SZKOLENIE ŚREDNIO ZAAWANSOWANE

Wykrywanie i unikanie podatności w aplikacjach natywnych

XPL

Czas trwania: 3 dni

Przedstawienie metod wykorzystywanych podczas audytów bezpieczeństwa oraz testach penetracyjnych przeznaczonych do znajdowania i wykorzystania błędów w aplikacjach

Cele szkolenia

- Uczestnik szkolenia będzie umiał identyfikować podatności, wykorzystywać je do zakłócenia bądź przejęcia kontroli nad systemem oraz przygotować odpowiedni kod PoC (Proof of Concept)

Zalety

- Istotnym elementem szkolenia są warsztaty, podczas których uczestnicy przećwiczą poznane techniki na praktycznych przykładach z wykorzystaniem wiodących narzędzi i technik
- Praktyka przed teorią - wszystkie szkolenia technologiczne prowadzone są w formie warsztatowej. Konieczna teoria jest wyjaśniana na przykładzie praktycznych zadań
- Konkretne umiejętności - w ramach każdego szkolenia rozwijamy praktyczne umiejętności związane z daną technologią i tematyką
- Nauka z praktykami - wszyscy trenerzy na co dzień pracują w projektach, gwarantuje to dostęp do eksperckiej wiedzy i praktycznego know-how

Dla kogo?

- Szkolenie jest adresowane do specjalistów z zakresu bezpieczeństwa aplikacji, testerów wykonujących testy penetracyjne oraz programistów

Wymagania

- Od uczestników wymagana jest znajomość zasad działania komputera na poziomie architektury oraz biegłe posługiwanie się wybranym systemem operacyjnym (Windows lub Linux)
- Sugerowana jest też umiejętność rozumienia kodu napisanego w języku C/C++ oraz asemblera x86 i x86-64

- Pomocne jest wcześniejsze odbycie szkolenia z Inżynierii wstecznej (Reverse engineering)

Program

1. Wprowadzenie
    a. Linux
        -Zarządzanie pamięcią i formaty plików wykonywalnych
        -Narzędzia: GDB, objdump
    b. Windows
        -Zarządzanie pamięcią i formaty plików wykonywalnych
        -Narzędzia: IDA Pro, OllyDbg, WinDbg, Immunity Debugger
2. Wyszukiwanie błędów - podejście
    a. Oprogramowanie z i bez dostępnego kodu źródłowego
3. Kategorie błędów
    a. Przepełnienie stosu (stack overflow)
    b. Łańcuchy formatujące (format string)
    c. Przepełnienie sterty (heap overflow)
    d. Sytuacje wyścigu (race condition)
    e. Użycie zwolnionej pamięci (use after free)
    f. Użycie nieaktualnej wartości (time of check, time of use)
    g. Odmowa usługi
    h. Do czego można wykorzystać błędy - DoS vs. zdobycie kontroli
4. Metody wyszukiwania błędów
    a. Analiza statyczna
    b. Fuzzing statyczny
    c. Fuzzing dynamiczny - zwiększanie skuteczności fuzzera przez badanie pokrycia kodu i modyfikację próbek
    d. Generowanie błędów (fault injection)
    e. Pokrycie kodu i wnioski z pokrycia kodu
    f. Porównywanie zmian w kodzie oraz poprawek
5. Narzędzia i ćwiczenia praktyczne
    a. Lint, FindBugs, PMD oraz SonarQube
    b. Valgrind
    c. Zzuf, SPIKE, AFL
    d. IDA, IDASploit
    e. BinNavi
6. Kody powłoki (shellcode)
    a. Po co shellcode?
    b. Budowa kodu shellcode
        -Linux
        -Windows
7. Implementacja PoC (Proof of Concept)
    a. Samodzielny program exploit - przykłady w C i Python
    b. Moduł Metasploit - implementacja przykładowego exploita
8. Mechanizmy zabezpieczające na poziomie OS

   a. W^X
   b. Niewykonywalny stos
   c. Filtry - shellcode z wykorzystaniem Unicode/ASCII
   d. ASLR
9. Mechanizmy zabezpieczające na poziomie kompilatora i języka programowania
   a. Wykrywanie nadpisania stosu (stack canaries)
   b. Użyteczne wzorce i biblioteki
   c. Przykłady z użyciem GCC i Visual C++