# Article information:

Buffer Overflow Attack | OWASP Foundation
<https://owasp.org/www-community/attacks/Buffer_overflow_attack>

# Article summary:

1. Buffer overflow errors occur when memory fragments of a process are overwritten, causing exceptions and errors to occur.

2. These errors can result in unexpected application termination or abnormal behavior.

3. To prevent buffer overflow vulnerabilities, it is important to use safe equivalent functions that check buffer lengths and to review code for potential vulnerabilities.

# Article rating:

Appears moderately imbalanced: The article provides some useful information, but is missing several important points or pieces of evidence that would be required to present the discussed topics in a balanced and reliable way. You are encouraged to seek a more balanced perspective on the presented issues by exploring the provided research topics and looking at different information sources.

# Article analysis:

The article provides an overview of buffer overflow attacks, describing them as errors that occur when memory fragments of a process are overwritten intentionally or unintentionally. It explains that these errors can lead to exceptions, segmentation faults, and other unexpected application behavior.

The article includes two examples written in C language on a GNU/Linux system. The first example demonstrates a simple application that reads input from the user and copies it into a buffer without checking for buffer overflow. The second example is similar but includes calls to the printf() function before and after the buffer operation.

The article highlights the dangers of using unsafe functions like gets(), strcpy(), strcat(), sprintf(), (f)scanf(), getwd(), and realpath(). It recommends using safe equivalents or rewriting functions with safe checks implemented. It also suggests using compilers that can identify unsafe functions and logic errors.

Overall, the article provides useful information about buffer overflow attacks and offers recommendations for mitigating them. However, there are some potential biases and missing points to consider:

1. Biases: The article focuses primarily on the technical aspects of buffer overflow attacks and prevention measures. It does not discuss potential motivations or intentions behind such attacks, such as malicious intent or exploitation of vulnerabilities for personal gain.

2. Unsupported claims: The article claims that buffer overflows can lead to code execution if shellcode is injected. While this is true in some cases, it does not provide evidence or examples to support this claim.

3. Missing evidence: The article mentions the use of objdump to analyze ELF format binaries for exploiting buffer overflow errors but does not provide any specific examples or evidence of how this analysis can be used in practice.

4. Unexplored counterarguments: The article does not explore potential counterarguments against its recommendations, such as challenges in implementing safe equivalents or rewriting functions with safe checks due to time constraints or compatibility issues.

5. Partiality: The article only presents one side of the argument by emphasizing the dangers of unsafe functions and the importance of using safe equivalents or implementing safe checks. It does not provide a balanced perspective by discussing any potential benefits or trade-offs of using unsafe functions.

6. Promotional content: The article mentions the OWASP Code Review Guide as a reference without providing any critical analysis or evaluation of its content. This could be seen as promotional rather than providing objective information.

In conclusion, while the article provides valuable information about buffer overflow attacks and prevention measures, it has some biases, unsupported claims, missing evidence, unexplored counterarguments, partiality, and potentially promotional content. Readers should consider these factors when evaluating the article's content and recommendations.

# Topics for further research:

* Examples of real-world buffer overflow attacks and their consequences
* Motivations and intentions behind buffer overflow attacks
* Techniques for injecting shellcode and executing arbitrary code through buffer overflows
* Practical applications of objdump in analyzing ELF format binaries for buffer overflow exploitation
* Challenges and limitations in implementing safe equivalents or rewriting functions with safe checks
* Benefits and trade-offs of using unsafe functions in certain scenarios

# Report location:

<https://www.fullpicture.app/item/2876ca6152126451e59cf56172d1726e>