Cpython JIT Copy-and-Patch Compiler 메커니즘의 보안 취약점 분석

조민혁, 정민서, 강수빈, 임정민, 유시환
단국대학교 모바일시스템공학과

KCC 2025
2025.07.02 (Wed)

Index

01. Introduction

Introduction

✓CPython?

: Python의 공식 구현체로 바이트코드를 인터프리팅하는 방식으로 동작하는 **C 기반 인터프리터**,
기본적으로 **JIT 컴파일을 포함하지 않으며** 인터프리팅 성능 보완을 위한 **연구 진행 중**

✓JIT Compile?

: Just-In-Time 컴파일은 실행 중 **바이트코드를 기계어로 변환**하여 인터프리팅보다 높은 성능 제공

✓Copy-and-Patch JIT?

: CPython에서 도입된 실험적 JIT 방식으로 반복 실행되는 바이트코드를 감지해 네이티브 코드로 패치하며
미리 복사한 코드 조각을 새로운 **메모리 영역에 배치**함으로써 실행 성능을 향상

=> "코드 조각을 메모리 영역에 배치해야 하기에 메모리 권한 변경 필요"

Introduction

본 논문의 필요성

“Python은 머신러닝, 개발, 클라우드 등 다양한 분야에서 활용되고 있음”

“CPython 런타임 환경에서 동작하는 Copy-and-Patch JIT의 보안 취약점 분석 필요”

Introduction

본 논문의 목표

“Copy-and-Patch JIT 구조가 공격 흐름의 진입점으로 악용될 수 있음을 입증한다”

“Copy-and-Patch JIT이 제공하는 성능 최적화가 보안성 저하로 이어질 수 있음을 밝힌다”

02. Related Work

Related Work

✓ **De Groef, Willem, et al. “Jitsec: Just-in-time security for code injection attacks.” Benelux Workshop on Information and System Security (WISSEC 2010).**

: JIT 컴파일을 활용하는 애플리케이션에서 발생할 수 있는 코드 인젝션 공격을 방지하기 위한 보안 메커니즘인 JITSec 제안

✓ **Chen, Ping, et al. “JITDefender: A defense against JIT spraying attacks.” Future Challenges in Security and Privacy for Academia and Industry: 26th IFIP TC 11 International Information Security Conference, 2011.**

: JIT Spraying 기법에 대한 방어를 목적으로 JITDefender라는 동적 보호 메커니즘을 설계 및 구현

✓ **Athanasakis, Michalis, et al. “The Devil is in the Constants: Bypassing Defenses in Browser JIT Engines” NDSS. 2015.**

: 브라우저 환경에서 JIT 컴파일러를 사용하는 가젯 없는 바이너리를 대상으로 런타임 시 동적으로 ROP 가젯을 생성할 수 있음을 입증

Related Work

관련 연구 정리

“선행 연구들은 공통적으로 JIT 환경이 제공하는 보안 취약점 분석”

“그러나 해당 연구들은 브라우저 JIT이나 범용 JIT 환경을 대상으로 연구 수행”

=> “CPython의 런타임 내 Copy-and-Patch JIT 메커니즘을 구체적으로 다룬 사례는 존재하지 않음”

03. Threat Model

Threat Model

✓Assumptions

- 공격자는 사용자 권한으로 Python 실행 환경에 접근 가능
- mprotect 시스템 콜 및 Python Internal API 호출 가능
- JIT 메모리 주소를 추론하거나 식별 가능

✓Attack Conditions

- JIT 메모리 영역에 RWX 권한 부여 가능
- W^X 메모리 보호 정책 우회 가능

=> “시스템 콜 제어 + 사용자 메모리 조작이 결합되면 임의 코드 삽입 및 실행 가능”

04. Attack Vector

Attack Vector

STEP 1) JIT 메모리 확보

- 반복 함수 호출로 JIT 컴파일러가 네이티브 코드 생성
- 메모리는 힙 또는 mmap 영역에 할당 (기본: RW 권한)

STEP 2) JIT 메모리 주소 식별

- CPython 내부 API 활용하여 JIT 코드 위치 추적
- trampoline 함수 진입 지점 식별 가능

Attack Vector

STEP 3) 권한 상승 (RW -> RWX)

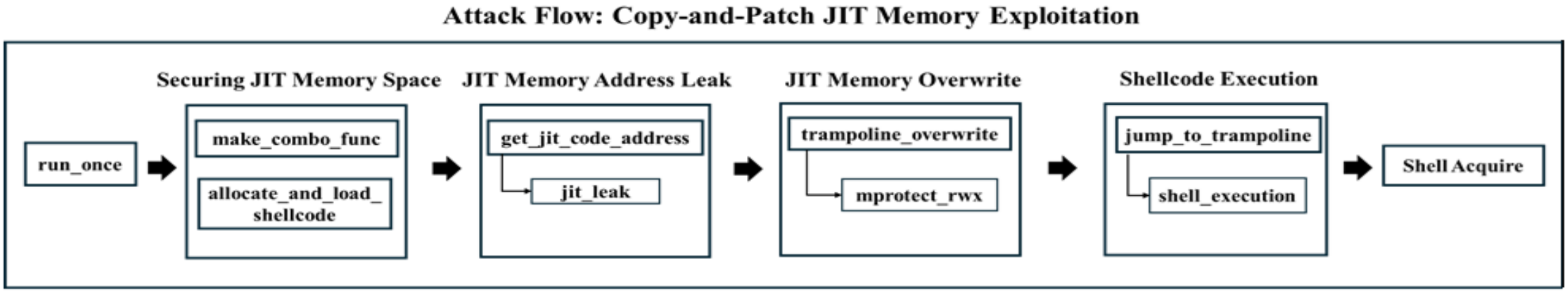
- ctypes 또는 C 모듈로 mprotect 시스템 콜 호출
- 실행 권한 추가로 W^X 정책 우회

STEP 4) 쉘 코드 삽입 및 실행

- 쉘 코드 또는 ROP Chain 삽입
- Trampoline 재작성으로 제어 흐름 전환

05. Proof Of Concept (PoC)

Proof of Concept (PoC)



✓실험 환경

- Architecture: AArch64
- OS: Ubuntu 22.04.05 LTS
- Python: Python 3.14 + Copy-and-Patch JIT 적용

Proof of Concept (PoC)

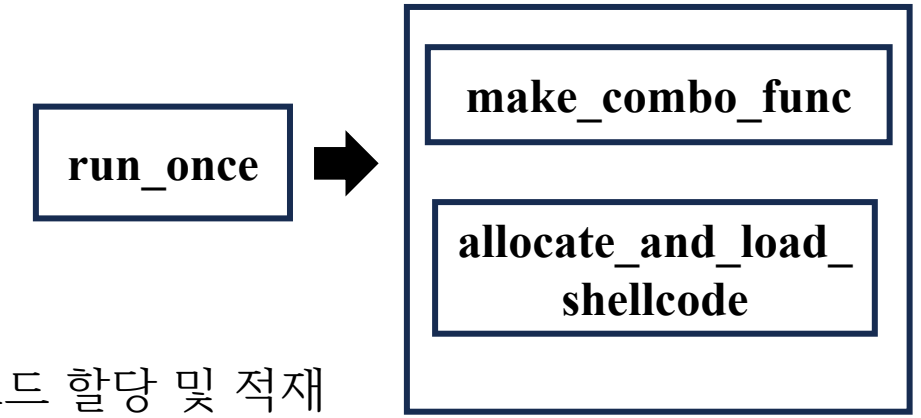
STEP 1) Securing JIT Memory Space

✓**run_once() → make_combo_func(), allocate_and_load_shellcode()**

- run_once()는 전체 공격의 시작점 (make_combo_func()를 호출)
- make_combo_func()는 반복 실행되는 바이트 코드 생성

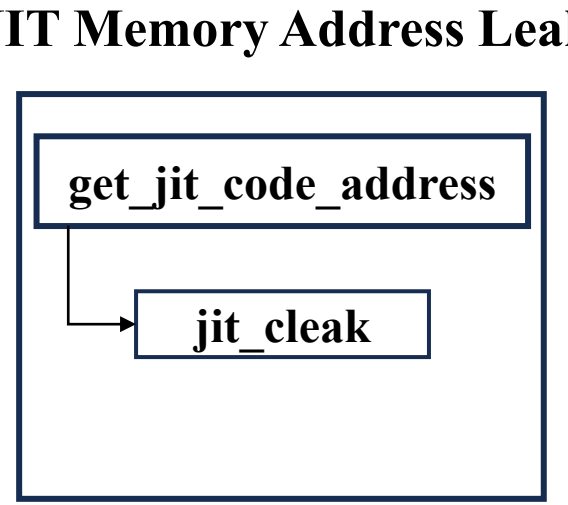
  > JIT 컴파일러가 이를 감지해 네이티브 코드 생성
- 네이티브 코드는 힙 또는 mmap된 메모리에 위치
- allocate_and_load_shellcode()는 이 영역에 공격자가 준비한 쉘코드 할당 및 적재

Securing JIT Memory Space



Proof of Concept (PoC)

STEP 2) JIT Memory Address Leak

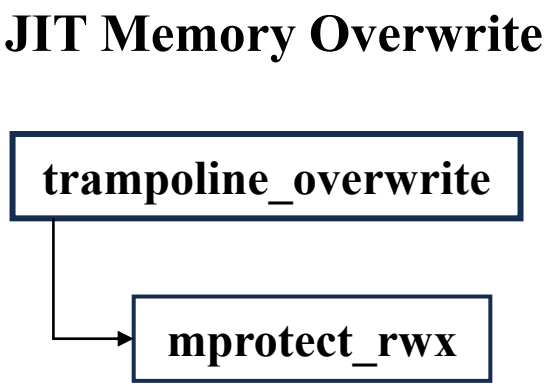


✓**get_jit_code_address() + jit_leak**

- get_jit_code_address()는 JIT 컴파일된 코드 블록의 실제 메모리 주소 유출
- 내부적으로 jit_leak 외부 모듈을 통해 CPython 내부 구조체 또는 객체 포인트 참조해 주소 확인

Proof of Concept (PoC)

STEP 3) JIT Memory Overwrite

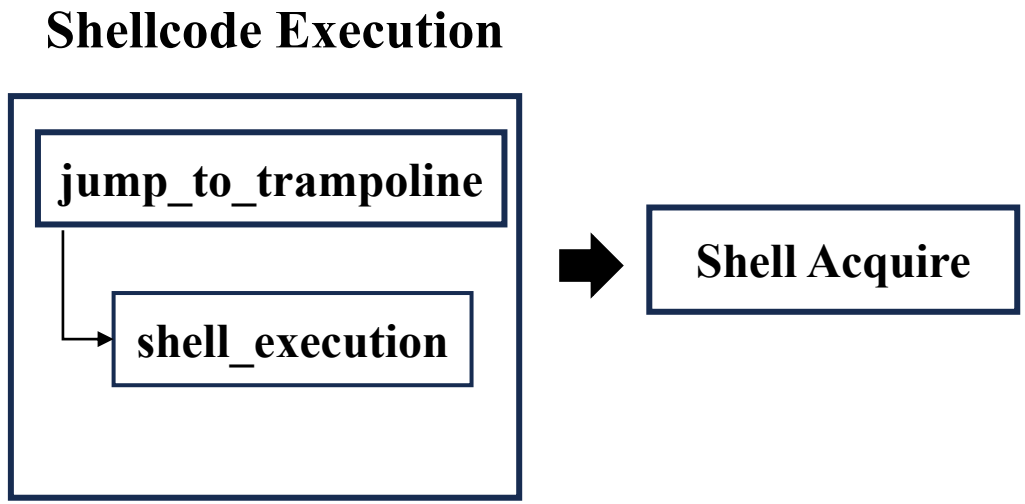


✓**trampoline_overwrite() + mprotect_rwx**

- trampoline_overwrite()는 유출된 주소를 기반으로 timepoline 코드를 덮어씀
- mprotect_rwx()는 해당 메모리 블록의 권한을 RW → RWX로 변경

Proof of Concept (PoC)

STEP 4) Shellcode Execution



✓**jump_to_trampoline() → shell_exection**

- Jump_to_trampoline()을 호출함으로써 트램폴린 코드 실행
- 해당 코드는 제어 흐름을 쉘코드가 위치한 메모리로 리디렉션
- 최종적으로 '/bin/sh' 명령을 실행하여 쉘 획득

Proof of Concept (PoC)

STEP 5) Shellcode Acquire

```
[*] code object @ 0xffffa3f05540
[*] executor found at offset 0
[*] executor @ 0xaaab1f98f820
[*] executor->jit_code @ 0xffffa3976000
[*] executor->jit_size @ 40960
#
```

06. Conclusion & Future Work

Conclusion & Future Work

✓Conclusion

: Python 3.13부터 지원되기 시작한 Copy-and-Patch JIT 컴파일러 메커니즘의 구조적 특성과 보안 취약점 분석

: 위협 모델과 공격 벡터를 기반으로 설계하여 공격 시나리오를 제안

: Proof of Concept(PoC)을 구현하여 실제 시스템 환경에서 검증

=> "쉘코드 실행하여 쉘을 획득함으로써 해당 공격이 리눅스 환경의 Cpython에서의 위협이 될 수 있음을 입증"

✓Future Work

: 제안한 공격 기법을 Intel x86 아키텍처에서도 재현 가능한지 확인

: 각 아키텍처 특성에 최적화된 ROP 및 메모리 조작 기법 탐색

=> "제안한 공격 기법을 일반화함으로써 보편적이고 효과적인 방어 기법 및 패치 적용의 필요성을 공동체에 제안"

References

1. H. Xu and F. Kjolstad, "Copy-and-patch compilation: a fast compilation algorithm for high-level languages and bytecode," in Proceedings of the ACM on Programming Languages, Vol. 5, OOPSLA, Chicago, IL, USA, 2021
2. https://docs.python.org/3/whatsnew/3.13.html
3. https://tonybaloney.github.io/posts/python-gets-a-jit.html
4. De Groef, Willem, et al. "Jitsec: Just-in-time security for code injection attacks." Benelux Workshop on Information and System Security (WISSEC 2010)
5. Chen, Ping, et al. "JITDefender: A defense against JIT spraying attacks." Future Challenges in Security and Privacy for Academia and Industry: 26th IFIP TC 11 International Information Security Conference, 2011
6. Athanasakis, Michalis, et al. "The Devil is in the Constants: Bypassing Defenses in Browser JIT Engines." NDSS. 2015.

Acknowledgement

본 연구는 2024년 과학기술정통신부 및 정보통신기획평가원의 SW중심대학사업 지원을 받아
수행되었음 (2024-0-00035)

Thank You

QnA