Vulnerability Management in Security By Design context

Project: Seamless Security-by-Design (SsbD)

Nan Messe

with

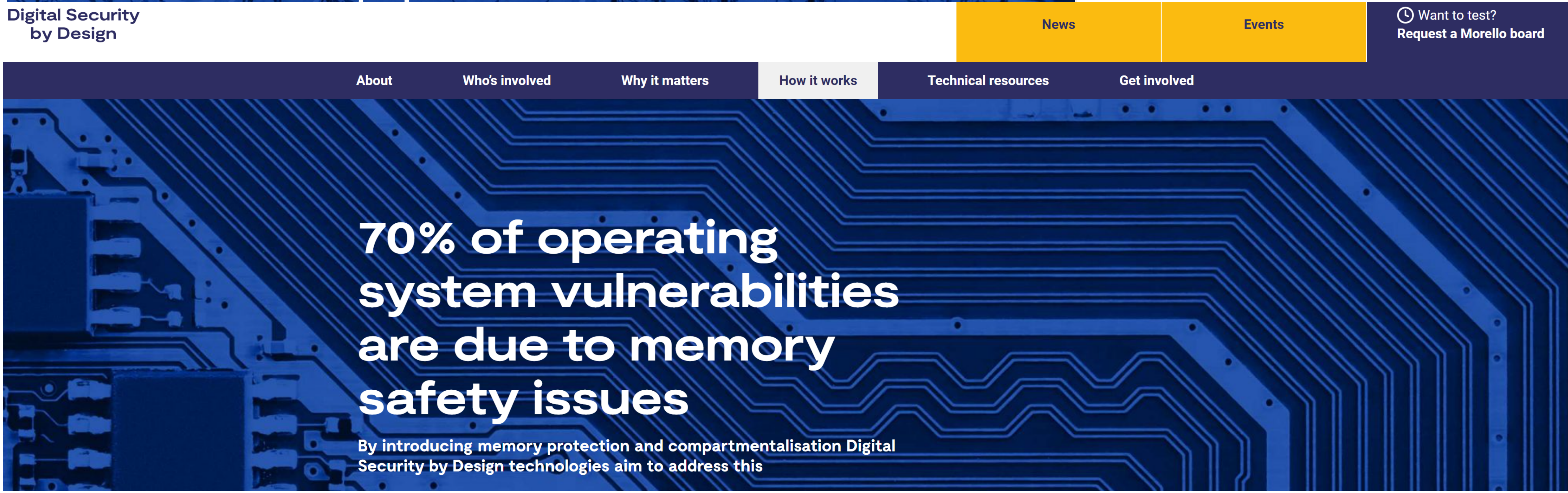
Avi Shaked (University of Oxford)

Tom Melham (University of Oxford)

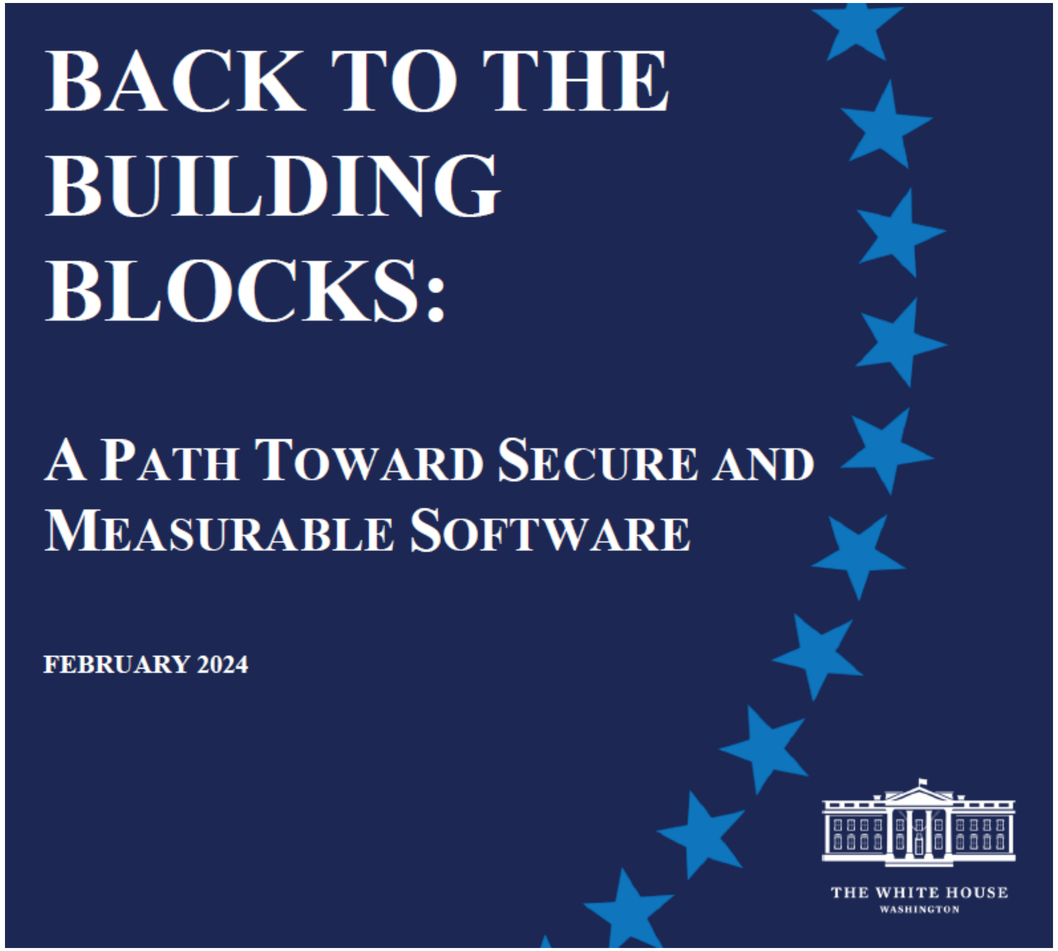
Our Problem: DSbD Value Proposition

Source: https://www.dsbd.tech/

Our Problem: DSbD Value Proposition



The Vulnerability Problem



"… the fact that mitigating known software vulnerabilities is a complex systems problem and the current ecosystem does not sufficiently incentivize the investments required to secure the foundations of cyberspace."

"A **proactive** approach that focuses on eliminating entire classes of vulnerabilities reduces the potential attack surface and results in more reliable code, less downtime, and more predictable systems."

Vulnerability Management?

National Cyber Security Centre

ABOUT NCSC CISP REPORT AN INCIDENT CONTACT US

Home Information for... Advice & guidance Education & skills Products & services News, blogs, events...

GUIDANCE

Vulnerability management

This area provides advice, guidance and other resources aimed specifically at those with an interest in vulnerability management.

Pages

PAGE 1 OF 8

iStock.com/bagotaj

All systems contain vulnerabilities. They may take the form of a configuration issue for system administrators to resolve, software defects requiring a vendor update, or even a vulnerability which the vendor doesn't yet know exists, for which a mitigation isn't available.

This makes vulnerability management a critical control for organisations.

An effective vulnerability management process allows your organisation to understand, and validate on a regular basis, which vulnerabilities are present in your technical estate, where updates are failing, and to actively reduce the impact of both. It also allows you to react quickly when a critical vulnerability is disclosed, by helping you understand your organisation's exposure to it.

Vulnerability Management?

National Cyber Security Centre

ABOUT NCSC | CISP | REPORT AN INCIDENT | CONTACT US

Home | Information for... | Advice & guidance | Education & skills | Products & services | News, blogs, events...

GUIDANCE

Vulnerability management

This area provides advice, guidance and other resources aimed specifically at those with an interest in vulnerability management.

Pages

PAGE 1 OF 8

iStock.com/bagotaj

All systems contain vulnerabilities. They may take the form of a configuration issue for system administrators to resolve, software defects requiring a vendor update, or even a vulnerability which the vendor doesn't yet know exists, for which a mitigation isn't available.

This makes vulnerability management a critical control for organisations.

An effective vulnerability management process allows your organisation to understand, and validate on a regular basis, which vulnerabilities are present in your technical estate, where updates are failing, and to actively reduce the impact of both. It also allows you to react quickly when a critical vulnerability is disclosed, by helping you understand your organisation's exposure to it.

OVERVIEW: VULNERABLE BY DESIGN

SECURE BY DESIGN

Revision Date: October 25, 2023

Technology is integrated into nearly every facet of daily life, as internet facing systems increasingly connect us to critical systems that directly impact our economic prosperity, livelihoods, and even health, ranging from personal identity management to medical care. One example of the disadvantage of such conveniences are the global cyber breaches resulting in hospitals canceling surgeries and diverting patient care. Insecure technology and vulnerabilities in critical systems may invite malicious cyber intrusions, leading to potential safety[1] risks.

As a result, it is crucial for software manufacturers to make secure by design and secure by default the focal points of product design and development processes. Some vendors have made great strides driving the industry forward in software assurance, while others continue to lag behind. The authoring organizations strongly encourage every technology manufacturer to build their products based on reducing the burden of cybersecurity on customers, including preventing them from having to constantly perform monitoring, routine updates, and damage control on their systems to mitigate cyber intrusions. We also urge the software manufacturers to build their products in a way that facilitates automation of configuration, monitoring, and routine updates. Manufacturers are encouraged to take ownership of improving the security outcomes of their customers. Historically, software manufacturers have relied on fixing vulnerabilities found after the customers have deployed the products, requiring the customers to apply those patches at their own expense. Only by incorporating secure by design practices will we break the vicious cycle of constantly creating and applying fixes. **Note:** The term "secure by design" encompasses both secure by design and secure by default.

To accomplish this high standard of software security, the authoring organizations encourage manufacturers to prioritize the integration of product security as a critical prerequisite to features and speed to market. Over time, engineering teams will be able to establish a new steady-state rhythm where security is truly designed-in and takes less effort to maintain.

Reflecting this perspective, the European Union reinforces the importance of product security in the Cyber Resilience Act, emphasizing that manufacturers should implement security throughout a product's life-cycle in order to prevent manufacturers from introducing vulnerable products into the market.

ICO
Institut Cybersécurité Occitanie

Vulnerability Management?

National Cyber Security Centre

ABOUT NCSC | CISP | REPORT AN INCIDENT | CONTACT US

Home | Information for... | Advice & guidance | Education & skills | Products & services | News, blogs, events...

GUIDANCE

Vulnerability management

This area provides advice, guidance and other resources aimed specifically at those with an interest in vulnerability management.

Pages

- **Vulnerability management**
- Guidance
  - 1. Put in place a policy to update by default
  - 2. Identify your assets
  - 3. Carry out assessments by triaging and prioritising
  - 4. The organisation must own the risks of not updating
  - 5. Verify and regularly review your vulnerability management process
- Understanding vulnerabilities

PAGE 1 OF 8

Patching oriented

Zero guidance regarding design

iStock.com/bagotaj

All systems contain vulnerabilities. They may take the form of a configuration issue for system administrators to resolve, software defects requiring a vendor update, or even a vulnerability which the vendor doesn't yet know exists, for which a mitigation isn't available.

This makes vulnerability management a critical control for organisations.

An effective vulnerability management process allows your organisation to understand, and validate on a regular basis, which vulnerabilities are present in your technical estate, where updates are failing, and to actively reduce the impact of both. It also allows you to react quickly when a critical vulnerability is disclosed, by helping you understand your organisation's exposure to it.

OVERVIEW: VULNERABLE BY DESIGN

SECURE BY DESIGN

Revision Date: October 25, 2023

Technology is integrated into nearly every facet of daily life, as internet facing systems increasingly connect us to critical systems that directly impact our economic prosperity, livelihoods, and even health, ranging from personal identity management to medical care. One example of the disadvantage of such conveniences are the global cyber breaches resulting in hospital ... technology and vulnerabilities i... ading to potential safety[1] risk...

As a result, it is cruc... nd secure by default the focal ... he vendors have made great str... hile others continue to lag behi... technology manufacturer to bu... curity on customers, includin... itoring, routine updates, an... ions. We also urge the softwa... itates automation of confi... are encouraged to take ... tomers. Historically, softwar... after the customers have de... e patches at their own expens... break the vicious cycle of con... by design" encompasses both ...

To accomplish this ... ions encourage manufac... critical prerequisite to feat... ll be able to establish a new s... takes less effort to maintain.

Reflecting this pers... product security in the Cybe... mplement security throughout ... om introducing vulnerable products into the market.

CISA · National Security Agency · Federal Bureau of Investigation · Australian Government Australian Signals Directorate · ASD Australian Signals Directorate ACSC · Federal Office for Information Security · Communications Security Establishment Canadian Centre for Cyber Security · Centre de la sécurité des télécommunications Centre canadien pour la cybersécurité · National Cyber Security Centre part of the GCSB · National Cyber Security Centre Ministry of Justice and Security · National Cyber Security Centre a part of GCHQ · OAS More rights for more people · NISC 内閣サイバーセキュリティセンター · CSA Singapore · certnz · NSM Norwegian National Cyber Security Centre · National Cyber and Information Security Agency · KISA Korea Internet & Security Agency · JPCERT CC®

Vulnerability Management By-Design

3. **Vulnerability management.** Some manufacturers have a vulnerability management program that focuses on patching vulnerabilities discovered internally or externally, and little more. More mature programs incorporate extensive data-driven analysis of vulnerabilities and their root causes, taking steps to systemically eliminate entire classes of vulnerability[3]. They implement formal programs around setting quality planning, quality control, quality improvement, and quality measurement. They view defect management as a business matter, not merely a security matter. These programs are not dissimilar in some ways to quality and safety programs in other industries.

OVERVIEW: VULNERABLE BY DESIGN

Technology is integrated into nearly every facet of daily life, as internet facing systems increasingly connect us to critical systems that directly impact our economic prosperity, livelihoods, and even health, ranging from personal identity management to medical care. One example of the disadvantage of such conveniences are the global cyber breaches resulting in hospitals canceling surgeries and diverting patient care. Insecure technology and vulnerabilities in critical systems may invite malicious cyber intrusions, leading to potential safety[1] r

As a result, it is cr … nd secure
by default the foc … he vendors
have made great … hile others
continue to lag be … technology
manufacturer to b … curity on
customers, incluc … toring,
routine updates, … ions. We
also urge the soft … tates
automation of co … are
encouraged to tal … tomers.
Historically, softw … after the
customers have c … e patches
at their own expe … break the
vicious cycle of c … by design"
encompasses bo

To accomplish thi … ons
encourage manuf … critical
prerequisite to fe … ll be able
to establish a new … takes less
effort to maintain

Reflecting this pe … product
security in the Cy … mplement
security throughout a product's life-cycle in order to prevent manufacturers from introducing vulnerable products into the market.

Source: "Secure By Design – Shifting the balance of cybersecurity risk: principles and approaches for secure by design software", CISA et al., 2023.

Vulnerability Management By-Design

“Eliminating Vulnerabilities” series

CISA's Secure by Design Alert Series highlights the prevalence of widely known and documented vulnerabilities, with available and effective mitigations, that have not been eliminated. The Series urges technology manufacturers to build security into products from the beginning to eliminate classes of vulnerability, or product defects, that impact the safety of their customers.

Alerts are released in response to threat actor activity, but further demonstrate how secure by design software development can help reasonably protect against malicious cyber actors successfully exploiting predictable and well-known vulnerabilities.

MAY 02, 2024
Secure by Design Alert: Eliminating Directory Traversal Vulnerabilities in Software

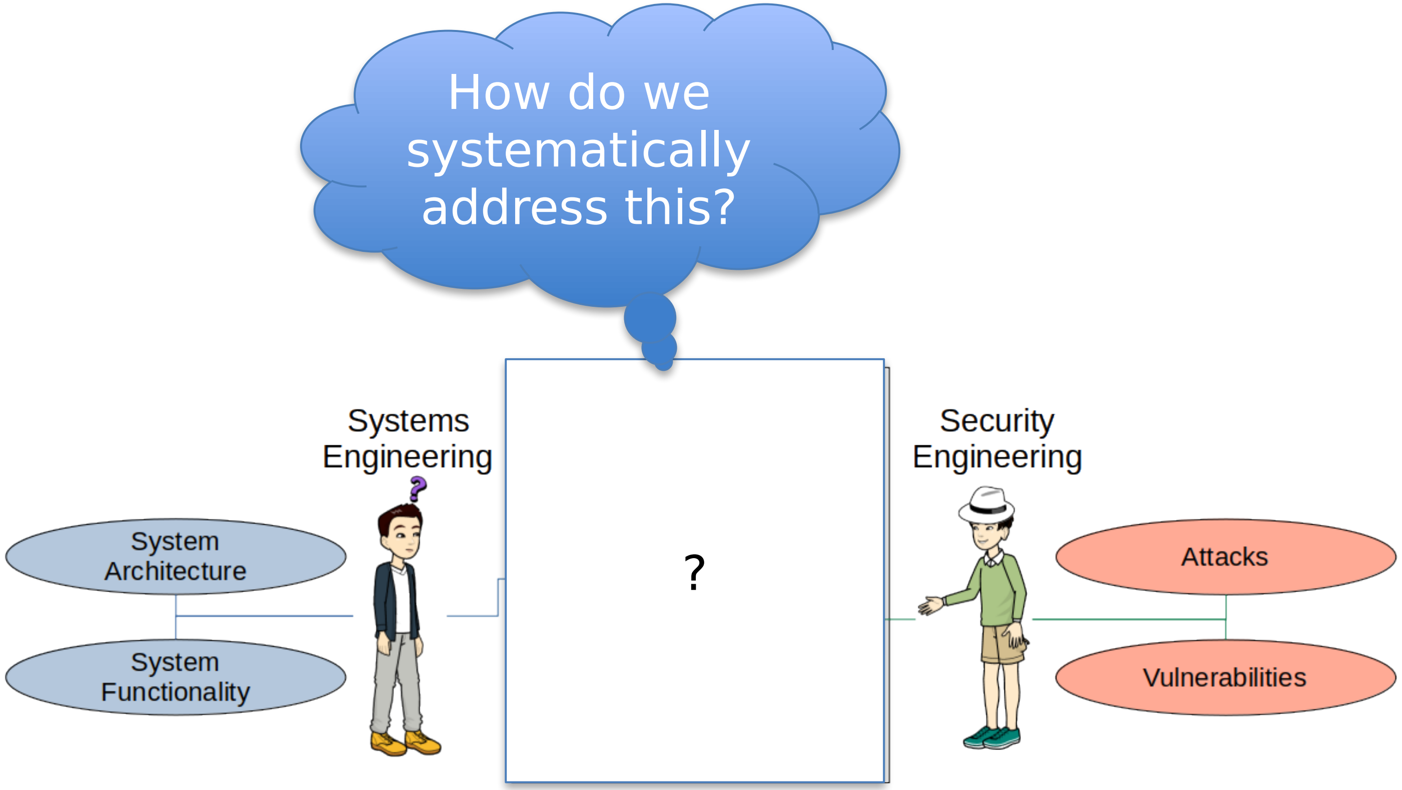
MAR 25, 2024
Secure by Design Alert: Eliminating SQL Injection Vulnerabilities in Software

JAN 31, 2024
Secure by Design Alert: Security Design Improvements for SOHO Device Manufacturers

DEC 15, 2023
Secure by Design Alert: How Manufacturers Can Protect Customers by Eliminating Default Passwords

NOV 29, 2023
Secure by Design Alert: How Software Manufacturers Can Shield Web Management Interfaces From Malicious Cyber Activity

APR 13, 2023 ■ ALERT
Shifting the Balance of Cybersecurity Risk: Security-by-Design and Default Principles



Existing knowledge bases

And the root cause of an attack

Source: https://capec.mitre.org/about/new_to_capec.html

Ontology to the rescue

Common Weakness Enumeration

A community-developed list of SW & HW weaknesses that can become vulnerabilities

New to CWE? Start here!

Home > CWE List > CWE- Individual Dictionary Definition (4.14) ID Lookup: Go

Home | About ▼ | CWE List ▼ | Mapping ▼ | Top-N Lists ▼ | Community ▼ | News ▼ | Search

CWE-787: Out-of-bounds Write

Weakness ID: 787
Vulnerability Mapping: ALLOWED
Abstraction: Base

View customized information: Conceptual | **Operational** | Mapping Friendly | Complete | Custom

▼ Description

The product writes data past the end, or before the beginning, of the intended buffer.

▼ Extended Description

Typically, this can result in corruption of data, a crash, or code execution. The product may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent write operation then produces undefined or unexpected results.

▼ Alternate Terms

Memory Corruption: Often used to describe the consequences of writing to memory outside the bounds of a buffer, or to memory that is invalid, when the root cause is something other than a sequential copy of excessive data from a fixed starting location. This may include issues such as incorrect pointer arithmetic, accessing invalid pointers due to incomplete initialization or memory release, etc.

▼ Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer
ParentOf	V	121	Stack-based Buffer Overflow
ParentOf	V	122	Heap-based Buffer Overflow
ParentOf	B	123	Write-what-where Condition
ParentOf	B	124	Buffer Underwrite ('Buffer Underflow')
CanFollow	B	822	Untrusted Pointer Dereference
CanFollow	B	823	Use of Out-of-range Pointer Offset
CanFollow	B	824	Access of Uninitialized Pointer
CanFollow	B	825	Expired Pointer Dereference

Hierarchy of vulnerabilities

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name
MemberOf	C	1218	Memory Buffer Errors

ICO
Institut Cybersécurité Occitanie

Ontology to the rescue

Common Weakness Enumeration
A community-developed list of SW & HW weaknesses that can become vulnerabilities

Top 25 | Top HW CWE | **New to CWE?** Start here!

Home > CWE List > CWE- Individual Dictionary Definition (4.14) ID Lookup: Go

Home | About ▼ | CWE List ▼ | Mapping ▼ | Top-N Lists ▼ | Community ▼ | News ▼ | Search

CWE-787: Out-of-bounds Write

Weakness ID: 787
Vulnerability Mapping: ALLOWED
Abstraction: Base

▼ Applicable Platforms

Languages

C *(Often Prevalent)*

C++ *(Often Prevalent)*

Class: Assembly *(Undetermined Prevalence)*

Technologies

Class: ICS/OT *(Often Prevalent)*

▶ Demonstrative Examples

▼ Observed Examples

Reference	Description
CVE-2021-21220	Chain: insufficient input validation (CWE-20) in browser allows heap corruption (CWE-787), as exploited in the wild per CISA KEV.
CVE-2021-28664	GPU kernel driver allows memory corruption because a user can obtain read/write access to read-only pages, as exploited in the wild per CISA KEV.
CVE-2020-17087	Chain: integer truncation (CWE-197) causes small buffer allocation (CWE-131) leading to out-of-bounds write (CWE-787) in kernel pool, as exploited in the wild per CISA KEV.
CVE-2020-1054	Out-of-bounds write in kernel-mode driver, as exploited in the wild per CISA KEV.
CVE-2020-0041	Escape from browser sandbox using out-of-bounds write due to incorrect bounds check, as exploited in the wild per CISA KEV.
CVE-2020-0968	Memory corruption in web browser scripting engine, as exploited in the wild per CISA KEV.
CVE-2020-0022	chain: mobile phone Bluetooth implementation does not include offset when calculating packet length (CWE-682), leading to out-of-bounds write (CWE-787)
CVE-2019-1010006	Chain: compiler optimization (CWE-733) removes or modifies code used to detect integer overflow (CWE-190), allowing out-of-bounds write (CWE-787).
CVE-2009-1532	malformed inputs cause accesses of uninitialized or previously-deleted objects, leading to memory corruption
CVE-2009-0269	chain: -1 value from a function call was intended to indicate an error, but is used as an array index instead.
CVE-2002-2227	Unchecked length of SSLv2 challenge value leads to buffer underflow.
CVE-2007-4580	Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130)
CVE-2007-4268	Chain: integer signedness error (CWE-195) passes signed comparison, leading to heap overflow (CWE-122)
CVE-2009-2550	Classic stack-based buffer overflow in media player using a long entry in a playlist
CVE-2009-2403	Heap-based buffer overflow in media player using a long entry in a playlist

Manifested by implementation vulnerabilities

▼ Potential Mitigations

Phase: Requirements

Strategy: Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer.

Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Phase: Architecture and Design

Strategy: Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Examples include the Safe C String Library (SafeStr) by Messier and Viega [REF-57], and the Strsafe.h library from Microsoft [REF-56]. These libraries provide safer versions of overflow-prone string-handling functions.

Note: This is not a complete solution, since many buffer overflows are not related to strings.

Phases: Operation; Build and Compilation

Can be mitigated by controls

Institut Cybersécurité Occitanie

Ontology to the rescue

NIST

NVD MENU

Information Technology Laboratory

NATIONAL VULNERABILITY DATABASE

VULNERABILITIES

NOTICE UPDATED - MAY, 29TH 2024

The NVD has a new announcement page with status updates, news, and how to stay con

CVE-2021-21220 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Description

Insufficient validation of untrusted input in V8 in Google Chrome prior to 89.0.4389.128 allowed a remote attacker to potentially exploit heap corruption via a crafted HTML page.

This CVE is in CISA's Known Exploited Vulnerabilities Catalog

Reference CISA's BOD 22-01 and Known Exploited Vulnerabilities Catalog for further guidance and requirements.

Vulnerability Name	Date Added	Due Date	Required Action
Google Chromium V8 Improper Input Validation Vulnerability	11/03/2021	11/17/2021	Apply updates per vendor instructions.

Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-787	Out-of-bounds Write	NIST

Known Affected Software Configurations Switch to CPE 2.2

Configuration 1 (hide)

cpe:2.3:a:google:chrome:*:*:*:*:*:*:*:* Show Matching CPE(s)	**Up to (excluding)** **89.0.4389.128**

Configuration 2 (hide)

cpe:2.3:o:fedoraproject:fedora:32:*:*:*:*:*:*:*
Show Matching CPE(s)

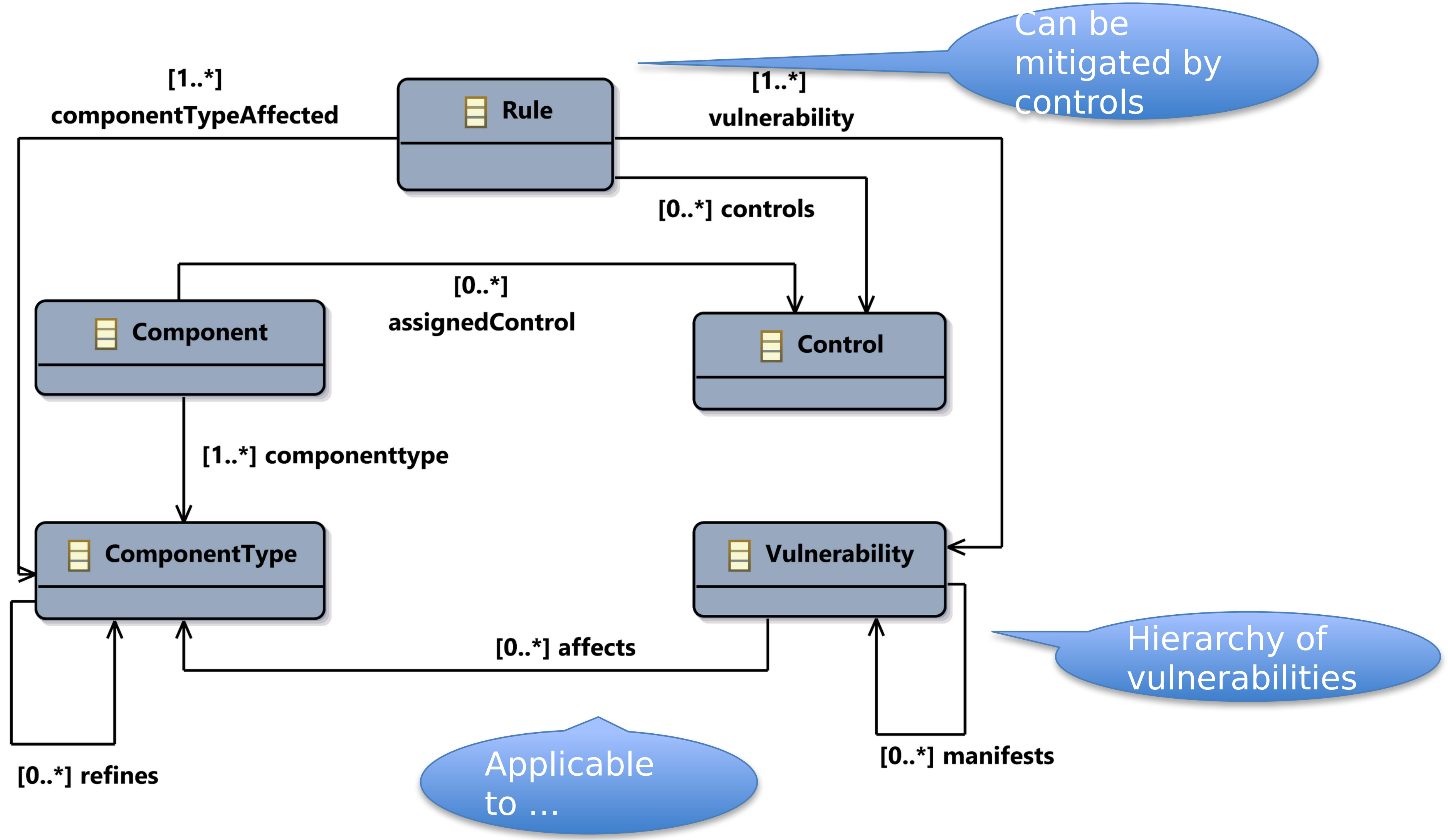
cpe:2.3:o:fedoraproject:fedora:33:*:*:*:*:*:*:*
Show Matching CPE(s)

cpe:2.3:o:fedoraproject:fedora:34:*:*:*:*:*:*:*
Show Matching CPE(s)

Applicable to...

CVE-2021-21220
NVD Published Date:
04/26/2021
NVD Last Modified:
12/14/2023
Source:
Chrome

Conceptual Modelling to the rescue



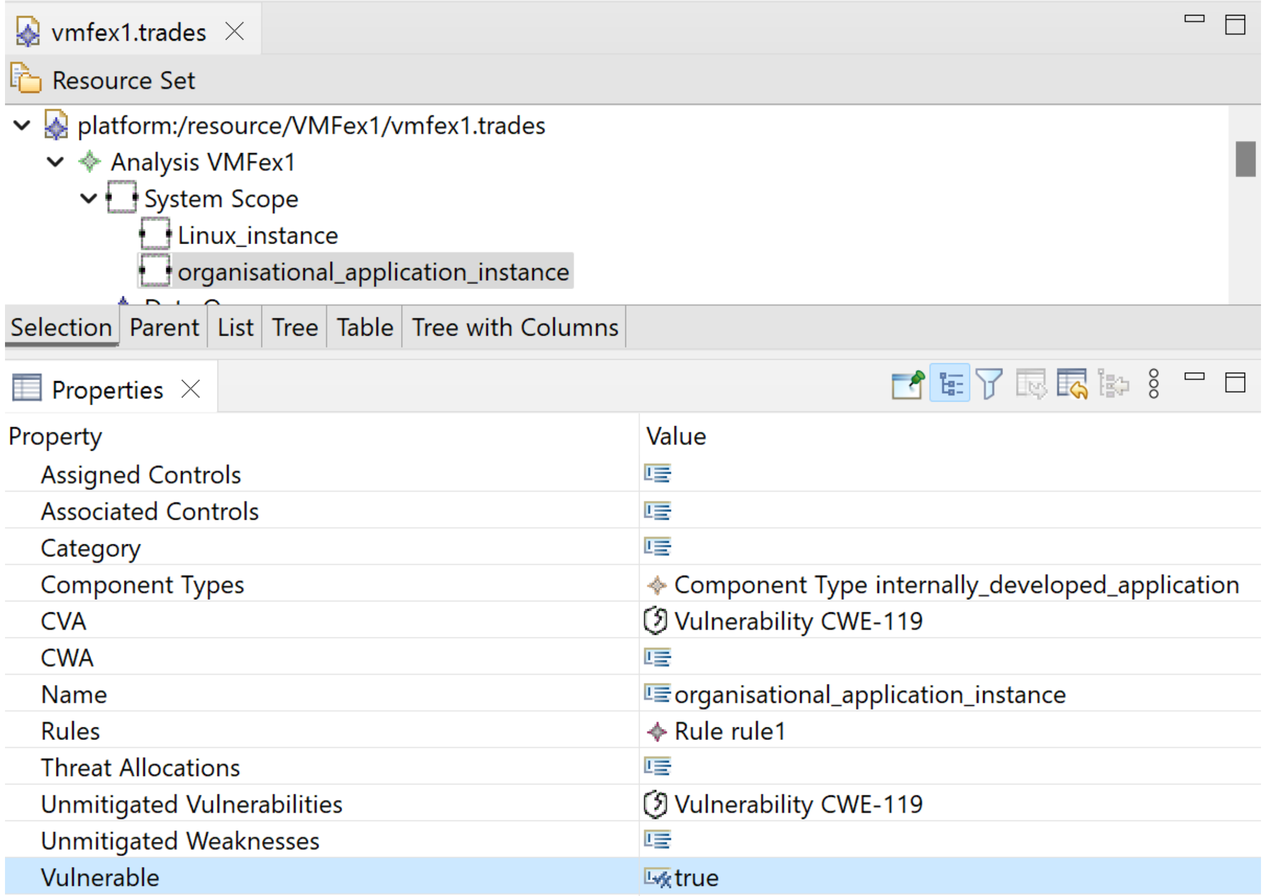
Example (1)

- Design situation: two components

$$C = \{\text{Linux\_instance}, \text{app\_instance}\}$$
$$T = \{\text{Linux\_OS}, \text{internal\_app}\}$$
$$V = \{\text{CWE-119}\}$$
$$S = \{\text{use\_memory\_safe\_languages}\}$$
$$R = \{\text{rule1}\}$$

CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

$$\begin{aligned}
\text{VULNS}(\text{internal_app}) &= \{\text{CWE-119}\}\\
\text{TYPES}(\text{Linux_instance}) &= \{\text{Linux_OS}\}\\
\text{TYPES}(\text{app_instance}) &= \{\text{internal_app}\}\\
\text{RVULNS}(\text{rule1}) &= \{\text{CWE-119}\}\\
\text{RTYPES}(\text{rule1}) &= \{\text{internal_app}\}\\
\text{RCONTROLS}(\text{rule1}) &= \{\text{use_memory_safe_languages}\}
\end{aligned}$$



Example (1)

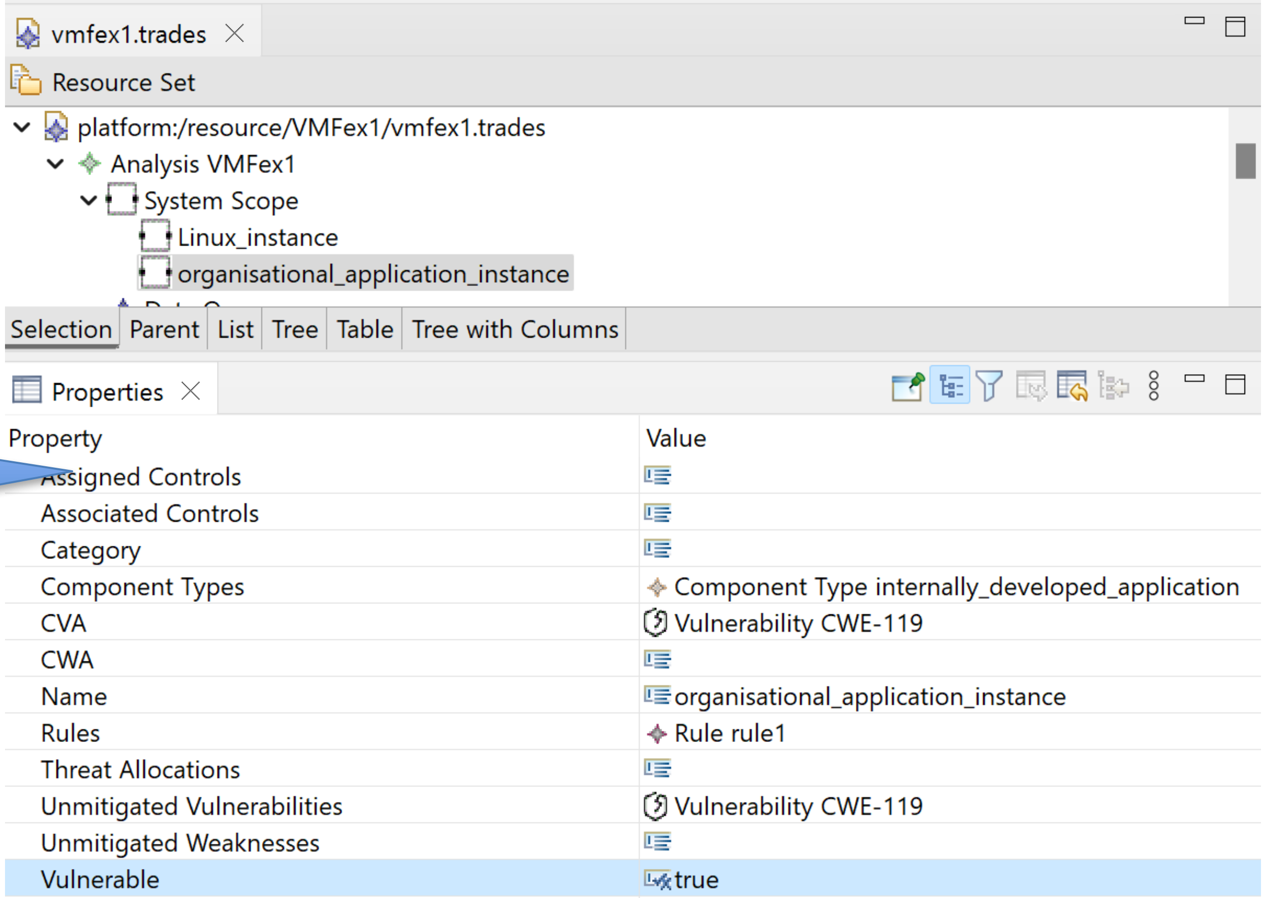
- Design situation: two components

$$
\begin{aligned}
C &= \{\text{Linux\_instance}, \text{app\_instance}\} \\
T &= \{\text{Linux\_OS}, \text{internal\_app}\} \\
V &= \{\text{CWE-119}\} \\
S &= \{\text{use\_memory\_safe\_languages}\} \\
R &= \{\text{rule1}\}
\end{aligned}
$$

CWE-119: Improper Restriction o
Bounds of a Memory Buffer

Let's add a control

$$
\begin{aligned}
\text{VULNS}(\text{internal\_app}) &= \{\text{CWE-119}\} \\
\text{TYPES}(\text{Linux\_instance}) &= \{\text{Linux\_OS}\} \\
\text{TYPES}(\text{app\_instance}) &= \{\text{internal\_app}\} \\
\text{RVULNS}(\text{rule1}) &= \{\text{CWE-119}\} \\
\text{RTYPES}(\text{rule1}) &= \{\text{internal\_app}\} \\
\text{RCONTROLS}(\text{rule1}) &= \{\text{use\_memory\_safe\_languages}\}
\end{aligned}
$$



Example (2)

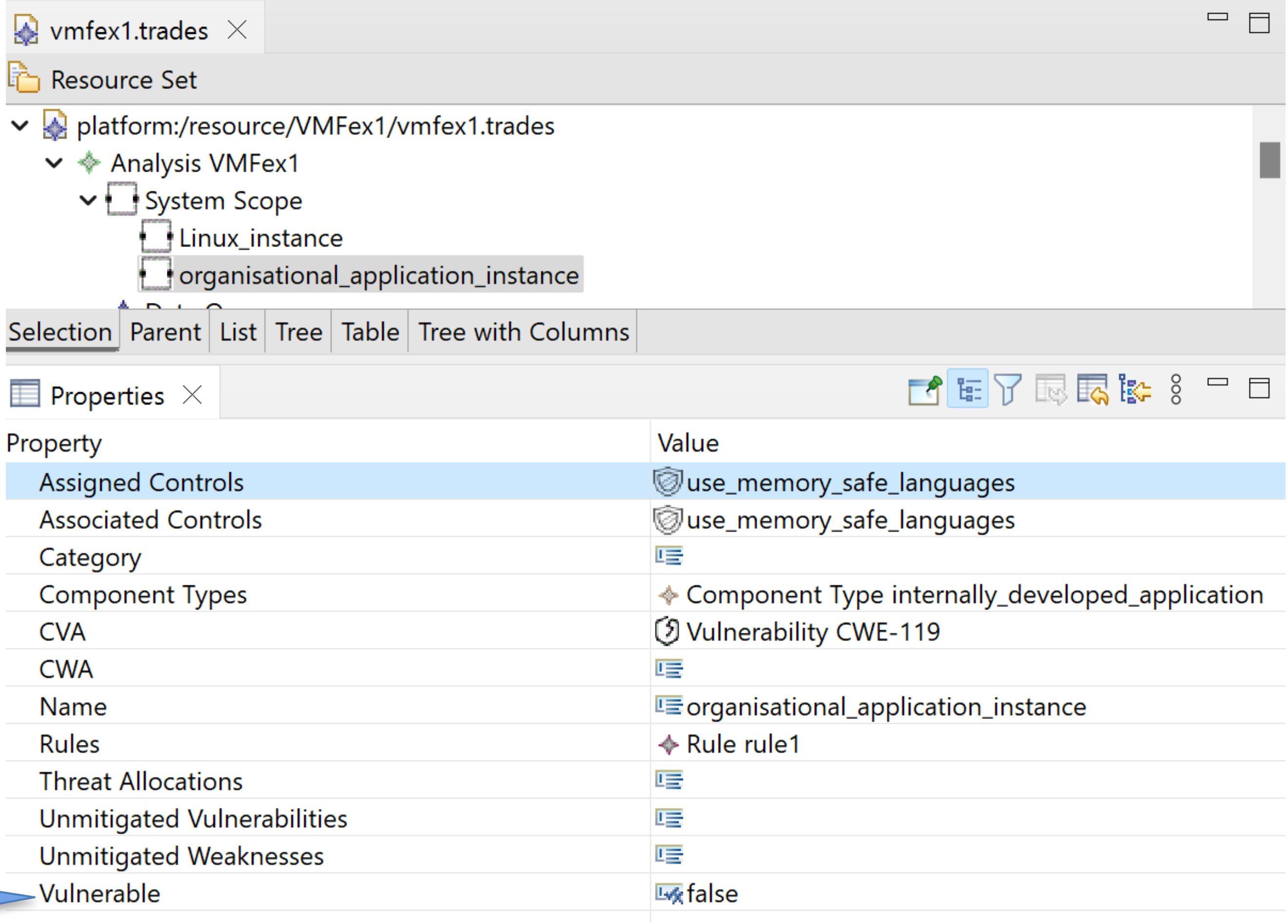
- Design situation: two components

$$C = \{\text{Linux\_instance}, \text{app\_instance}\}$$
$$T = \{\text{Linux\_OS}, \text{internal\_app}\}$$
$$V = \{\text{CWE-119}\}$$
$$S = \{\text{use\_memory\_safe\_languages}\}$$
$$R = \{\text{rule1}\}$$

CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

$$\text{VULNS}(\text{internal\_app}) = \{\text{CWE-119}\}$$
$$\text{TYPES}(\text{Linux\_instance}) = \{\text{Linux\_OS}\}$$
$$\text{TYPES}(\text{app\_instance}) = \{\text{internal\_app}\}$$
$$\text{RVULNS}(\text{rule1}) = \{\ldots$$
$$\text{RTYPES}(\text{rule1})$$
$$\text{RCONTROLS}(\text{rule1}) = \{\ldots\text{\_languages}\}$$

No longer vulnerable



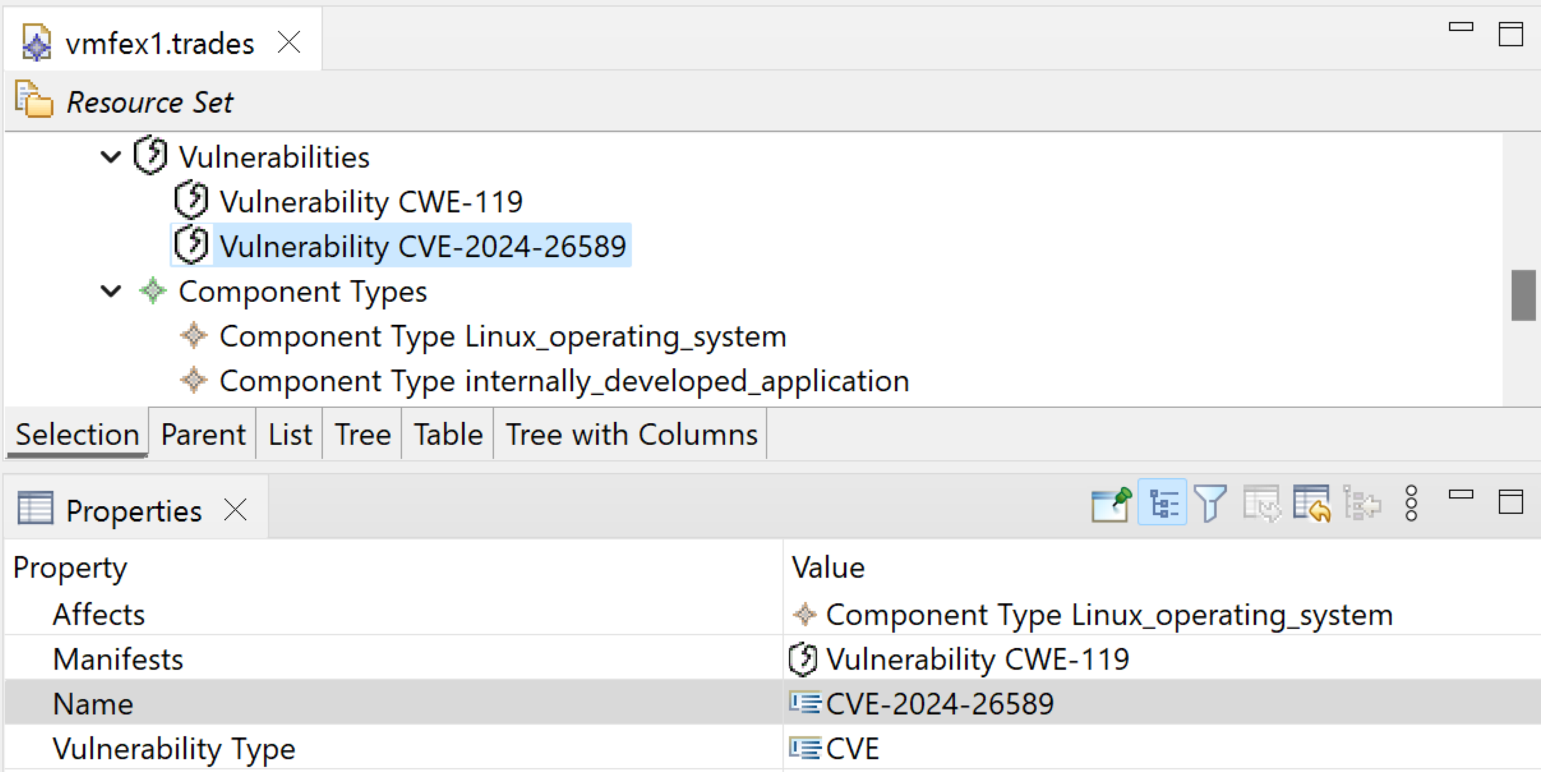
Example (3)

- Design situation: two components

$$C = \{Linux\_instance, app\_instance\}$$
$$T = \{Linux\_OS, internal\_app\}$$
$$V = \{CW\ \text{-}119\}$$
$$S = \{use\_\ \ mory\_safe\_languages\}$$
$$R = \{$$

New vulnerability disclosure:

CWE … Operations withi…
Bound…



$$
\begin{aligned}
\mathrm{VULNS}(internal\_app) &= \{CWE\text{-}119\}\\
\mathrm{TYPES}(Linux\_instance) &= \{Linux\_OS\}\\
\mathrm{TYPES}(app\_instance) &= \{internal\_app\}\\
\mathrm{RVULNS}(rule1) &= \{CWE\text{-}119\}\\
\mathrm{RTYPES}(rule1) &= \{internal\_app\}\\
\mathrm{RCONTROLS}(rule1) &= \{use\_memory\_safe\_languages\}
\end{aligned}
$$

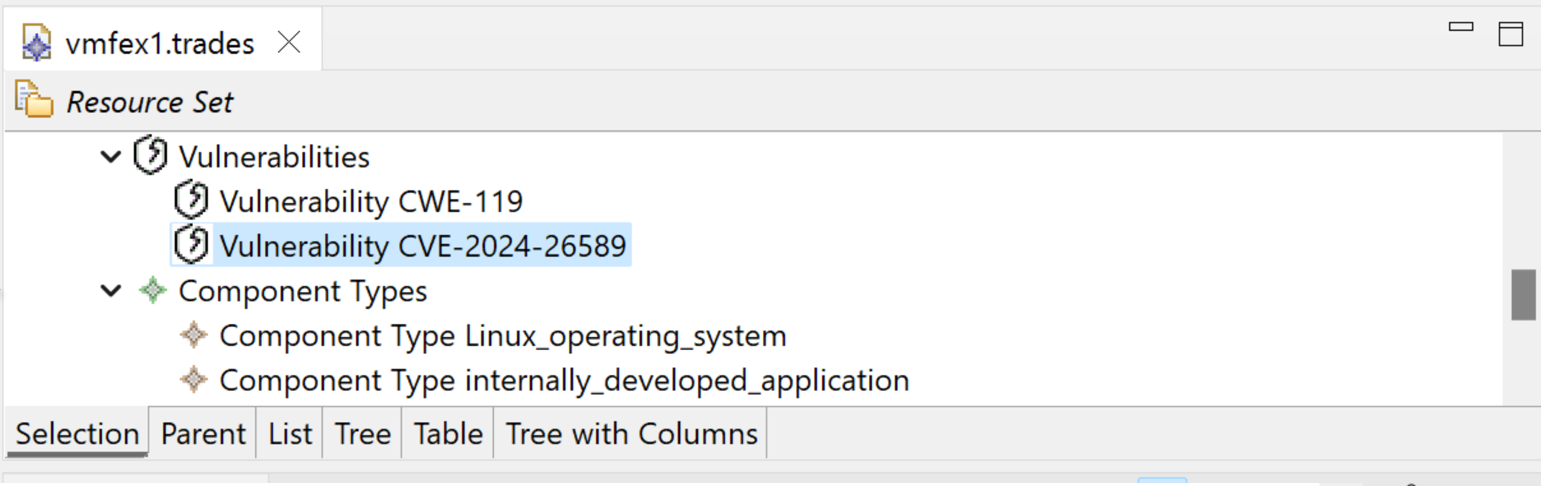
Example (3)

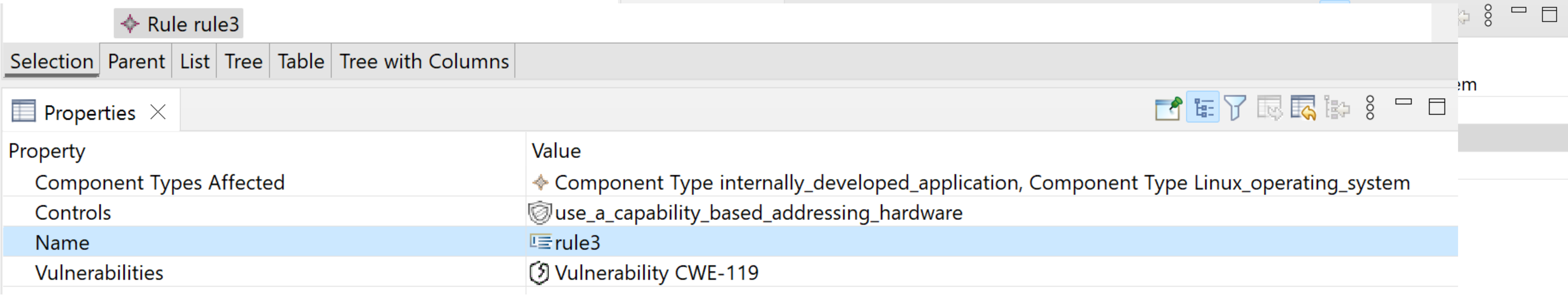
- Design situation: two components

$$C = \{\text{Linux\_instance}, \text{app\_instance}\}$$
$$T = \{\text{Linux\_OS}, \text{internal\_app}\}$$
$$V = \{\text{CWE-119}\}$$
$$S = \{$$
$$R = \{\text{rule1}\}$$

Let's solve "by-design"

CWE-119: Improper Restriction of Operations withi Bounds of a Memory Buffer





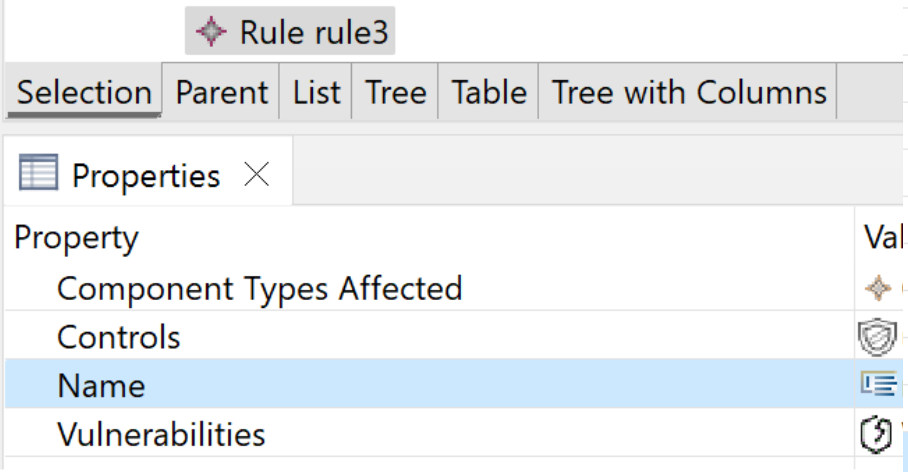
Example (3)

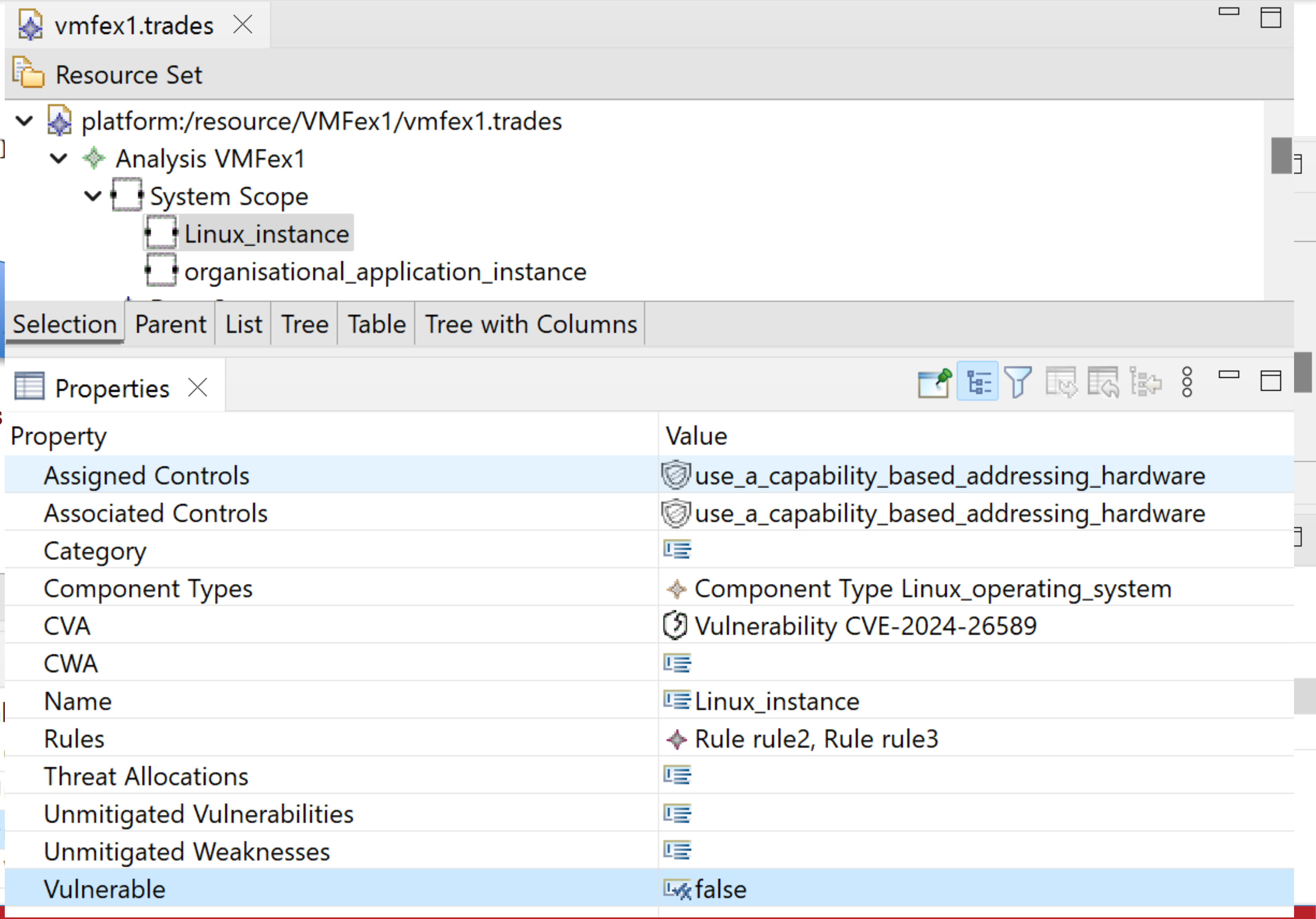
- Design situation: two com

$$C = \{Linux\_instance, app\_instan$$
$$T = \{Linux\_OS, internal\_app\}$$
$$V = \{CWE\text{-}119\}$$
$$S = \{$$
$$R = \{rule1\}$$

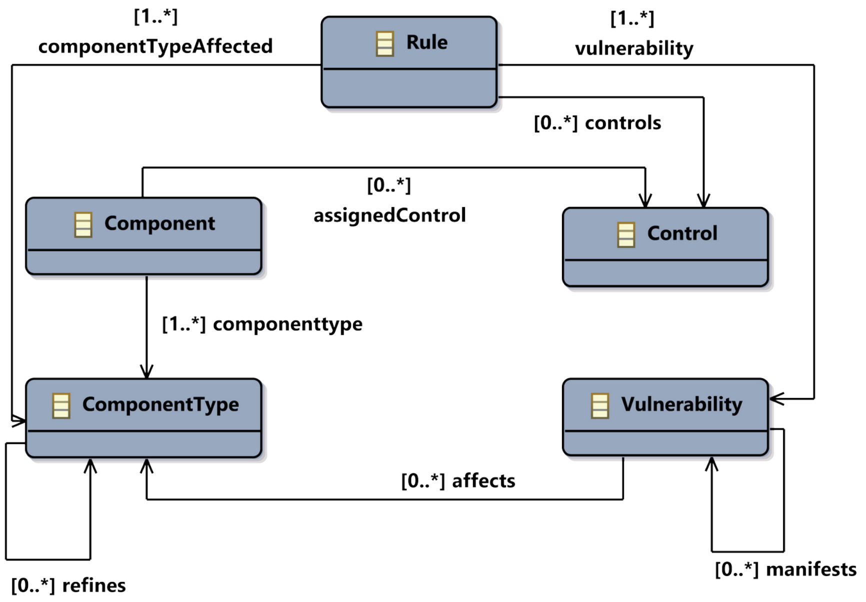
Let's solve "by-design"

CWE-119: Improper Restriction of Operations Bounds of a Memory Buffer

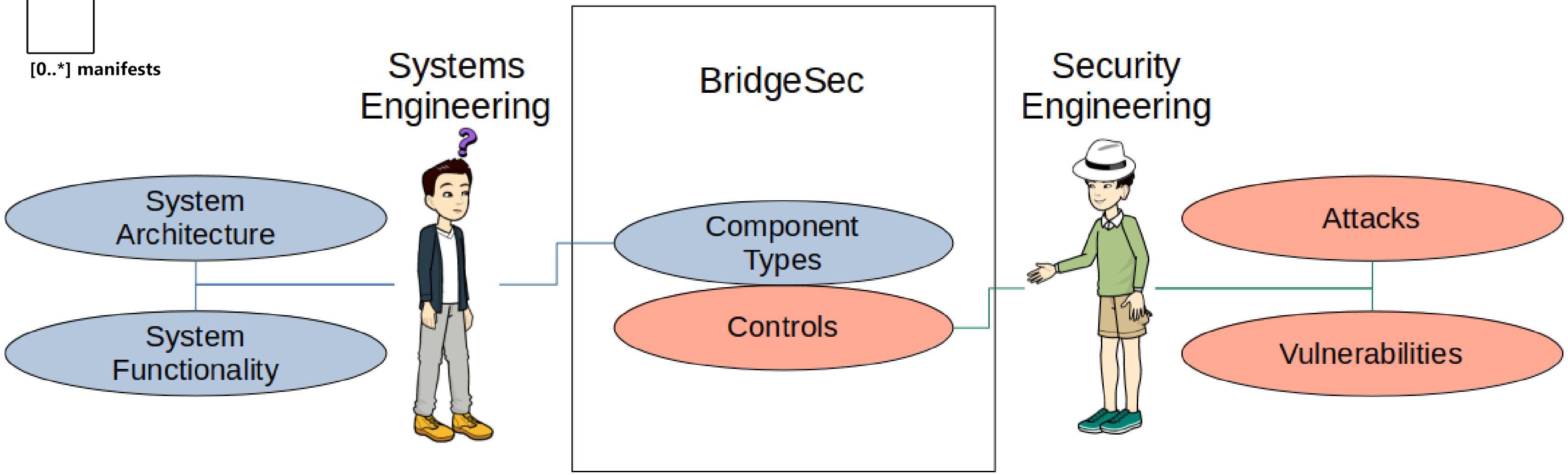




Design is a sociotechnical system



Information exchange interface



Summary

- Model-driven methodology for vulnerability management
- Automated reasoning mechanism
- Information-exchange interface
- Vulnerability management made:
    - ➢Systematic
    - ➢Scalable
    - ➢Rigorous

Thank You

nan.messe@irit.fr