Using GrammaTech CodeSentry and CodeSonar to improve Software Security and comply with IEC 62443

Customer:

GrammaTech
Bethesda, MD
USA

Contract No.: Q21/02-139
Version V1, Revision R2, May 4th, 2021
Bill Thomson

Table of Contents

1 Purpose and Scope

In order to develop secure code free of vulnerabilities, suppliers are increasingly following a secure development lifecycle to achieve these goals. The IEC 62443-4-1 standard (Security for industrial automation and control systems –Part 4-1: Secure product development lifecycle requirements) defines specific requirements for using a secure development lifecycle in the design, implementation, maintenance and testing of products used in industrial automation and control systems. Grammatech's CodeSentry and CodeSonar tools can be used to help suppliers comply with this standard.

Two major contributors to security vulnerabilities found in products today are implementation weaknesses in programs created in languages such as C and C++ and the use of Third Party Software (TPS). The CodeSentry and CodeSonar tools can address both of these issues.

This document introduces common causes of security vulnerabilities including implementation programming weaknesses in programing languages and TPS. In addition it describes TPS types, describes TPS specific security challenges and provides guidance on how to use the Grammatech CodeSentry and CodeSonar tools in a workflow to select and manage TPS and overall product security.

2 Abbreviations

BOM	Bill Of Materials – list of software components and this versions in a software library or software image
CVE	Common Vulnerabilities and Exposures - publicly known information-security vulnerabilities and exposures
OSS	Open Source Software – software where source code is available free of charge subject to licensing agreements
TPS	Third Party Software – software developed by someone outside of a product vendor's organization

3 Software Vulnerabilities: Introduction

As mentioned in the introduction, software implemented in widely used programming languages such as C, C++ and Java has common implementation (programming) weaknesses which can lead to potential security vulnerabilities with examples including but not limited to:

- Buffer overflows
- Using banned functions with known vulnerabilities
- Memory leaks
- Deadlock
- Unhandled exceptions

Awareness of these vulnerabilities and the importance of mitigating them was the driver behind including requirements for this in the IEC 62443-4-1 standard. Specifically, IEC 62443-4-1 requires performing SA (Static Analysis), identifying potential vulnerabilities along with their severity and addressing vulnerabilities based on the risk level.

The CodeSonar tool was designed and implemented with these requirements in mind and can help with the process of identifying security issues along with their severity in internally developed code or TPS distributed as binaries. Examples of the types of security issues CodeSonar can uncover include but are not limited to:

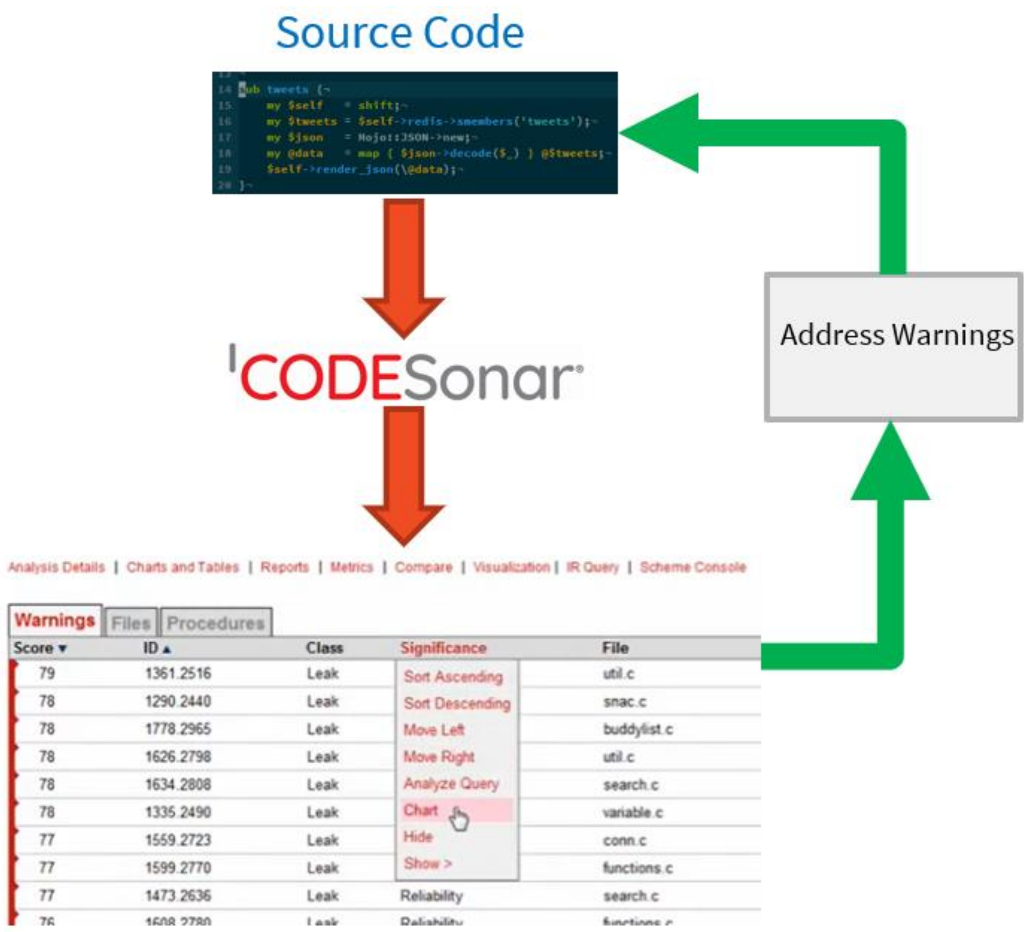
- Unitialized variables
- Use of banned (unsafe) functions
- Buffer over- and underruns
- Cast and conversion problems
- Command injection
- Copy-paste error
- Concurrency
- Ignored return value
- Memory leak
- Tainted data
- Null pointer dereference

4 Addressing Software Vulnerabilities and Meeting IEC 62443-4-1 SA requirements with CodeSonar

With the increasing prevalence of Agile, DevOps and CI/CD development methodologies meeting the 4-1 requirements for running SA requires use of a tool well suited for this workflow and which supports widely used programming languages such as C, C++ and Java.

4.1 CodeSonar Workflow For Internally Developed Software

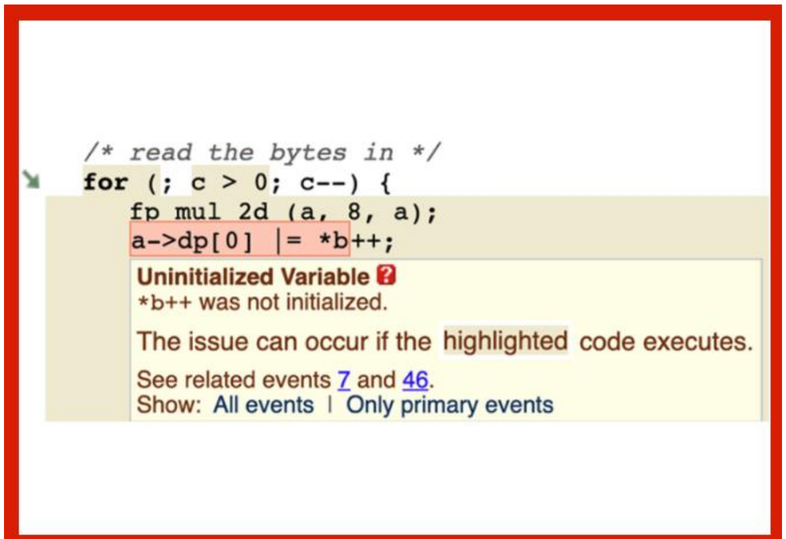
CodeSonar makes the workflow for identifying and addressing security issues in C, C++ and Java source code simple enough to be shown in a 1-page flow chart:



```
sub tweets {
    my $self   = shift;
    my $tweets = $self->redis->smembers('tweets');
    my $json   = Mojo::JSON->new;
    my @data   = map { $json->decode($_) } @$tweets;
    $self->render_json(\@data);
}
```

Score ▾	ID ▴	Class	Significance	File
79	1361.2516	Leak		util.c
78	1290.2440	Leak		snac.c
78	1778.2965	Leak		buddylist.c
78	1626.2798	Leak		util.c
78	1634.2808	Leak		search.c
78	1335.2490	Leak		variable.c
77	1559.2723	Leak		conn.c
77	1599.2770	Leak		functions.c
77	1473.2636	Leak	Reliability	search.c
76	1608.2780	Leak	Reliability	functions.c

Since "Address Warnings" requires evaluating the security risk of issues, CodeSonar's classification and colloquial description of the issue help quicky determine which issues have high enough risk to warrant fixing.   For issues which do require fixing CodeSonar's path and call tree visualizations help reduce the effort of implementing a fix.



```
/* read the bytes in */
for (; c > 0; c--) {
    fp_mul_2d (a, 8, a);
    a->dp[0] |= *b++;
```

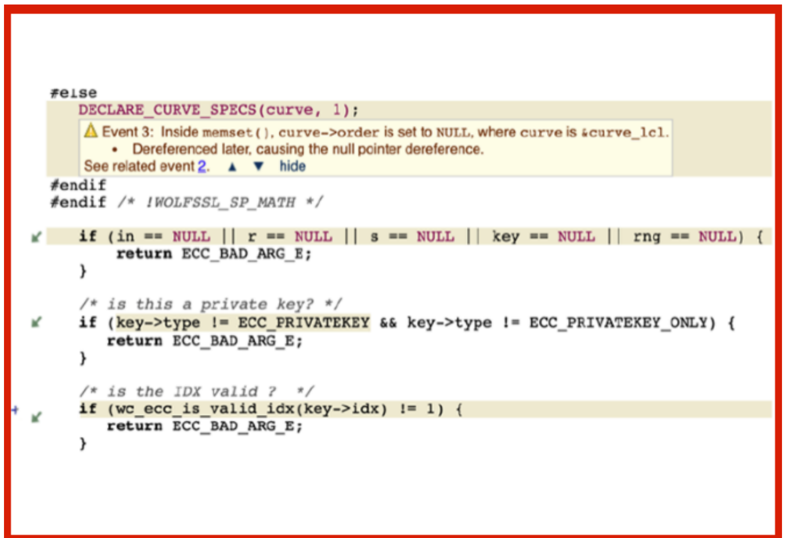
Uninitialized Variable
*b++ was not initialized.

The issue can occur if the highlighted code executes.

See related events 7 and 46.
Show: All events | Only primary events

1. Textual Descriptions

Easy, clear textual descriptions describe what the problem is.



```
#else
    DECLARE_CURVE_SPECS(curve, 1);
```

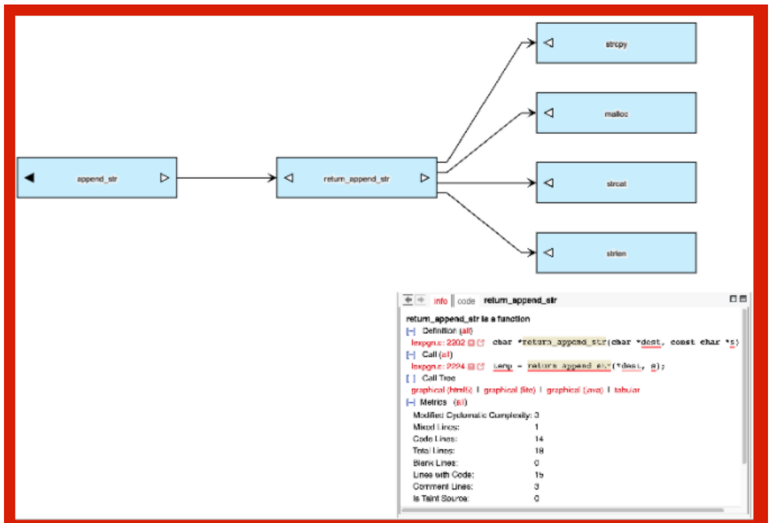
⚠ Event 3: Inside memset(), curve->order is set to NULL, where curve is &curve_lcl.
- Dereferenced later, causing the null pointer dereference.

See related event 2. ▲ ▼ hide

```
#endif
#endif /* !WOLFSSL_SP_MATH */

    if (in == NULL || r == NULL || s == NULL || key == NULL || rng == NULL) {
         return ECC_BAD_ARG_E;
    }

    /* is this a private key? */
    if (key->type != ECC_PRIVATEKEY && key->type != ECC_PRIVATEKEY_ONLY) {
        return ECC_BAD_ARG_E;
    }

    /* is the IDX valid ?  */
    if (wc_ecc_is_valid_idx(key->idx) != 1) {
        return ECC_BAD_ARG_E;
    }
```

2. Path Visualization

Shaded background and annotations explain the defect path.



```
return_append_str is a function
[-] Definition (all)
  lexpgn.c: 2202   char *return_append_str(char *dest, const char *s)
[-] Call (all)
  lexpgn.c: 2224   temp = return_append_str(*dest, s);
[ ] Call Tree
  graphical (html5) | graphical (lite) | graphical (java) | tabular
[-] Metrics  (all)
  Modified Cyclomatic Complexity: 3
  Mixed Lines:                    1
  Code Lines:                     14
  Total Lines:                    18
  Blank Lines:                    0
  Lines with Code:                15
  Comment Lines:                  3
  Is Taint Source:                0
```

3. Call Tree Visualization

To understand how a function fits in the larger application. Learn more >

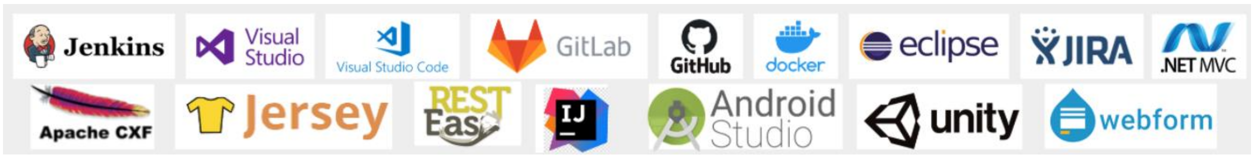
CodeSonar provides the ability to integrate with commonly used tools in modern development methodologies such as DevOps, DevSecOps, CI/CD and Agile.

AGILE
1 Requirements
2 Design
3 Development
4 Testing
5 Deployment
6 Review

CI/CD

Dev
Ops
code
plan
build
test
release
deploy
operate
monitor

Some of the supported tools include:



This integration helps facilitate IEC 62443-4-1 compliance and streamline the overall development process.

5 TPS Introduction: TPS is not our software but it is in our offering

5.1 What is TPS?

TPS (Third Party Software) is software created and maintained by a third party. TPS comes in two forms:

<u>OSS (Open Source Software)</u>

- Community developed and available to use free of charge subject to licensing restrictions.
- Full access to source code
- Common examples include: Linux distributions, OpenSSL, mbedTLS, netSNMP

<u>Commercial Software</u>

<u>COTS (Commercial Off The Shelf) Software</u>

- Commercially developed software
- Examples include entire operating systems such as Microsoft Windows, protocol stacks such as Pyramid Solutions EtherNet/IP and CIP protocol

<u>Custom Commercial Software</u>

- Software developed under contract specifically for use in a vendors product.
- Examples include outsourced software development and development of a custom protocol stack by a protocol stack vendor

5.2 Why is TPS used?

TPS is increasingly common in a wide variety of products because it has advantages including but not limited to:

- Peer reviewed and “battle hardened” implementations for complex security-related functionality such as SSL/TLS are widely available.
- Since OSS is “free” it doesn’t require capital expenditures to use provided the licensing agreements are complied with.
- TPS can incorporate capabilities a company doesn’t have the skillset to develop internally. For example, developing a protocol stack.
- Frees up engineering resources to develop software aligned with a company’s core competencies.

5.3 TPS-specific security risks and challenges

Suppliers are responsible for the security of their offering irrespective of who provides the components used to implement it. For example, if Anisha purchases an automobile and the remote unlocking feature has a security issue, she fully expects the automobile manufacturer to resolve the issue without her needing to learn who provided the locking subsystem for the automobile. Customers and security standards alike require vendors bear full responsibility for the security of any offering carrying their brand.

TPS (Third Party Software) and TPS security issues are increasingly prevalent in the industry. In addition, TPS security issues pose some unique challenges that make TPS selection as well as timely identification and resolution of security issues critical. This has led to customers, governments and cybersecurity standards all mandating that suppliers manage security for the TPS in their offering.

TPS in general and OSS in particular introduces some unique security risks and challenges:

1. Attackers have full access to the source code for OSS which increases their chances of finding vulnerabilities.
2. Since TPS is so widely used, if attackers find a vulnerability in a TPS component (CVEs - Common Vulnerabilities and Exposures) it will affect multiple vendor's offerings. This increases attacker motivation. For example, when the Heartbleed vulnerability was discovered in OpenSSL it affected products ranging from financial web sites to security cameras to network equipment because OpenSSL is incredibly prevalent. As a result, Heartbleed made the mainstream news and there was a harried effort to get patched versions of OpenSSL deployed into a multitude of vendor's offerings before attackers could exploit it.
3. OSS licensing agreements often require disclosing the OSS BOM (bill of materials) used in a vendor's offering.
4. **For TPS in general but OSS in particular, attackers learn about TPS vulnerabilities at the same time or before vendors using TPS in their offering and the vendor's customers.**
5. Vendors using TPS in their offering are generally dependent upon the provider of the TPS to address security issues.

Of the items above, 4 is perhaps the most problematic and unique to TPS. It is also one of the main drivers for the content of this paper and the reason security standards such as IEC 62443-4-1 contain requirements for managing TPS security.

6 Managing TPS Security – TPS Selection

6.1 Selecting TPS

Customers hold vendors accountable for the security of the vendor's product regardless of who implemented the software components contained within it. In addition, security standards such as IEC 62443-4-1 (see SM-9: Security requirements for externally provided components) require the security of TPS components within a product is evaluated and managed. The implications of the two previous statements are TPS selection needs to be approached carefully.

6.2 TPS Selection Basic Workflow

Selecting a TPS component follows the basic workflow outlined below.

1. Determine whether to use OSS or commercial software

   OSS Pros

   - No capital expenditures required
   - Some OSS is widely used and battle hardened

   OSS Cons

- Operating expense required to keep up with CVEs
- If OSS community abandons support for component, vendor needs to pickup support or switch components

Commercial Software Pros

- Vendor is responsible for operating expense of managing CVEs
- Some vendors follow a robust secure development lifecycle

Commercial Software Cons

- Vendor may fall behind in managing CVEs
- Capital expense of acquiring the software and paying for support

2. Identify BOM and CVEs in candidate TPS:
    - Compare CVEs in candidate TPS libraries and use that as one of the TPS selection criteria.
    - Include the quantity of CVEs, severity of CVEs and age of unresolved CVEs in the evaluation
3. For candidate TPS libraries distributed as binaries, identify the type of security vulnerabilities static analysis tools would find in source code.

6.3 TPS Selection: Power Tools To The Rescue

As one might imagine, manually establishing an accurate BOM (bill of materials) and set of applicable CVEs is impractical for all but trivial TPS components. This is especially true for cases where a TPS component might be an entire application. For example, an IACS (Industrial Automation and Control Systems) system integrator intending to comply with the IEC 62443-3-3 standard must identify and manage the CVEs in all the components in the system. In the IACS system context, component encompass everything from embedded devices to Windows applications to Browsers.

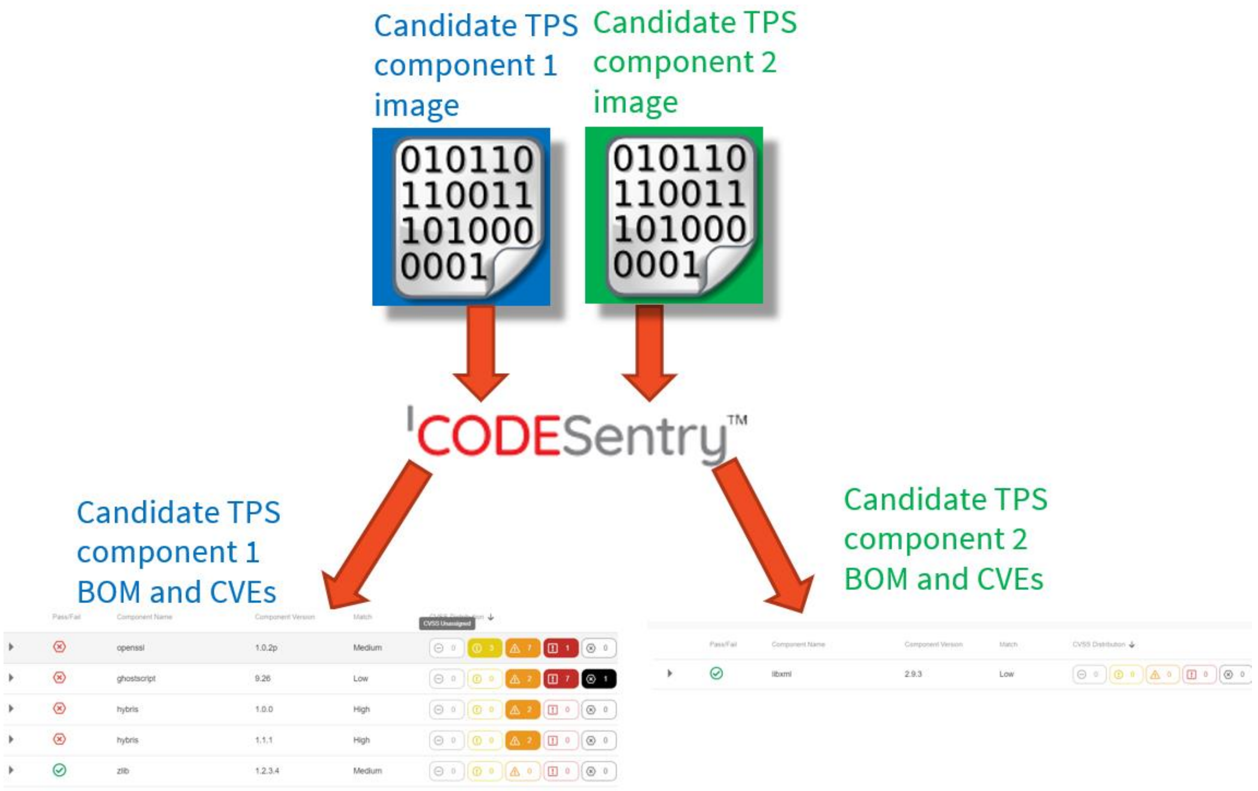
In addition, many commercial TPS software components are distributed as binary libraries so running static analysis on source code isn't feasible. But, as previously stated both the IEC 62443-4-1 standard and customers require you address security vulnerabilities in your product regardless of where the components come from.

The GrammaTech CodeSentry and CodeSonar Binary tools will allow a product or system vendor to scan a binary software component or application:

- compile a BOM
- identify the CVEs for the components in the BOM.
- Identify potential vulnerabilities related to known weaknesses in programming languages such as C, C++ and Java

6.3.1 Power Tools To The Rescue Part 1: CodeSentry

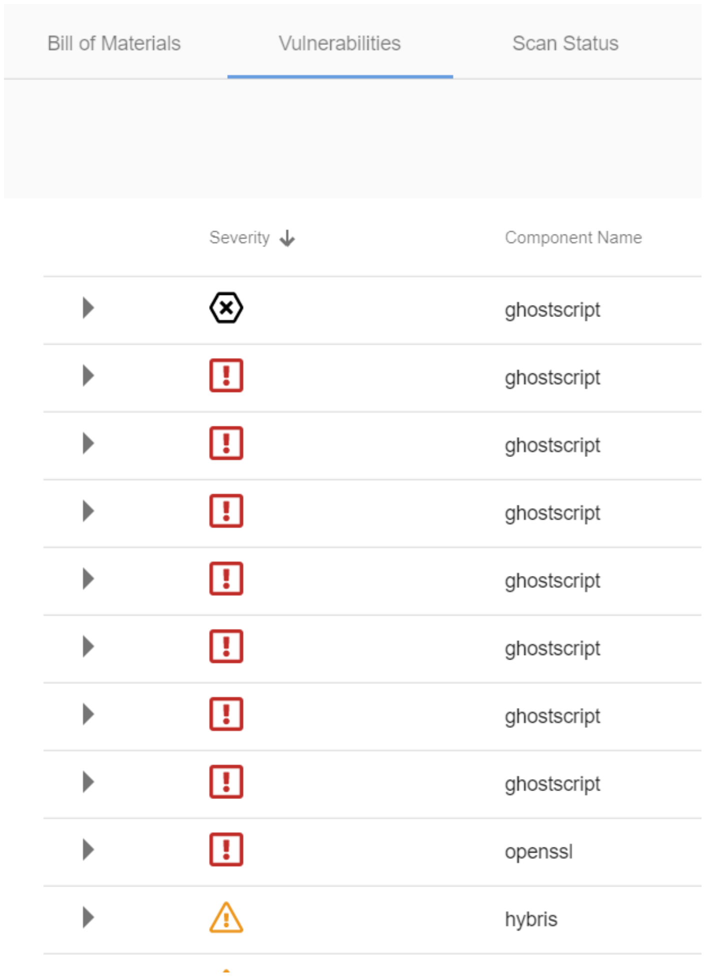
Below is a CodeSentry Workflow diagram for compiling a BOM and CVE list for two candidate TPS components.


Candidate TPS component 1 image
Candidate TPS component 2 image
010110 110011 101000 0001
010110 110011 101000 0001
CODESentry™
Candidate TPS component 1 BOM and CVEs
Candidate TPS component 2 BOM and CVEs
CVSS Unassigned

	Pass/Fail	Component Name	Component Version	Match	CVSS Distribution ↓				
▸	✗	openssl	1.0.2p	Medium	0	3	7	1	0
▸	✗	ghostscript	9.26	Low	0	0	2	7	1
▸	✗	hybris	1.0.0	High	0	0	2	0	0
▸	✗	hybris	1.1.1	High	0	0	2	0	0
▸	✓	zlib	1.2.3.4	Medium	0	0	0	0	0

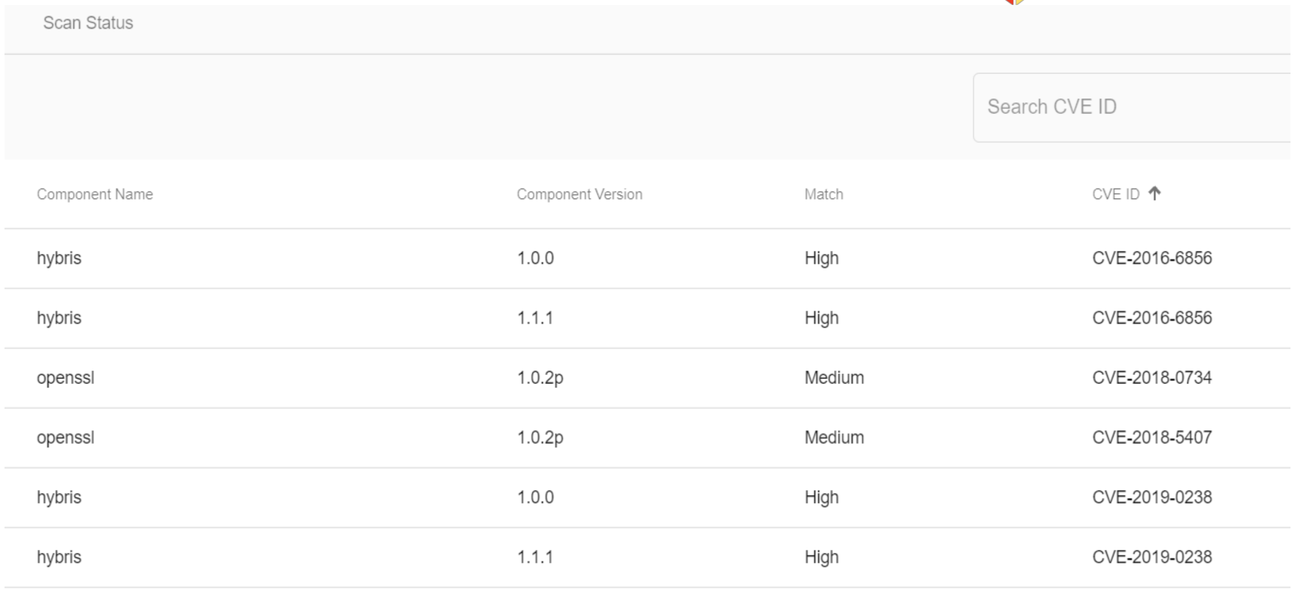
	Pass/Fail	Component Name	Component Version	Match	CVSS Distribution ↓				
▸	✓	libxml	2.9.3	Low	0	0	0	0	0

In this example CodeSentry shows Candidate TPS component 2 has far fewer vulnerabilities so is likely a better choice than TPS component 1. Drilling down to the next level of detail, on the Vulnerabilities tab one can sort vulnerabilities by severity or age:



	Severity ↓	Component Name
▸	⊗	ghostscript
▸	!	ghostscript
▸	!	ghostscript
▸	!	ghostscript
▸	!	ghostscript
▸	!	ghostscript
▸	!	ghostscript
▸	!	ghostscript
▸	!	openssl
▸	⚠	hybris

The above example is sorted by Severity and shows 1 critical and 8 high severity CVEs in the gostscript subcomponent. It would likely be a bad idea to select this TPS library because all of those high and critical severity CVEs would become part of *your* product and *your* customers and IEC 62443-4-1 would hold *you* accountable for addressing them.



Scan Status

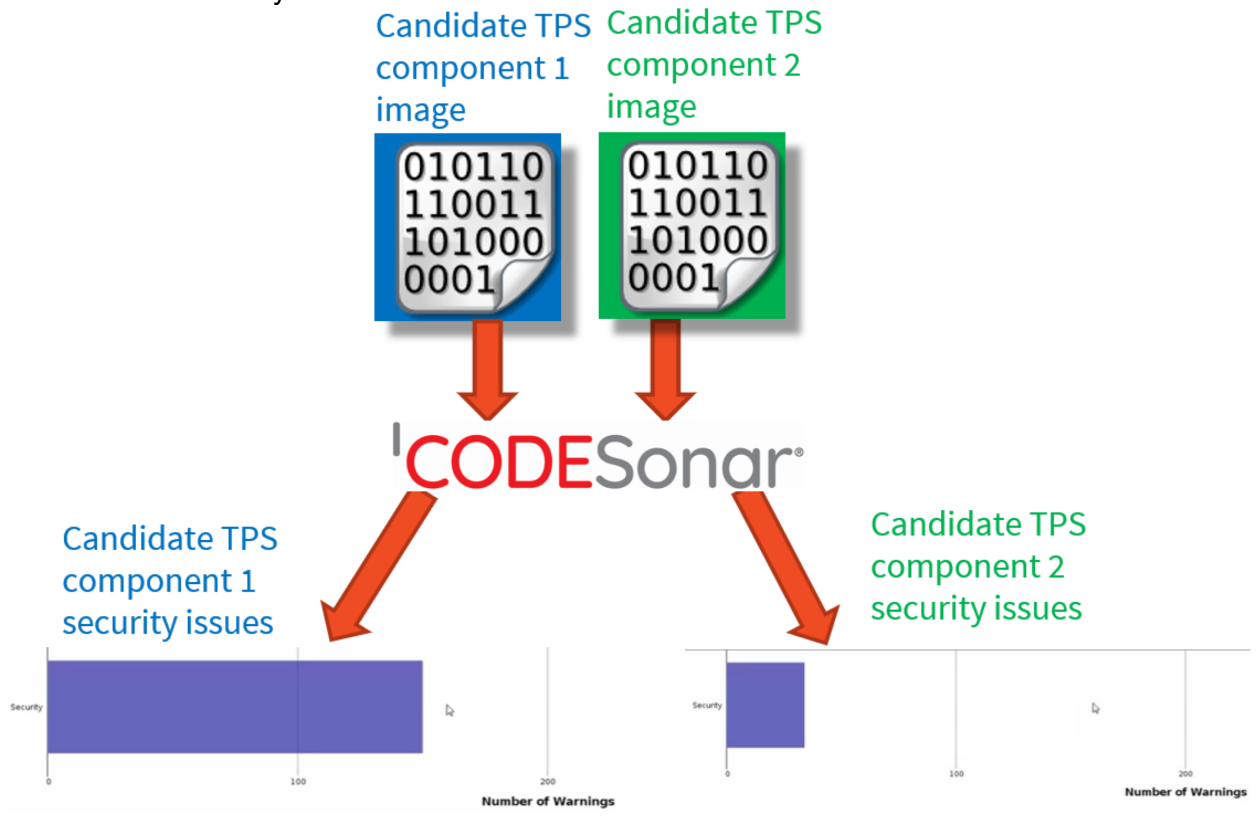
Search CVE ID

Component Name	Component Version	Match	CVE ID ↑
hybris	1.0.0	High	CVE-2016-6856
hybris	1.1.1	High	CVE-2016-6856
openssl	1.0.2p	Medium	CVE-2018-0734
openssl	1.0.2p	Medium	CVE-2018-5407
hybris	1.0.0	High	CVE-2019-0238
hybris	1.1.1	High	CVE-2019-0238

The above example sorted by CVE ID shows 2 High severity CVEs from 2016 (CVE-2016-xxxx). The component vendor is not keeping up with CVEs in their application. *This indicates a lack of process rigor by the TPS provider in managing the CVEs in the subcomponents of their software.*

6.3.2 Power Tools To The Rescue Part 2: CodeSonar For Binaries

Below is a CodeSonar Workflow diagram for assessing known weaknesses in two candidate TPS binary libraries:



Candidate TPS component 1 image
Candidate TPS component 2 image
010110 110011 101000 0001
CODESonar
Candidate TPS component 1 security issues
Candidate TPS component 2 security issues
Security
0
100
200
Number of Warnings

CodeSonar shows Candidate TPS component 2 has far fewer security warnings than component 1. This analysis is valuable for complying with the IEC 62443-4-1 requirement to evaluate the security of TPS components used in a product. It also provides evidence to help motivate the supplier of the chosen TPS library to address the security issues.

7 Managing TPS Security – Addressing CVEs

7.1 TPS CVE Management Basic Workflow

The prevalence of TPS (OSS in particular) security issues has led customers, governments and cybersecurity standards all mandating vendors manage security for the TPS in their offerings. For example, the IEC 62443-4-1 cybersecurity standard requires vendors employ a process for identifying security issues in TPS and managing them. At a conceptual level, the workflow for managing TPS security issues is very straight forward:

1. Construct a BOM (bill of materials) consisting of all the TPS components (names and versions) used in the offering.
2. Identify *potential* CVEs (Common Vulnerabilities and Exposures – publicly known security issues) in each component in the BOM.
3. Investigate each CVE and for any which are legitimate:
    a. Contact the COTS vendor or OSS provider to see if a fix is available.
    b. Once a fix is available, incorporate the patched or new TPS component version in your offering.

There are several challenges with the above workflow:

1. Manually constructing an *accurate* BOM can be very problematic.
2. Since attackers and security researchers are constantly uncovering new CVEs in TPS, customers and standards alike require vendors continuously check for CVEs in the TPS in their offerings. This combined with 2 above makes manually tracking TPS CVEs impractical.

7.2 TPS CVE Management: Power Tools To The Rescue

The CodeSentry tool greatly improves the efficiency of the CVE management work flow and facilitates the ongoing effort of managing TPS security required by customers and security standards alike. On the next page is a sample CodeSentry-based workflow. This workflow is scalable to large numbers of TPS components and CVEs. Since the list of CVEs is constantly changing customers and security standards alike require regularly checking for new CVEs even when the TPS component versions are static.

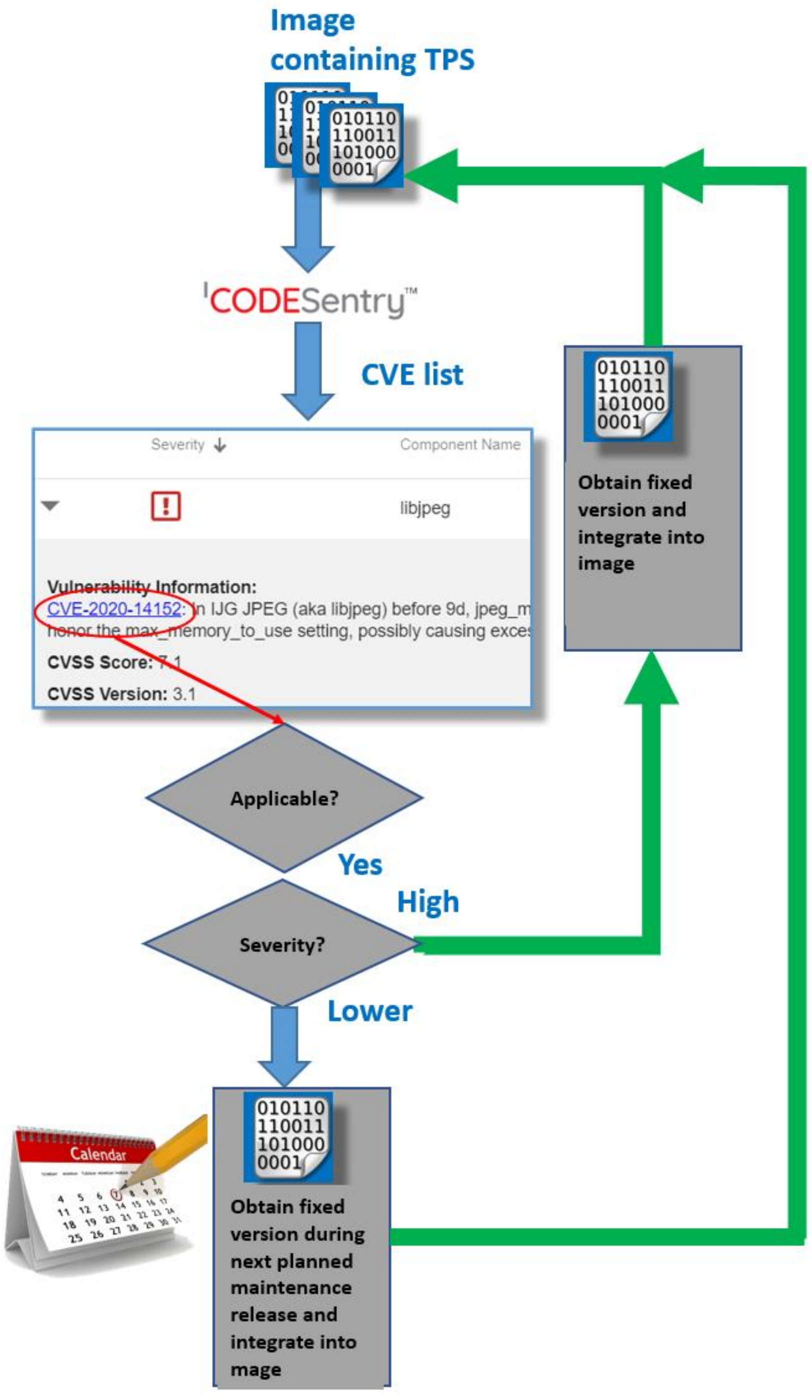


Image containing TPS
010110
110011
101000
0001
CODESentry™
CVE list
Severity
Component Name
libjpeg
Vulnerability Information:
CVE-2020-14152: In IJG JPEG (aka libjpeg) before 9d, jpeg_m
honor the max_memory_to_use setting, possibly causing exce
CVSS Score: 7.1
CVSS Version: 3.1
Applicable?
Yes
Severity?
High
Lower
Obtain fixed version and integrate into image
Obtain fixed version during next planned maintenance release and integrate into mage
Calendar

Notes on the above work flow

1. In some cases, CVEs for a component won't apply to the application they are used in because the affected functionality isn't used by the application. For example, the widely used OSS component OpenSSL provides many functions in addition to its core SSL/TLS functionality. If an application is using only one of these functions and not SSL/TLS it won't be affected by vulnerabilities in SSL/TLS. It is not feasible for an automated tool to detect this so this analysis must be performed manually.
2. When applicable critical CVEs are discovered by CodeSentry, it is critical to put in place a plan to release a patched software image as soon as a fix for the TPS is available because, as previously stated, attackers will immediately be aware of the vulnerability. The other implication of this is images should be scanned frequently for new vulnerabilities to help close the gap between when a TPS vulnerability is published and users of the TPS become aware of it.

8 Compliance with safety standards

IEC 62443 does not have any specific requirements for the security tools that are used for security verification and validation testing. However, the functional safety standards, IEC 61508 and ISO 26262 do have such requirements for tools used in product development and testing. The Grammatech CodeSonar tool has been certified with the tool requirements in these standards. These standards recognize that failures in these tools could lead to faults in the product that could lead to safety related failures. The same can be said from a security point of view that failures in these tools could lead to faults in the product that could lead to security related failures. In order to address these types of failures, the functional safety standards include an assessment of development tools in three different areas: the development process that is used to develop the tool, the validation testing that is done to ensure that the tool meets its specification, and field usage experience and history in the tool. These standards require that a rigorous development and validation process be used to create and test the tool, and compliance with these standards indicates that a tool follows such processes in their development and therefore is less likely to include faults that lead to safety and security related failures.

9 Conclusions

1. IEC 62443-4-1 and customers alike require identifying and addressing known classes of vulnerabilities in programming languages with the use of static source code analysis tools. The CodeSonar tool is well suited for meeting this requirement in today's common development methodologies and programming languages.
2. TPS offers many advantages but also some unique security considerations and challenges.
3. Customers and security standards alike are driving managing security in TPS (Third Party Software).
4. The CodeSentry and CodeSonar Binary tools can provide valuable input for selecting a TPS component or library.
5. Attackers and security researchers are constantly finding new CVEs in TPS. As a result, using a tool such as CodeSentry to facilitate frequent scanning for new CVEs becomes necessary to help keep up.

10 Reference Documents

The services delivered by *exida* were performed based on the following standards:

N1	IEC 62443-4-1	Secure product development lifecycle requirements