Apêndice - Augusto Stambassi Duarte

Durante a leitura e a apresentação do artigo sobre C++ e os problemas relacionados ao buffer overflow, alguns pontos despertaram especial interesse e merecem ser destacados. Primeiramente, chamou minha atenção a variedade de ferramentas existentes para a detecção automática de vulnerabilidades em código fonte, como o CppCheck e o Flawfinder. Até então, eu conhecia apenas superficialmente esses tipos de ferramentas, mas o artigo me motivou a explorá-las mais profundamente e considerar seu uso prático no desenvolvimento seguro de software.

Outro aspecto relevante abordado foi a utilização de métricas de software menos convencionais, como as métricas ODC (Orthogonal Defect Classification). A proposta dessas métricas, voltadas à classificação de defeitos com o objetivo de entender e melhorar o processo de desenvolvimento, mostrou-se uma abordagem bastante rica e complexa. A discussão sobre elas me levou a refletir sobre a importância de não apenas identificar falhas, mas também compreender sua origem e contexto, promovendo assim uma evolução mais consciente da qualidade do código.

Quanto à temática geral do artigo, considero que o tema escolhido é extremamente pertinente, especialmente diante da relevância contínua da linguagem C++ em sistemas de alto desempenho e da sua proximidade com a manipulação direta de memória. O foco na vulnerabilidade de buffer overflow segue sendo um tópico crítico na segurança de software e, portanto, de grande valor para qualquer programador ou engenheiro da computação.

No entanto, cabe também apontar algumas limitações no que diz respeito à metodologia adotada pelos autores. Em certos trechos, senti falta de uma explicação mais clara sobre os critérios de escolha das ferramentas analisadas e dos parâmetros utilizados nos testes apresentados. A falta de detalhamento compromete, em certa medida, a reprodutibilidade e a compreensão completa dos resultados. Uma abordagem metodológica mais transparente teria conferido maior solidez ao estudo.

De maneira geral, a leitura e discussão do artigo foram enriquecedoras, contribuindo para ampliar minha perspectiva tanto sobre as vulnerabilidades clássicas em C++ quanto sobre as ferramentas e métricas modernas aplicáveis à engenharia de software seguro.

Apêndice - Davi Cândido de Almeida

O desenvolvimento deste trabalho, tanto na preparação do seminário quanto na produção do estudo sobre o artigo *“Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects”*, me envolveu em conceitos práticos relacionados ao estudo de Linguagens de Programação (LPs). Pude observar como esses conceitos são aplicados na prática e como sua interação é fundamental para complementar minha formação acadêmica.

O estudo contribuiu significativamente para ampliar meu raciocínio lógico e para a construção de soluções mais robustas e teoricamente embasadas. O artigo analisado trouxe como foco central as vulnerabilidades de estouro de buffer em projetos C/C++. Dentre os temas abordados, destaco a forma como os autores ressaltam que, muitas vezes, a natureza dos problemas computacionais exige atenção especial por parte do programador, dada a dificuldade de detectar todas as possíveis falhas.

Linguagens como C e C++, que delegam ao programador o controle manual da memória e o gerenciamento de ponteiros, acabam tornando ainda mais desafiador o desenvolvimento de sistemas seguros e confiáveis. Influenciado por essa problemática, o artigo também discute um movimento recente voltado à criação de softwares mais seguros. Nesse contexto, linguagens como Rust e frameworks modernos surgem como alternativas para lidar com partes críticas do sistema, oferecendo mais segurança sem comprometer o desempenho.

Apesar disso, vale ressaltar que C++ dificilmente será substituído, considerando sua alta performance, maturidade e o vasto ecossistema já consolidado. O que se observa, portanto, é uma tendência de complementaridade entre as linguagens, onde o uso de novas tecnologias visa suprir fragilidades específicas de linguagens mais antigas, sem abrir mão de seus pontos fortes.

Por fim, a leitura crítica e a discussão coletiva do artigo reforçaram a importância da adoção de boas práticas de desenvolvimento seguro, da constante atualização dos profissionais da área e da integração de diferentes técnicas — automatizadas e manuais — para garantir a confiabilidade de sistemas cada vez mais complexos. A experiência foi enriquecedora e contribuiu significativamente para a minha formação como desenvolvedor, ao unir teoria, prática e uma visão crítica sobre a segurança em linguagens de programação.

Apêndice - João Pedro Torres

A leitura do artigo “Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects” foi bastante enriquecedora e trouxe reflexões importantes sobre segurança em linguagens de baixo nível como C e C++. Um dos pontos que mais me chamou atenção foi a aplicação da classificação ODC, que vai além da simples categorização de erros e busca entender a origem e o contexto das falhas. Essa abordagem mostrou-se extremamente útil para pensar em soluções mais estruturadas e eficazes.

Outro destaque foi a análise crítica das ferramentas de análise estática (SATs). Embora largamente utilizadas no desenvolvimento, ferramentas como CppCheck e Flawfinder demonstraram baixa eficácia na detecção de vulnerabilidades reais de buffer overflow. Isso evidencia a necessidade de combinar essas ferramentas com outras estratégias de verificação e de aprimorar suas regras para cobrir casos mais sutis, como verificações ausentes.

Também me chamou atenção a discussão sobre métricas de software (SMs). A princípio, pensei que métricas como complexidade poderiam indicar vulnerabilidades, mas o artigo mostrou que não há relação causal clara. Essa constatação reforça a importância de se avaliar o código com base em contexto e intenção, não apenas em números.

Nesse sentido, o artigo também me fez refletir sobre as características da linguagem C++, que, apesar de seu alto desempenho e controle direto sobre recursos do sistema, carrega consigo o ônus da responsabilidade manual por parte do programador, especialmente no que se refere ao gerenciamento de memória e uso de ponteiros. Essas características, embora poderosas, aumentam significativamente o risco de falhas como buffer overflows. Isso justifica o interesse crescente por linguagens mais seguras como o Rust, que oferece garantias de segurança em tempo de compilação e impede muitos erros comuns de memória sem abrir mão do desempenho.

Apesar de o C++ ainda ser largamente utilizado e insubstituível em muitos contextos, é evidente que linguagens como Rust vêm surgindo como alternativas viáveis e complementares para o desenvolvimento de sistemas mais robustos e seguros.

No geral, a leitura ampliou minha visão sobre segurança em software e sobre os limites das ferramentas atuais, além de reforçar a importância do pensamento crítico na engenharia de sistemas.

Apêndice - Lucas Carneiro Nassau Malta

O seminário apresentado, cujo foco principal recai sobre os erros de *buffer overflow* nas linguagens C e C++, mostrou-se ser um tema de grande relevância no cenário atual da cibersegurança. A experiência foi extremamente enriquecedora, permitindo-me explorar em detalhes um tópico crítico e compartilhar conceitos valiosos com meus colegas.

Um dos destaques durante a leitura do estudo foi a excelente contextualização do problema. Os autores iniciaram dando uma visão clara sobre as vulnerabilidades de software, destacando-as como falhas críticas que podem levar a ataques de segurança e danos severos, como a perda de dados confidenciais e prejuízos financeiros. Essa introdução foi crucial para ressaltar a importância do tema, especialmente no cenário pós-pandemia, onde o aumento do trabalho e serviços online intensificou a exposição a essas ameaças. A menção do significativo investimento em cibersegurança em 2021 (mais de 40% do orçamento de TI) reforçou a urgência da discussão.

Além disso, a pesquisa e a escolha de dados relevantes para o tema foram um diferencial. Fiquei particularmente impressionado com a estatística de que as linguagens C e C++ são responsáveis por 52% das vulnerabilidades em sistemas de código aberto – 46% em C e 6% em C++. Essa informação, por si só, já justifica a profunda análise do *buffer overflow*, que é o tipo mais frequente de vulnerabilidade nessas linguagens. Os dados do estudo Mend.io, que listam os *buffer errors* (CWE-119) como uma das principais vulnerabilidades em C++, solidificaram a base da nossa discussão, mostrando o impacto real desses erros.

Por fim, a boa definição de conceitos fundamentais, com destaque para o próprio *buffer overflow*, foi um dos meus pontos favoritos. Explicar essa vulnerabilidade de gerenciamento de memória – que ocorre quando o software lê ou escreve dados além do espaço alocado – de forma clara e concisa foi essencial. O ponto alto, para mim, foi a inclusão de exemplos de códigos reais, como o trecho vulnerável no Kernel Linux que utilizava a função *strcpy* para copiar nomes. Esse exemplo prático, mostrando como uma função aparentemente inofensiva pode gerar uma falha crítica, tornou o conceito tangível e demonstrou a importância de substituir funções inseguras por alternativas mais robustas, como *strlcpy*, ou adicionar verificações de limite.

Acredito que o seminário não apenas cumpriu seu objetivo de informar, mas também despertou um interesse maior nos colegas sobre a segurança em desenvolvimento de software.