

FÓRUM DE SISTEMAS EMBARCADOS E IOT 2026

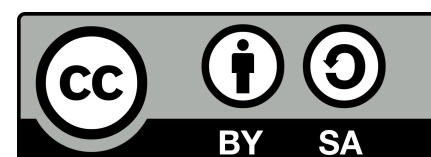
GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS: SBOM, CVES E O CYBER RESILIENCE ACT

Sergio Prado, Embedded Labworks
sergio.prado@e-labworks.com
<https://www.linkedin.com/in/sprado>



SOBRE ESTE DOCUMENTO

- Slides da palestra "Gestão de vulnerabilidades em sistemas embarcados: SBOM, CVEs e o Cyber Resilience Act".
- A versão mais atual dos slides está disponível no link abaixo:
<https://sergioprado.blog/talks/>
- Este documento é disponibilizado sob a licença Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0). Os termos completos desta licença estão disponíveis no link abaixo:
<https://creativecommons.org/licenses/by-sa/4.0/>



SOBRE O PALESTRANTE

- Mais de 30 anos de experiência em desenvolvimento de software para sistemas embarcados (Linux embarcado, Android embarcado, RTOS, etc).
- Consultor e instrutor na Embedded Labworks, com foco em arquitetura de sistemas embarcados, uso de sistemas operacionais, desenvolvimento de BSP e segurança.
<https://e-labworks.com>
- Contribuidor de software livre (U-Boot, Linux kernel, Buildroot, Yocto Project, etc).
- Blog e redes sociais:
<https://sergioprado.blog>
<https://www.linkedin.com/in/sprado>
<https://twitter.com/sergioprado>
<https://www.youtube.com/spradotube>



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

O PROBLEMA



VULNERABILIDADES EM SOFTWARE

- Uma vulnerabilidade é uma falha de software que alguém consegue explorar para obter do sistema algo que ele não deveria permitir, como ler dados de outro processo, ganhar privilégios, assumir o controle do dispositivo, interromper sua execução, etc.
- Nem todo defeito é uma vulnerabilidade!
 - O defeito comum quebra uma funcionalidade, enquanto a vulnerabilidade quebra uma garantia de segurança.
- As vulnerabilidades nascem de erro de programação, de falha no projeto de protocolos ou algoritmos, de configuração insegura entregue de fábrica, etc.
- O problema pode estar em código escrito por nós, por um fornecedor ou em um projeto open-source, e muitas vezes leva anos até alguém descobrir!



EXEMPLOS REAIS DE VULNERABILIDADES

Nome	Vulnerabilidade	Descrição
Mirai	Credenciais fracas	Malware/botnet que comprometia câmeras IP, DVRs e roteadores para uso em ataques DDoS
Heartbleed	Estouro de buffer	Falha no OpenSSL que permitia ler memória do processo remoto, expondo chaves, credenciais e outros dados sensíveis
KRACK	Falha de protocolo	Ataque contra o handshake do WPA2, permitindo decriptar qualquer informação trafegada no canal de comunicação WiFi
Dirty COW	Race condition	Falha no kernel Linux que permitia escalonamento local de privilégios



UMA NOVA VULNERABILIDADE É IDENTIFICADA...

- Você consegue responder estas perguntas?
 - Ela afeta o seu produto?
 - O componente vulnerável está realmente presente?
 - A vulnerabilidade pode ser explorada na configuração utilizada?
 - É necessário corrigir imediatamente?
- Na maioria das vezes, a resposta mais honesta para estas perguntas é "não sei".
- O objetivo desta palestra é transformar essas perguntas em um processo que consegue respondê-las.



POR QUE ISSO É DIFÍCIL EM SISTEMAS EMBARCADOS?

- Um único produto pode combinar bootloader, kernel (Linux, Zephyr, etc), software open-source, aplicações proprietárias, firmware bare-metal, bitstream de FPGA, etc.
 - Cada componente tem versão, configuração e ciclo de atualização próprios.
- Alguns destes componentes podem não ter sido escolhidos por você, e sim herdados do SDK do fabricante ou instalados por dependência de outros componentes.
- A maioria é compilada a partir do código-fonte, com patches e opções de configuração que mudam o que está realmente presente na imagem final.
- E o produto tem vida longa. Daqui a dez anos ainda estará em campo, muito depois do fim do suporte upstream de vários componentes.



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

POR QUE AGORA: NORMAS E O CYBER RESILIENCE ACT



NORMAS E REGULACOES

- [ISA/IEC 62443](#): requisitos de segurança para dispositivos e sistemas de controle e automação industrial.
- [ISO/SAE 21434](#): obriga fabricantes de veículos a implementar gestão de riscos de cibersegurança ao longo de todo o ciclo de vida, impactando ECUs e sistemas embarcados automotivos.
- [FDA](#): requisitos de cibersegurança para dispositivos médicos, incluindo a obrigação de fornecer a lista dos componentes de software junto com a submissão do produto.
- [EU Cyber Resilience Act \(CRA\)](#): regulação europeia que exige que produtos com elementos digitais atendam a requisitos mínimos de segurança.
- Todas convergem para exigências parecidas, incluindo o rastreamento do software do produto e o tratamento de vulnerabilidades ao longo do seu ciclo de vida.



CYBER RESILIENCE ACT (CRA)

- O CRA ([Regulamento \(UE\) 2024/2847](#)) define requisitos obrigatórios de segurança para produtos com elementos digitais colocados no mercado europeu.
- Aplica-se a quem vende na Europa, e não apenas para empresas europeias, alcançando fabricantes, importadores e distribuidores.
- Em vigor desde dezembro de 2024, com as obrigações entrando em etapas:
 - **11 de setembro de 2026:** reporte de vulnerabilidades ativamente exploradas e de incidentes graves, inclusive para produtos já no mercado.
 - **11 de dezembro de 2027:** todas as demais obrigações.
- Descumprir pode resultar em multas e na retirada do produto do mercado.



O QUE O CRA EXIGE NA PRÁTICA

- **Segurança por design:** colocar o produto no mercado sem vulnerabilidades conhecidas exploráveis e com configuração segura por padrão.
- **Inventário de componentes:** documentar o que existe dentro do produto, em uma SBOM legível por máquina.
- **Reporte de vulnerabilidades** ativamente exploradas e de incidentes graves à [ENISA](#) e ao CSIRT nacional:
 - Alerta inicial em 24 horas, notificação completa em 72 horas e relatório final 14 dias após a correção.
- **Correções de vulnerabilidades** durante o período de suporte (em regra, cinco anos), com correções gratuitas e distribuídas sem demora.



DO REQUISITO LEGAL AO PROCESSO DE ENGENHARIA

Obrigações do CRA	Prática de engenharia
SBOM em formato legível por máquina	Gerar a SBOM automaticamente no build
Identificar e documentar vulnerabilidades	Cruzar a SBOM com bases públicas de vulnerabilidades (CVEs)
Produto sem vulnerabilidades conhecidas exploráveis	Triagem documentada do que afeta e do que não afeta (VEX)
Reportar em 24h o que já é explorado	Saber em minutos se o produto é afetado
Distribuir correções de segurança	Mecanismo de atualização seguro



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

INVENTÁRIO E SBOM

O QUE É UMA SBOM

- A SBOM (*Software Bill of Materials*) é o inventário dos componentes de software que compõem um produto, mesmo conceito da lista de materiais que a engenharia de hardware usa há décadas.
- Para cada componente, registra pelo menos nome, versão, origem e licença, além dos relacionamentos entre eles (o que depende do quê, o que foi gerado a partir do quê).
- Precisa ser gerada automaticamente e legível por máquina, e não uma planilha preenchida à mão que ninguém consegue manter.
- A SBOM não é um documento de segurança por si só, mas serve de base para responder de forma automática se uma vulnerabilidade afeta ou não o produto.
- Também não é um artefato único, porque cada release gera a sua SBOM, que precisa ser arquivada junto com a imagem correspondente.



FORMATOS: SPDX E CYCLONEDX

- [SPDX](#) (ISO/IEC 5962) nasceu na Linux Foundation com foco em conformidade de licenças, sendo o formato gerado nativamente pelo Yocto Project e pelo Zephyr.
- [CycloneDX](#) (ECMA-424) nasceu na [OWASP](#) com foco em segurança da cadeia de suprimentos de software, sendo mais comum em ferramentas de análise de dependências e em aplicações web e corporativas.
- As duas especificações evoluíram para cobrir o mesmo escopo, incluindo a descrição de vulnerabilidades e de análise de impacto (VEX).
- Na prática, a escolha costuma ser ditada pela ferramenta que gera e pela ferramenta que consome, e não por uma diferença conceitual.
- Há conversores entre os dois, mas a conversão pode perder relacionamentos e identificadores. Prefira o formato nativo do seu ecossistema.



O QUE UMA SBOM ÚTIL PRECISA TER

- Versão exata de cada componente, e não apenas o nome.
- Identificadores padronizados ([CPE](#), [PURL](#)), evitando que o cruzamento com bases de vulnerabilidades vire uma busca por string, o que é exatamente a origem da maior parte dos falsos positivos e falsos negativos.
- Origem e patches aplicados, porque um componente com correção de segurança aplicada pela sua camada é diferente do código upstream de mesma versão.
- Relacionamentos entre os componentes, para distinguir o que foi realmente instalado na imagem do que apareceu apenas durante o build.
- Hashes dos artefatos, que permitem verificar se a SBOM corresponde mesmo à imagem que está em campo.



ONDE A SBOM ENTRA NO PROCESSO?

- **No build**, gerada automaticamente pelo build system, e não como uma tarefa manual no fim do projeto.
- **No release**, arquivada e versionada junto com a imagem, porque daqui a alguns anos você vai precisar saber o que existe dentro de cada versão que está em campo.
- **Em CI**, com o cruzamento de vulnerabilidades rodando a cada build e falhando quando aparece algo acima do critério definido.
 - Este cruzamento precisa ser feito periodicamente, reavaliando as SBOMs das versões já liberadas, já que as bases de vulnerabilidades mudam todos os dias.
- **Na entrega ao cliente**, cada vez mais frequente como requisito contratual, especialmente em mercados regulados (ex: [EUA \(EO\) 14028](#)).



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

VULNERABILIDADES E CVES



COMO UMA VULNERABILIDADE É PUBLICADA?

- CVEs (*Common Vulnerabilities and Exposures*) são identificadores padronizados para vulnerabilidades públicas (ex: [CVE-2014-0160](#)).
 - São atribuídos por [CNAs](#) (*CVE Numbering Authorities*), como a Linux Foundation, o projeto Zephyr e vários fabricantes.
- O registro é publicado na [CVE List](#) e enriquecido pela [NVD](#) com severidade (CVSS), produtos afetados (CPE) e referências.
 - Desde 2024 a NVD acumula um backlog de registros não enriquecidos, o que tornou comum combinar outras fontes, como a [EUVD](#) da ENISA e o [OSV](#).
- Tenha em mente que nem toda vulnerabilidade vira CVE, e nem todo CVE descreve corretamente o que é afetado.



CRUZANDO A SBOM COM OS CVES

- O cruzamento é sobre casar identificadores, verificando, para cada componente e versão da SBOM, se algum CVE declara aquela combinação como afetada.
- Ferramentas comuns nesse papel incluem a [sbom-cve-check](#) da Bootlin, que o Yocto Project adotou como mecanismo oficial, o [Grype](#), o [Trivy](#), o [cve-bin-tool](#) da Intel e o [Dependency-Track](#) da OWASP.
- O cruzamento deve acontecer a cada build em CI e, de forma recorrente, sobre as SBOMs das versões que já estão em campo.
- Automatize a execução e o armazenamento do resultado, para conseguir comparar duas análises e enxergar o que mudou desde a última vez.
- O resultado é uma lista de possíveis vulnerabilidades, não de problemas confirmados. É aí que começa a parte mais trabalhosa do processo!



O PROBLEMA DOS FALSOS POSITIVOS

- Uma imagem típica de Linux embarcado produz dezenas ou centenas de CVEs "em aberto" a cada análise, e a maior parte não representa risco real para o produto.
- As causas são quase sempre as mesmas:
 - **CPE** incorreto/genérico, que associa o CVE a outro produto de nome parecido.
 - Correção aplicada como backport, mantendo o número de versão original.
 - Funcionalidade vulnerável não habilitada na configuração de build.
 - Código vulnerável presente, mas inalcançável no produto.
- O efeito colateral é pior do que o ruído em si, porque quando toda análise devolve duzentos itens a equipe simplesmente para de olhar.
 - É aí que entra o VEX!



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

VEX E O PROCESSO DE TRIAGEM



DOCUMENTANDO O QUE NÃO SE APLICA COM VEX

- VEX (*Vulnerability Exploitability eXchange*) responde, de forma legível por máquina, se um CVE afeta ou não um produto específico, e por quê.
- Cada CVE recebe um estado (*not affected, affected, fixed, under investigation*), com justificativa padronizada quando não afeta (componente ausente, código vulnerável ausente ou inalcançável, mitigação existente, etc).
- Implementações mais comuns: [OpenVEX](#), [CycloneDX VEX](#), [CSAF VEX](#) e o [SPDX 3.x](#) (suporte nativo).
- No Yocto Project, a triagem fica na própria receita e vira o relatório VEX da imagem:

```
CVE_STATUS[CVE-2026-XXXXX] = "not-applicable-config: not enabled"
```

PRIORIZANDO O QUE CORRIGIR

- Nem tudo precisa ser corrigido imediatamente, e tratar todos os CVEs com a mesma urgência é a forma mais rápida de não tratar nenhum.
- O **CVSS** (*Common Vulnerability Scoring System*) mede severidade técnica, não risco, tanto que um CVSS 9.8 em um componente que o produto não expõe pode ser menos urgente do que um 6.5 no serviço de rede que fica aberto na internet.
- O **EPSS** (*Exploit Prediction Scoring System*) estima a probabilidade de a vulnerabilidade ser explorada nos próximos 30 dias, ajudando a separar o que é teórico do que é provável.
- O catálogo **KEV** (*Known Exploited Vulnerabilities*) da CISA lista o que já está sendo ativamente explorado, a mesma categoria que, no CRA, dispara o prazo de 24 horas.
- A isso se soma o contexto do produto, que só você conhece!



O PLANO DE CORREÇÃO

Análise	Ação
Não afeta o produto	Registrar no VEX durante a triagem
Afeta e já é explorado	Corrigir e reportar (24h segundo o CRA)
Afeta, alto risco	Corrigir no próximo release de segurança
Afeta, baixo risco	Corrigir no próximo release de manutenção



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

CORREÇÃO E ATUALIZAÇÃO SEGURA

PATCH, BACKPORT OU MITIGAÇÃO?

- Atualizar o componente para a versão corrigida é o caminho mais simples, mas nem sempre é viável, porque a nova versão pode quebrar a API, exigir um toolchain diferente ou puxar dependências que não cabem no produto.
- O backport do patch para a versão em uso preserva a compatibilidade, sendo um processo normalmente facilitado pelo build system.
 - O custo é que você passa a manter aquele patch.
- Mitigar sem corrigir o código também é uma resposta legítima, seja desabilitando a funcionalidade vulnerável, restringindo a interface exposta, reduzindo privilégios ou mudando a configuração padrão.
- Não corrigir é uma decisão válida, desde que seja registrada e justificada. É para isso que serve o VEX.



SISTEMA DE UPDATE SEGURO

- **Assinatura digital** dos artefatos, para garantir integridade e autenticidade.
- **Criptografia** do artefato, opcional, quando há requisito de confidencialidade ou propriedade intelectual.
- **Atomicidade e rollback operacional**, tipicamente com partições A/B, para que uma queda de energia no meio do update não deixe o dispositivo inoperante.
- **Anti-rollback**, para impedir a instalação de versões antigas já corrigidas.
- **Gestão de chaves** com PKI mínima, chave privada offline e separação entre desenvolvimento e produção.
- Entre as opções open-source, [RAUC](#) e [SWUpdate](#) em Linux embarcado, e o [MCUboot](#) em Zephyr e bare-metal.



FECHANDO O CICLO

- Todo o processo anterior (inventário, cruzamento, triagem, priorização) não serve para nada se a correção não chegar ao dispositivo que está em campo.
- O CRA fecha esse ciclo de forma explícita, ao exigir correções de segurança gratuitas e distribuídas sem demora durante todo o período de suporte do produto.
- Publicar a correção não é o mesmo que instalá-la, já que um dispositivo desligado, sem conectividade ou atrás de NAT continua vulnerável.
 - Acompanhe quanto da frota realmente atualizou!
- Acrescentar partições A/B a um produto que já está em campo é muito difícil, porque o mecanismo de atualização é decisão de arquitetura, tomada no início do projeto, e não quando o primeiro CVE crítico aparece.



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

NA PRÁTICA



LINUX EMBARCADO E YOCTO PROJECT (1)

- É o cenário mais favorável, porque o build system sabe exatamente quais componentes, versões e patches entraram na imagem e já fornece essa informação estruturada.
- No Yocto Project 6.0 (Wrynose), a classe *create-spdx* vem habilitada por padrão e gera a SBOM automaticamente (VEX habilitável via variável *SPDX_INCLUDE_VEX*).
- A [análise de vulnerabilidades](#) roda sobre essa SBOM, pela classe *sbom-cve-check*, que chama a [ferramenta de mesmo nome](#) da Bootlin, substituindo a antiga *cve-check*.
- A base de vulnerabilidades é uma receita com revisão fixa, então dois builds com os mesmos metadados darão o mesmo resultado.
 - Porém, a configuração recomendada troca a revisão fixa pela mais recente (a escolha entre repetir e atualizar a análise é do desenvolvedor).



LINUX EMBARCADO E YOCTO PROJECT (2)

- Em uma receita, *CVE_PRODUCT* é usado para indicar o nome do componente na base de dados do CVE (usa-se o nome da receita por padrão), evitando a divergência de nomes (ex: tiff vs libtiff).
- A variável *CVE_STATUS* pode ser usada para marcar um CVE com seu estado atual e uma justificativa:

```
CVE_STATUS[CVE-1970-0001] = "not-applicable-platform: it only affects Windows"
```

- Um patch que corrige um CVE é reconhecido sozinho, pelo nome do arquivo ou por uma linha no cabeçalho (ex: *CVE: CVE-2022-3341*).
- Ao final, sai a SBOM com as vulnerabilidades identificadas, e cada pacote com CVE em aberto vira um aviso no log do build.



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

DEMO: LINUX EMBARCADO E YOCTO PROJECT



ZEPHYR (1)

- O Zephyr gera SBOM em SPDX através do comando *west spdx*, que usa os metadados de build do CMake para saber exatamente quais arquivos entraram na imagem.
 - Basta habilitar *CONFIG_BUILD_OUTPUT_META*, compilar normalmente e rodar *west spdx* apontando para o diretório de build.
 - São gerados documentos separados para a aplicação, o Zephyr, os artefatos de build e os módulos, em SPDX 2.3 por padrão.
- A granularidade é por arquivo de código-fonte, o que é mais preciso do que uma lista de pacotes, mas também mais difícil de cruzar com as informações de CVEs, que são organizadas por produto e versão.



ZEPHYR (2)

- O projeto é um CNA desde 2017, ou seja, atribui os próprios CVEs e mantém uma equipe dedicada a tratar os relatos de segurança.
 - [Vulnerabilidades](#) só se tornam públicas depois de um embargo de no máximo 90 dias (30 dias para corrigir e 60 dias para distribuir a correção).
- O que falta, em comparação com o Yocto Project, é a verificação integrada ao build, porque o cruzamento com as bases de vulnerabilidades é feito por fora, a partir da SBOM.
- E há uma consequência do modelo de execução, já que o Zephyr é linkado junto com a aplicação em um único binário, corrigir qualquer CVE significa recompilar e redistribuir o firmware inteiro.



FIRMWARE BARE-METAL (1)

- Aqui não existe build system que saiba o que entrou na imagem, porque o SDK do fabricante é copiado para dentro do repositório, a HAL é versionada junto com a aplicação e as bibliotecas de terceiros muitas vezes nem têm versão.
- Ninguém consegue responder qual versão do stack TCP/IP, do driver de WiFi ou da biblioteca criptográfica roda no produto já vendido.
- Analisadores de binário como o *cve-bin-tool* e o *binwalk* ajudam a descobrir o que já está lá, mas não são capazes de reconstruir um inventário que nunca existiu.



FIRMWARE BARE-METAL (2)

- A solução está no processo de desenvolvimento, e não em ferramentas:
 - Tratar cada componente de terceiros como dependência versionada, e não como código copiado para dentro do projeto.
 - Manter o inventário à mão e atualizado a cada release, em CycloneDX, que é o formato mais aceito pelas ferramentas de análise.
 - Exigir do fornecedor a SBOM de todo binário entregue pronto, do SDK ao módulo de conectividade.
- É o cenário mais difícil, e muitas vezes o mais comum em produtos de baixo custo e alto volume.



COMPARANDO

Critério	Linux / Yocto	Zephyr	Bare-metal
Geração de SBOM	Sim	Sim	Não
Formato do SBOM	SPDX	SPDX	Manual (CycloneDX)
Verificação de CVE no build	Sim	Não	Não
Triagem versionada (VEX)	Sim	Externa	Externa
Granularidade	Pacote	Arquivo	Indefinida
Identificadores (CPE/PURL)	Bons	Parciais	Inexistentes
Esforço para começar	Baixo	Médio	Alto



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

CONCLUSÃO



POR ONDE COMEÇAR

- Comece pequeno, escolhendo o produto mais relevante da linha e colocando um processo de pé antes de tentar cobrir todo o portfólio.
- Gere a SBOM no build e archive com cada release (custa pouca configuração na maioria dos build systems).
- Faça a primeira análise e aceite o ruído, já que um produto nunca analisado pode retornar centenas de CVEs.
- Faça a triagem e registre as decisões em VEX via build system ou ferramenta externa como a [vexctl](#), impedindo o mesmo falso positivo de voltar na análise seguinte.
- Automatize, defina responsáveis, periodicidade e critérios de prioridade.
- Garanta um caminho de correção até o produto em campo.



PARA SE APROFUNDAR

- Introduction to SBOM management on embedded Linux
<https://sergioprado.blog/introduction-to-sbom-management-on-embedded-linux/>
- Cyber Resilience Act: Reporting obligations
<https://digital-strategy.ec.europa.eu/en/policies/cra-reporting>
- Yocto Project: Checking for vulnerabilities e Creating a SBOM
<https://docs.yoctoproject.org/security-manual/vulnerabilities.html>
<https://docs.yoctoproject.org/dev-manual/sbom.html>
- Zephyr: west spdx e processo de segurança
<https://docs.zephyrproject.org/latest/develop/west/zephyr-cmds.html>
<https://docs.zephyrproject.org/latest/security/index.html>



GESTÃO DE VULNERABILIDADES EM SISTEMAS EMBARCADOS

OBRIGADO! PERGUNTAS?

Sergio Prado, Embedded Labworks
sergio.prado@e-labworks.com
<https://www.linkedin.com/in/sprado>
<https://e-labworks.com>

