“Let Me Unwind That For You: Exceptions to Backward Edge Protection”

Submitted to Dr. Shaoquan Jiang

TOC

Introduction

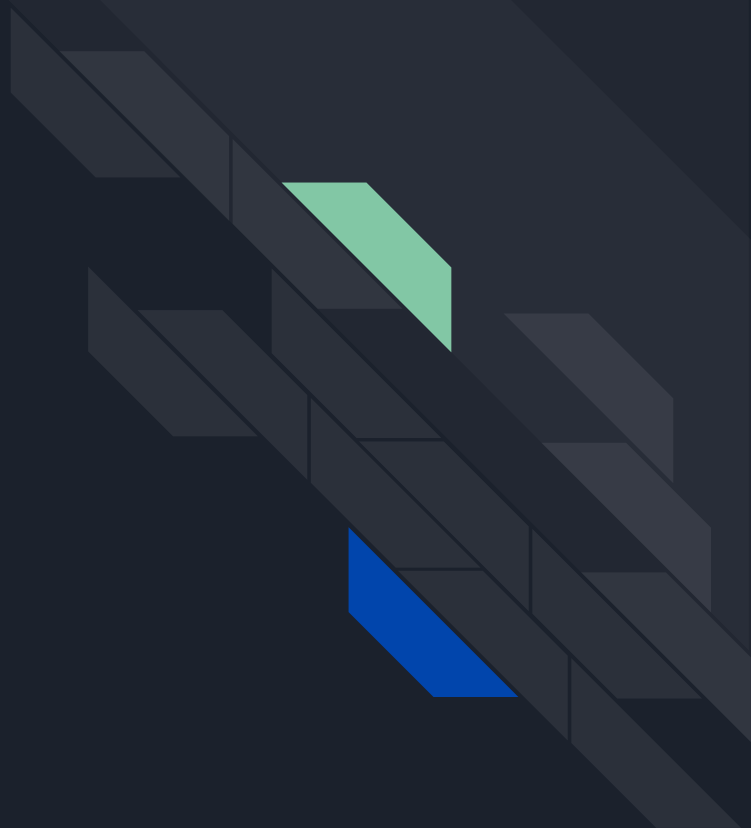
Paper Overview

Problem

Existing Protections

New Paradigms

Conclusion



Paper Overview

- The paper investigates backward-edge control-flow hijacking in software exploitation, challenging the assumption that attackers need to directly manipulate return addresses.
- It introduces Catch Handler Oriented Programming (CHOP), exploiting stack corruption to divert execution to controlled catch handlers, enabling powerful primitives like arbitrary code execution.
- The study demonstrates CHOP's applicability across platforms, surveys open-source packages to identify exploitable exception handlers, and concludes with proposed mitigations.
- This research uncovers novel attack opportunities, emphasizing the evolving landscape of control-flow manipulation and the necessity for adaptive defense strategies.

Exceptional control flow

- Exceptional control flow is a concept in computer science and programming that refers to the path a program takes in response to unexpected or exceptional events.
- Unlike the normal control flow, where a program executes sequentially or jumps to a specified section due to loops or conditions, exceptional control flow is triggered by events such as errors, exceptions, or external interrupts.
- For instance, consider a program that reads a file. If the file doesn't exist, instead of crashing, the program might take an exceptional control flow to handle this error, perhaps by displaying an error message to the user.

Importance

01 Robustness: It allows programs to handle unexpected situations gracefully without crashing. This is vital for software reliability.

02 User Experience: By providing meaningful error messages or alternative actions when something goes wrong, users are not left in the dark about what happened.

03 Security: Proper exception handling can prevent potential security vulnerabilities. For instance, without proper error handling, a program might reveal sensitive information in its error messages.

Protection Method

1. Stack Canaries:

What are they?
Stack canaries are small random values placed on the stack right before the return address.

How do they work?
When a buffer overflow attack tries to overwrite the return address (to redirect the program's flow), it has to overwrite the canary value first. Before a function returns, the program checks if the canary value has changed.

If it has, the program knows a buffer overflow has occurred and can take appropriate action, like terminating the process.

Why are they important?
Shadow stacks provide a robust mechanism against return-oriented programming (ROP) attacks, where attackers exploit existing code sequences in unintended ways.

Protection Method

2. Shadow Stacks:

What are they?
A shadow stack is a separate, protected stack that stores return addresses.

How do they work?
Every time a function is called, the return address is pushed onto both the regular stack and the shadow stack. When the function returns, the return address from the main stack is compared to the one on the shadow stack.

If they differ, a security violation has occurred.

Why are they important?
They provide a simple yet effective mechanism to detect stack-based buffer overflows, which are common attack vectors.

CHOP Attacks

What is CHOP Attacks?

Confused Handler-Oriented Programming (CHOP) is a novel attack paradigm that specifically targets the exception handling mechanisms in programming languages.

Exception handling is designed to manage unexpected or exceptional situations in code. However, CHOP exploits vulnerabilities within this system to achieve malicious objectives.

CHOP Attacks

Why is it significant?

Exception handling is a fundamental component of many modern software applications. If attackers can exploit this system, they can potentially gain unauthorized access or control over a significant portion of software infrastructure.

CHOP Attacks

Confusion Primitives

These are basic building blocks or tools that an attacker uses to induce confusion or misbehavior in a program's exception handling routines.

By leveraging confusion primitives, attackers can manipulate the normal flow of exception handling, causing the program to execute unintended sequences of code or operations.

Gadgets

In the context of CHOP, gadgets refer to small pieces or sequences of code that can be exploited by an attacker to perform specific tasks or operations. These are not malicious by themselves but can be used maliciously when combined in a particular sequence.

Gadgets are the "tools" that attackers string together to craft a CHOP attack. By chaining together multiple gadgets, attackers can create a sequence of operations that lead to a full-fledged exploit.

CHOP Attacks: Exploitation Strategies

The synthesis of confusion primitives and gadget capabilities in CHOP-style attacks enables diverse exploitation strategies. These strategies exploit vulnerabilities in exception handling mechanisms for arbitrary code execution and mitigation circumvention.

Golden Gadget: The "golden handler" technique within libstdc++ exemplifies direct exploitation. It diverts execution by manipulating the saved return address and controlling stack data, using an indirect call for potency.

Pivot-to-ROP: Attackers exploit functions with gadgets to perform traditional ROP attacks, bypassing backward-edge protections, even in the presence of stack canaries.

Data-Only Corruptions: Data-only attacks involve manipulating local stack frames, using cleanup or exception handlers to work on manipulated data. This approach can bypass ASLR without overwriting return addresses.

Cleanup-Handler Chaining: Attackers chain fake stack frames with cleanup handlers, creating sequences for multi-gadget use, similar to traditional ROP attacks.

SigReturn-to-ROP: By pivoting the unwinding process from the stack to an attacker-controlled buffer via SigReturn frame confusion, attackers execute ROP chains in controlled buffers. This tactic showcases versatile attack possibilities arising from CHOP-style techniques.

CHOP attack: Implementation and Evaluation

- Researchers employed a two-step procedure for data collection and analysis.
- Packages were downloaded from Debian Buster's main AMD64 repository.
- Extracted metadata included file names, associated packages, file types, and directory locations.
- Metadata stored in a PostgreSQL database for further analysis.
- Challenge: Accurately identifying exception-handling packages.
- Traditional dependency-based approach (using libstdc++) deemed inaccurate.
- Precise approach adopted: Focused on binary programs and shared libraries.
- Binary Ninja's capabilities used to identify programs with exception-handling characteristics.
- Refined method successfully overcame previous limitations in package identification.

CHOP attack: Implementation and Evaluation

- Researchers employed a comprehensive strategy to assess CHOP attack potential attack surface.
- Stack canaries were used to identify memory-corruptible functions; analysis extended to callee functions.
- CHOP gadgets' capabilities were deeply analyzed, identifying provided exploitation primitives.
- Static taint tracking assessed impact of controlled registers, categorized into taint sink types.
- Attacker model involved stack corruption, controlled backward edges, and callee-saved registers.
- Taint analysis engine created using Binary Ninja's HLIL representation, categorizing sink types.
- Feasibility scoring assigned to each gadget based on nested branches, loops, and function calls.
- Potential gadget stack frame manipulation attack surface prudently ignored.
- Study concludes by raising research questions about C++ software's exception handling and CHOP feasibility.

Mitigations

- Advanced hardening strategies are proposed for scenarios where attackers can circumvent stack canaries.
- Context-aware unwinding mechanism is suggested as a potential solution to enhance security.
- Modifying the unwinder to operate on shadow stack data instead of the original stack data is recommended.
- Authors acknowledge that these modifications could be extensive and might conflict with the Itanium C++ ABI.
- The Itanium C++ ABI is a specification for exception handling in C++ on Itanium and other platforms.
- Propose minimizing the use of 'catch-all' and lenient catch handlers to limit attacker opportunities.
- Recommend employing specific handlers for each exception and reducing the number of handlers in popular libraries.
- Suggest randomization-based defenses like function reordering and stack layout randomization.

Thank you!