
  - Focus on characteristics that are discernable in *code*
  - Focus on *computation* as the viewpoint that can support automation
    - Computation causes observable events, and
    - Certain "observable" code constructs are characteristics of computations
  - "Larger bins" for weaknesses
  - 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)
  - Enables *mathematical* reasoning about vulnerability findings

What is a Software Fault Pattern (SFP)?

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

SFP approach: transforming CWEs into a formal specification

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.

Cluster: Memory Management

SFP formalization approach uses restricted *natural language* on top of a logical model

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 optional)	
Values	heap	stack	data segment	write	read	inside the buffer	outside the buffer	Array with index	pointer
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')				√		√			

Security Injuries associated with SFP8 parameters

- Loss of availability of service (write access);
- Subversion of service (especially "bulk" write access, where the buffer is located in the stack);
- Loss of integrity of service (write access);
- Loss of integrity of data (write access);
- Loss of confidentiality (read access);

Alignment with security injuries facilitates use of SFPs for risk analysis

Improved Reporting Based on Injury

Parameters / Priority	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
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

How does the new approach enables automation?

1 Common, agreed upon ***vocabulary*** for systems elements:

Pipework element

Pipe, Valve, Pump, Gauge, Meter, T-connector

Pipe *is connected to* pipework element

2 Normalized mathematical ***description*** of a given system is based on the vocabulary:

Valve1 *is connected to* pipe2;

pipe2 *is connected to* meter3;

Pump4 *is connected to* pipe5 and pipe6; etc.

3 ***Capability*** to produce mathematical descriptions

4 Software Fault ***Pattern*** description is based on the system vocabulary

this makes all characteristics *discernable*

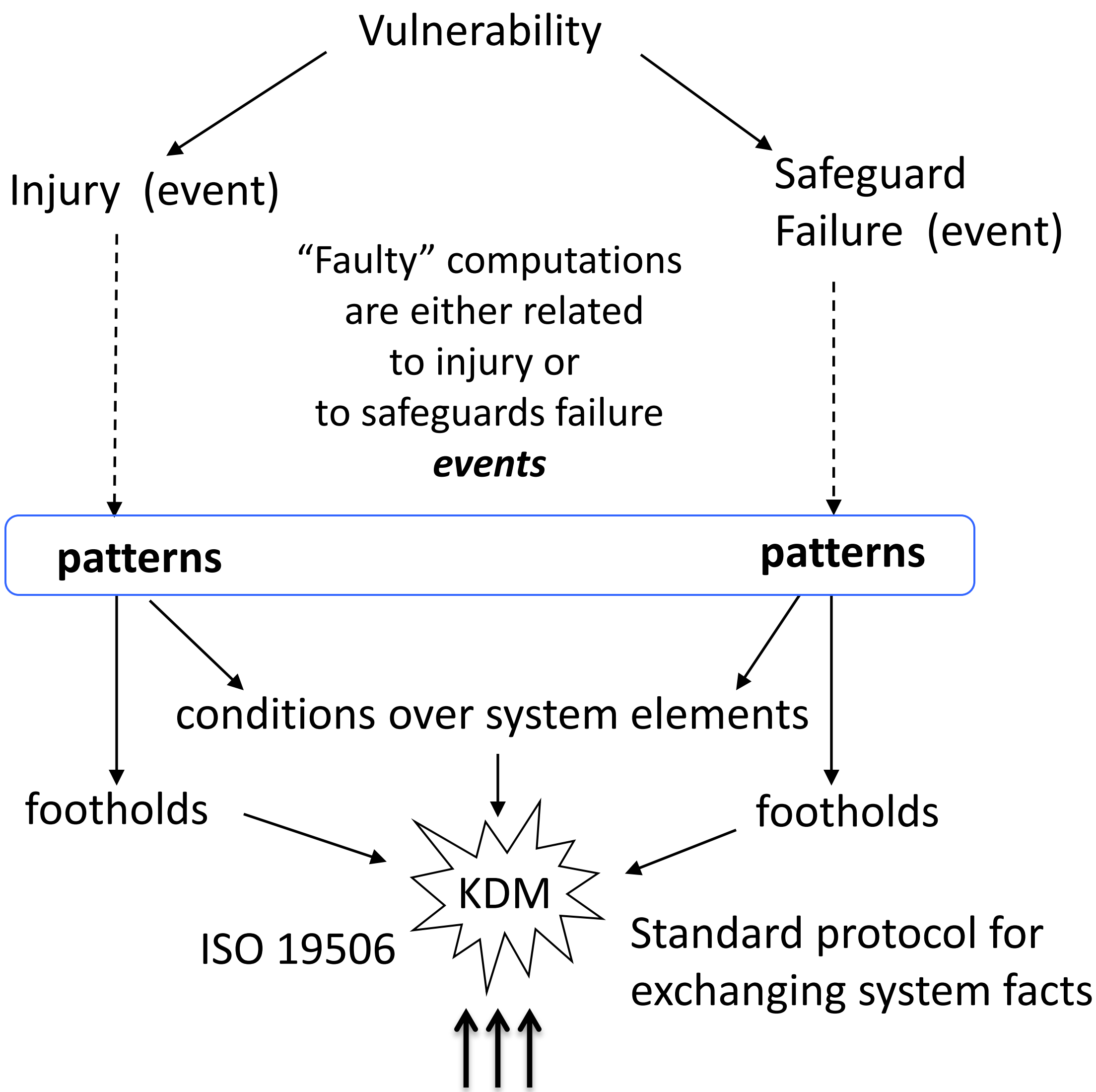
this enables information *interexchange*

allows mathematical reasoning about findings

allows mathematical reasoning about assurance

5 ***Capability*** to mine patterns (evidence collection)

Machine-consumable vulnerability patterns



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

Vulnerability implies a **failure event**

Foot-hold: a "known" construct in the system's artifacts that is ***necessary*** for the fault event to occur

SFP-8 Faulty Buffer Access

foothold

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.

condition

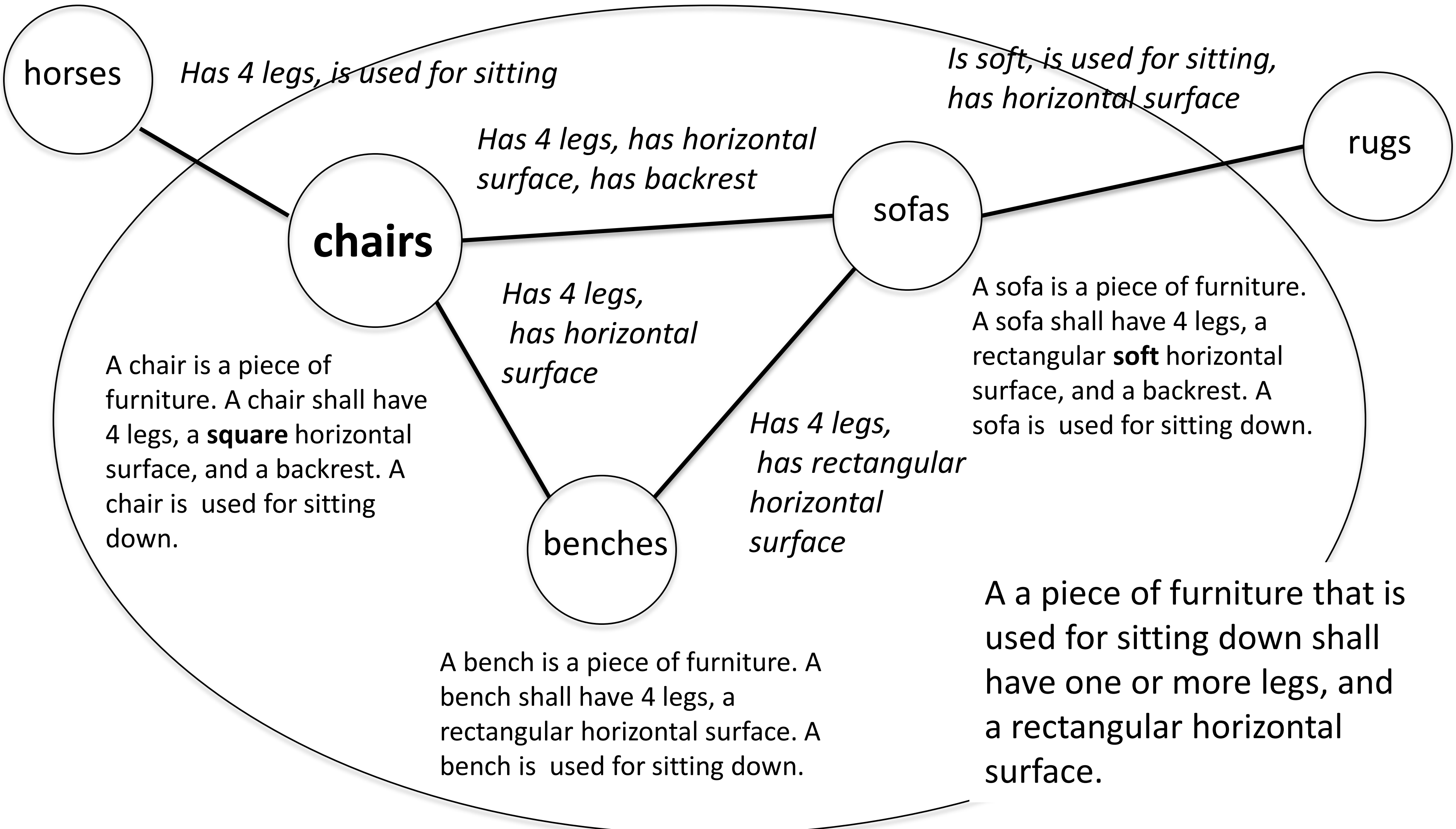
Unique Foothold is essential for both classification and automation

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)

Foothold and Injury create clusters of vulnerabilities

What about classification and “larger bins” ?



Methodology for Defining SFPs: "Larger Bins"

Clusters define few manageable vulnerability families

4. Clusters

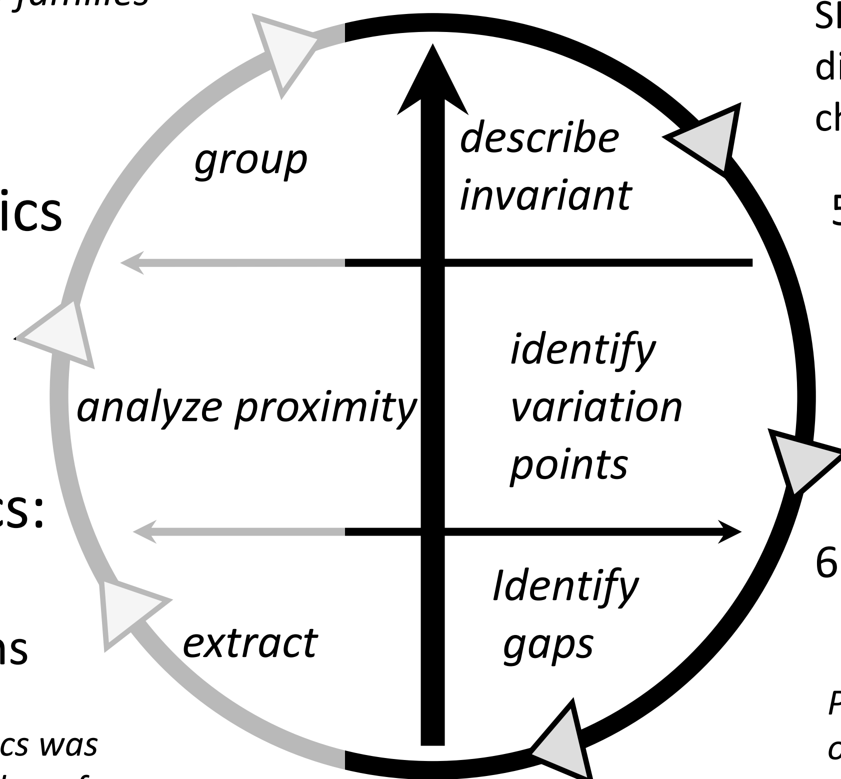
SFP focuses on discernable characteristics

3. Generalized characteristics

5. SFP

based on the use of common noun concepts and injury

Parameterization is a step toward full formalization



2. Characteristics: footholds & conditions of computations

6. Parameters

Initial analysis of characteristics was informal because a large number of non-discernable CWEs was anticipated

Parameterization is only done for discernable CWEs

1. CWEs *used as observation samples*

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 f*oot-holds* of the computation
    - Shared vocabulary for fact collection and vulnerability definition
  - Alignment with injury (defined in NIST SCAP CVSS)

Extracting and Generalizing SFP Characteristics

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.

Characteristics 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

CWE 194

- *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

CWE 195

- *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

CWE 196

- *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

CWE 197

- *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

CWE 681

- *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

CWE 704

- *computation involves data element DE1 of data type T1*
- *computation involves cast of DE1 to data type T2*
- cast changes value

invariant characteristics

common foothold

cast of DE1 of data type T1 to datatype T2

common (generalized)condition

T1, T2 and the value of DEI result in change of value of DE1

common injury

loss of data in use

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 DEI results in change to value of DEI
common injury *loss of data in use*	*because under certain circumstances the cast operation violates a naive assumption that the value remains unchanged;*

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)

- data type T1 is signed
- data type T1 is unsigned

datatype T2 (target)

- data type T2 is signed
- data type T2 is unsigned

relation between T1 and T2

- data type T1 is larger than data type T2
- data type T1 is smaller than data type T2

data element DE1 (input)

- 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	
sample values	value changes sign	value trun-cates	value enlarges	signed	unsigned	signed	unsigned	positive	negative	larger than max datatype2	sensitive	smaler	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 DEI results in change to value of DEI
common injury *loss of data in use*	*because under certain circumstances the cast operation violates a naive assumption that the value remains unchanged;*

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

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.)

SFP Catalog (1 of 4)

Larger "bins" Smaller"bins" Individual CWEs

Primary	Secondary	# of CWEs	Primary CWE Totals	Pattern & Condition Available?	Discernable CWEs	SFP #
Risky Values			31			
	Glitch in Computation	31		partial	27	SFP1
Unused entities			3			
	Unused entities	3		yes	3	SFP2
API			28			
	Use of an improper API	28		partial	20	SFP3
Exception Management			27			
	Unchecked status condition	17		partial	13	SFP4
	Ambiguous exception type	2		yes	2	SFP5
	Incorrect exception behavior	8		partial	3	SFP6
Memory Access			20			
	Faulty pointer use	3		yes	3	SFP7
	Faulty buffer access	11		yes	11	SFP8
	Faulty string expansion	2		yes	2	SFP9
	Incorrect buffer length computation	3		partial	2	SFP10
	Improper NULL termination	1		singular	1	SFP11
Memory Management			5			
	Faulty memory release	5		yes	5	SFP12
Resource Management			17			
	Unrestricted consumption	4		partial	3	SFP13
	Failure to release resource	7		yes	7	SFP14
	Faulty resource use	2		yes	2	SFP15
	Life cycle	4		no	0	-

Automatable"bins"

Cluster: Memory Access: SFP 7 Faulty Pointer Use

Faulty Pointer Use
A weakness where the code path has all of the following:
- an end statement that performs use of pointer with NULL or "out of range" value
Where a “out of range” is defined as access to memory chunk through exactly one of the following:
-- faulty address obtained as a subtraction of two pointers to different memory chunks or
-- faulty type such as use of a pointer to access a structure element where the pointer was cast from a data item that is not of a structure datatype.

Parameters	the end statement that performs use of pointer			incorrect pointer value for identified end statements			
Sample Values	pointer dereference	pointer subtraction	pointer cast	NULL	out of range: faulty address	out of range: faulty type	buffer de allocated
CWE							
476 - NULL Pointer Dereference	√			√			
469 - Use of Pointer Subtraction to Determine Size		√			√		
588 - Attempt to Access Child of a Non-structured Pointer	√		√			√	

SFP Catalog (2 of 4)

Path Resolution			51			
	Path traversal	43		partial	38	SFP16
	Failed chroot jail	1		singular	1	SFP17
	Link in resource name resolution	7		partial	4	SFP18
Synchronization			22			
	Missing lock	13		partial	10	SFP19
	Race condition window	5		partial	4	SFP20
	Multiple locks/unlocks	3		yes	3	SFP21
	Unrestricted lock	1		singular	1	SFP22
Information Leak			96			
	Exposed data	76		partial	38	SFP23
	State disclosure	7		no	0	-
	Exposure through temporary file	3		no	0	-
	Other exposures	7		no	0	-
	Insecure session management	3		no	0	-
Tainted Input			138			
	Tainted input to command	87		partial	68	SFP24
	Tainted input to variable	8		yes	8	SFP25
	Composite tainted input	0		no	0	SFP26
	Faulty input transformation	15		no	0	-
	Incorrect input handling	17		no	0	-
	Tainted input to environment	11		partial	3	SFP27

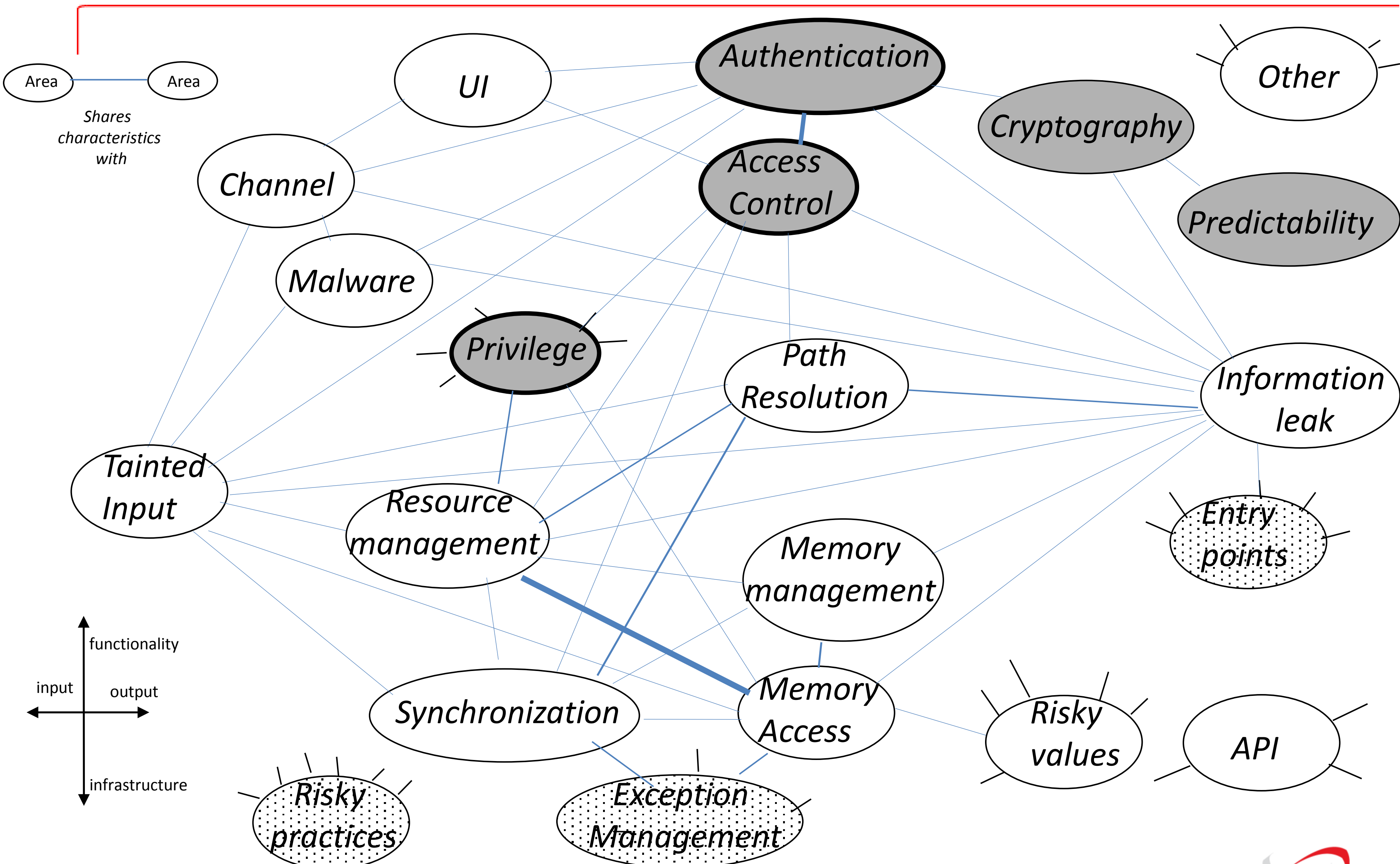
SFP Catalog (3 of 4)

Entry Points			11			
	Unexpected access points	11		yes	11	SFP28
Authentication			43			
	Authentication bypass	10		no	0	-
	Faulty endpoint authentication	11		partial	6	SFP29
	Missing endpoint authentication	2		yes	2	SFP30
	Digital certificate	6		no	0	-
	Missing authentication	2		yes	2	SFP31
	Insecure authentication policy	6		no	0	-
	Multiple binds to the same port	1		singular	1	SFP32
	Hardcoded sensitive data	4		partial	2	SFP33
	Unrestricted authentication	1		singular	1	SFP34
Access Control			16			
	Insecure resource access	4		partial	2	SFP35
	Insecure resource permissions	7		no	0	-
	Access management	5		no	0	-
Privilege			12			
	Privilege	12		partial	1	SFP36

SFP Catalog (4 of 4)

Channel			13			
	Channel Attack	8		no	0	-
	Protocol error	5		no	0	-
Cryptography			13			
	Broken cryptography	5		no	0	-
	Weak cryptography	8		no	0	-
Malware			11			
	Malicious code	8		no	0	-
	Covert channel	3		no	0	-
Predictability			15			
	Predictability	15		no	0	-
UI			14			
	Feature	7		no	0	-
	Information loss	4		no	0	-
	Security	3		no	0	-
Other			46			
	Architecture	11		no	0	-
	Design	29		no	0	-
	Implementation	5		no	0	-
	Compiler	1		no	0	-
TOTAL			**632**		310	36

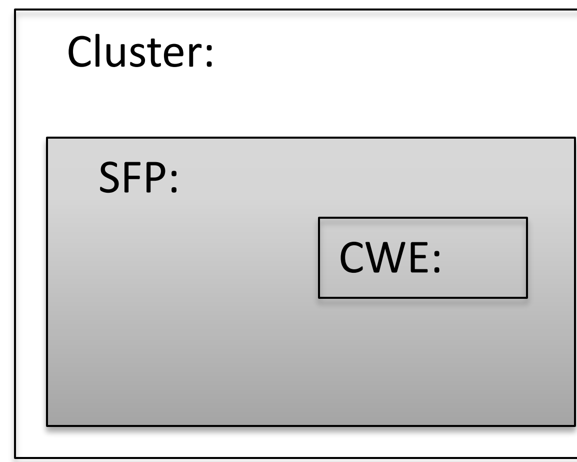
21 clusters and their associations



SFP SUMMARY

- 21 primary clusters (large “bins” but still well-defined)
  - Cover 632 CWEs
- 62 secondary clusters
  - Contain both discerable 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

SFP catalog



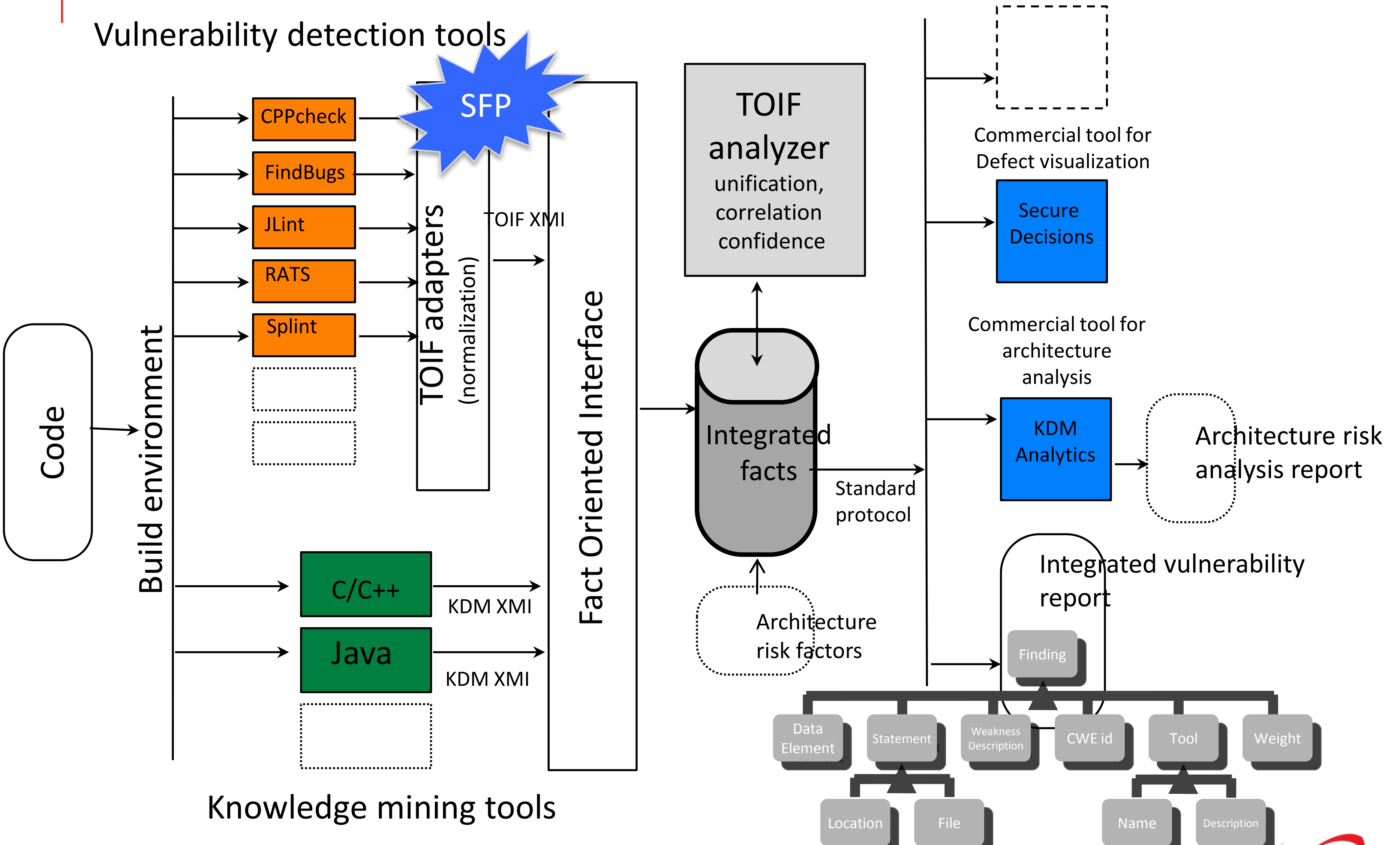
SFP BENEFITS

Benefits of SFP specification

- Specify assurance claims related to families of faults aligned with risk assessment process
- Automatically obtain evidence by using tools that support SFP-CWE specification
- Access compliance of weakness detection tools and their coverage of CWE (coverage claims)
- Provide broader CWE coverage

CURRENT AND FUTURE USES OF SFP

Open Source TOIF Architecture



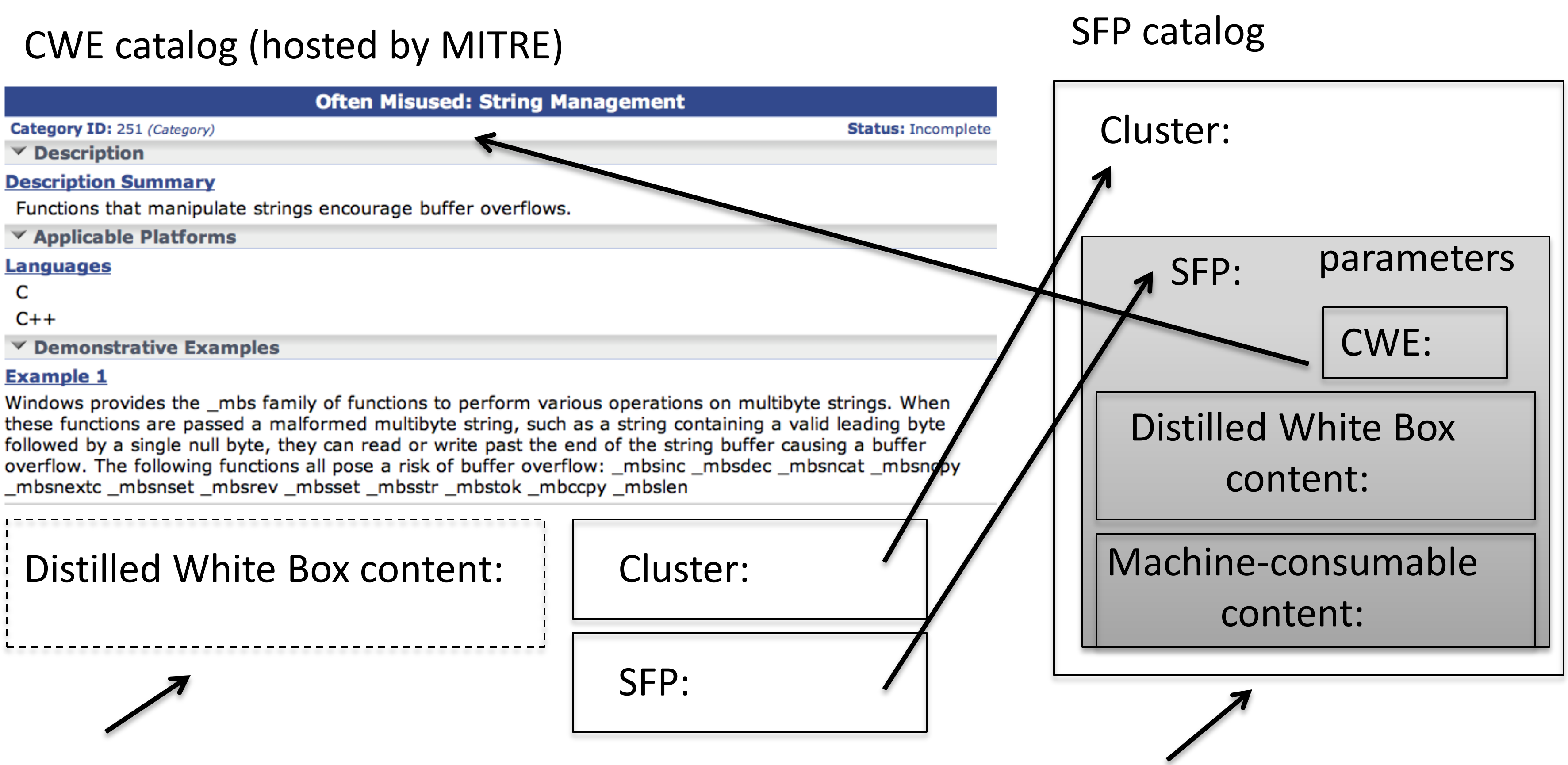
Standardization of SFPs

- Standardization track:
  - Object Management Group (OMG)
  - Then ISO/IEC
    - SFP leverages ISO 19506 Knowledge Discovery Metamodel, developed by OMG
- Technical process:
  - SFP Metamodel, describing components of SFP and their relations
    - Use OMG standards
    - Defines interchange format SFP XMI
  - Interface to static analysis tools
  - Interface to software platform/parameters
  - Catalog of SFPs in machine-consumable format
    - Clusters, footholds, conditions

DISCUSSION

- Expand the formalization approach (incl. to other areas of Software Assurance)
  - There are non-discernable CWEs
    - Ill-defined code weaknesses
    - Design weaknesses
    - Architecture weaknesses
    - etc.
  - More parameter values
  - Address gaps in CWEs
- Full formalization of SFPs
- Formalization of security policies/compliance

- Accumulate and share machine-readable CWE patterns based on the SFP approach



SFPs : going forward

- Adoption of SFPs in various software assurance contexts
  - SFPs for Coverage and Claims Representation (CCR)
    - SFP clusters are formally defined “bins” to make claims against
    - They are hierarchically arranged and link to individual CWEs
    - Relations between various “bins” are formally defined
    - SFPs are already aligned with security injuries/risk analysis
  - Claims (near term):
    - Cluster Memory Management: SFP-8 Faulty Buffer Access: CWE 121
  - Claims (future):
    - Cluster Memory Management: SFP-8 Faulty Buffer Access: CWE 121:{full,partial}
    - Cluster Memory Management: SFP-8 Faulty Buffer Access {partial CWE121, partial CWE122, adding XXX}
    - Cluster Memory Management {partial SFP-8, partial SFP-9, adding xxx}