Bezpieczeństwo systemów i sieci (BSS)

Ćwiczenie nr 7: Bezpieczeństwo aplikacji

dr inż. Krzysztof Cabaj
wersja 3.1, maj 2016

Cel

Celem ćwiczenia jest zapoznanie się ze szczegółami technicznymi błędu typu „buffer overflow" występującego w oprogramowaniu komputerów. Mimo wykrycia tego zagadnienia na początku lat 70', nadal jest to jedna z głównych metod pozwalających na zdobycie uprawnień na zdalnej maszynie. Dodatkowo słuchacze na przykładzie prostej podatnej aplikacji sieciowej zapoznają się z procesem wyszukania błędu oraz przygotowania exploit-a, który go wykorzystuje.

W ramach ćwiczenia należy znaleźć podatność w dostarczonej aplikacji sieciowej – program VulnSrv.exe dostarczony w postaci skompilowanej (wraz z plikiem symboli). W celu komunikacji z aplikacją i przygotowaniu exploita należy skorzystać z dostarczonego projektu dla środowiska Visual Studio – Exploit01.

Zadanie 1

Celem zadania jest wykrycie, jaki komunikat (jego nazwa) oraz jaka jego minimalna długość powoduje wystąpienie błędu, poprzez nadpisanie adresu powrotu.

W celu wykonania zadania należy uruchomić dwie kopie środowiska Visual Studio. W pierwszym należy otworzyć dostarczony projekt exploit-a a w drugim debugowanie skompilowanie wersji programu VulnSrv.exe.

Przygotowane do wysłania dane należy umieścić w tablicy exploit01. Po ich wysłaniu należy sprawdzić wynik zwrócony przez serwer. Dzięki analizie odpowiedzi należy znaleźć podatny fragment kodu. Dla ułatwienia kolejnych zadań należy wysyłać pakiet wypełniony bajtem o wartości dziesiętnej 65 - reprezentującym w kodzie ASCII znak A (duża litera A).

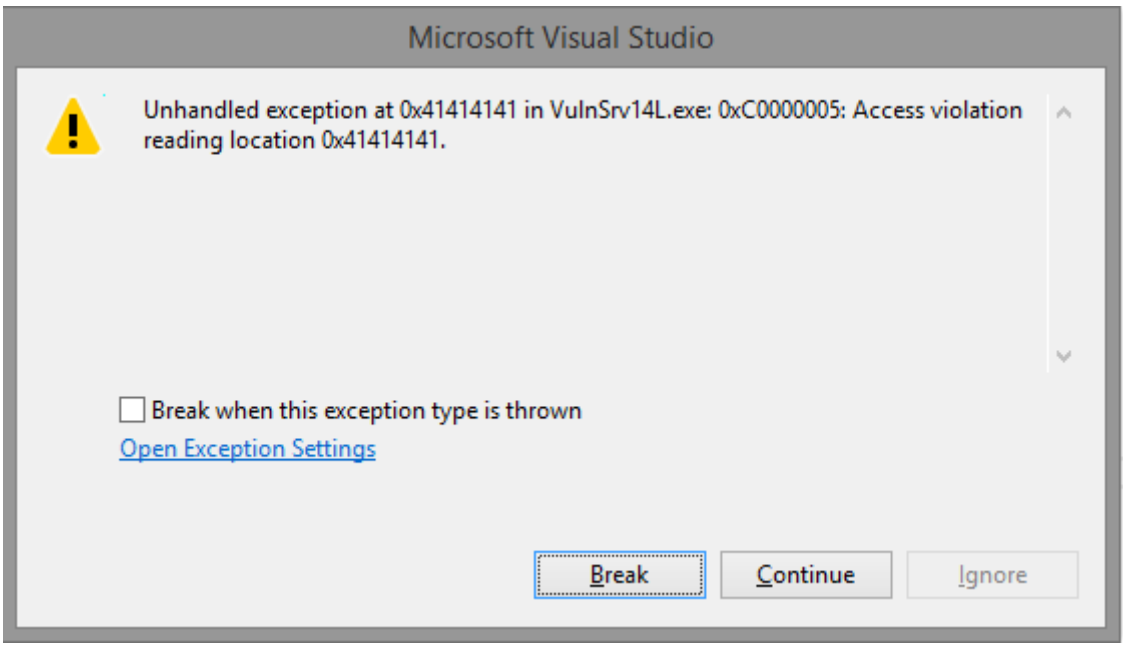
Uwagi!!!

W razie wystąpienia błędu związanego z tworzeniem socket-a należy uruchomić oba programy z dysku lokalnego (C:\Temp) a nie sieciowego.
Podatną funkcją nie jest „Aqq".
Każdy wysłany komunikat musi kończyć się znakiem $!!!
Proszę czytać wiadomości odsyłane przez serwer !!!.
Komunikaty należy wysyłać przy działającym programie VulnSrv.exe. Błąd w programie VulnSrv.exe może spowodować potrzebę ponownego uruchomienie programu.
Wysyłane dane nie powinny przekraczać 1024 bajtów (wysłanie większych spowoduje błąd ... ale nie w tym miejscu, które powinni Państwo zbadać podczas ćwiczeń).

W drugim środowisku należy uruchomić debugowanie skompilowanego programu VulnSrv.exe. W tym celu należy:

- Otworzyć program VulnSrv.exe w Visual Studio (File -> Open -> Project/Solution -> .exe),
- Rozpocząć debugowanie programu (Debug -> Start Debugging) .

Przy działającym pod kontrolą debuggera programie VulnSrv.exe należy wysłać przygotowane w projekcie Exploit01 dane i obserwować odpowiedź. W efekcie wysłania danych wykorzystujących podatność środowisko, w którym debugowany jest program VulnSrv.exe powinno wyświetlić następujący komunikat:



Zadanie do wykonania

Zidentyfikować nagłówek (nazwę) komunikatu oraz minimalną długość, która powoduje wystąpienie błędu typu „buffer overflow” oraz nadpisanie adresu powrotu.

Czy umiesz wytłumaczyć, dlaczego błąd dotyczy lokalizacji 0x41414141?

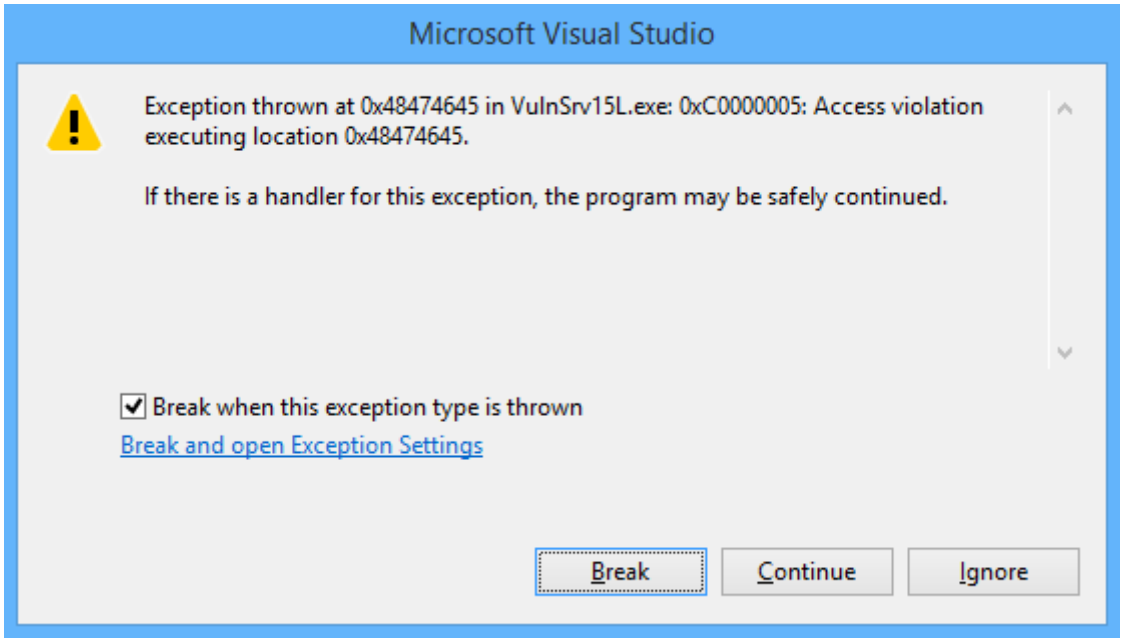
Zadanie 2

Celem tego zadania jest manipulacja nadpisywanym podczas ataku adresem powrotu z funkcji. W efekcie wyświetlany komunikat powinien zawierać informację, że błąd dotyczy określonej, deterministycznej i zamierzonej przez atakującego lokalizacji w pamięci.

Podpowiedź

Wykorzystując informację uzyskaną podczas wykonywania zadania 1 o minimalnej długości komunikatu powodującego wystąpienie błędu należy w tym rejonie (najlepiej rozpocząć kilka bajtów wcześniej) umieścić zmienny wzorzec (np. kolejne cyfry lub litery). Analizując adres, w którym wystąpił błąd możliwa jest bezpośrednia lokalizacja bajtów w wysyłanym pakiecie, które nadpisują adres powrotu z funkcji.

Przykładowy błąd obserwowany w środowisku Visual Studio dla exploit-a przygotowanego zgodnie z tą techniką.



Czy potrafisz w tablicy reprezentującej exploit-a wskazać konkretne 4 bajty, które nadpisują adres powrotu?

Zadanie do wykonania

Przygotować exploit-a, który spowoduje modyfikację adresu powrotu na adres 0x12345678, co zostanie potwierdzone odpowiednim komunikatem błędu obserwowanym podczas uruchomienia podatnego programu VulnSrv.exe w środowisku Visual Studio.

Zadanie 3

Celem tego zadania jest przygotowanie exploit-a, którego wysłanie spowoduje wykonanie przesłanego w nim kodu. Wysyłane wcześniej dane zawierają bajty reprezentujące w kodzie ASCII znak 'A', który dla procesora rodziny x86 jest dekodowany, jako instrukcja ***inc ecx***. W związku z tym możliwe jest wykonanie wysłanych danych, jako prawidłowego kodu procesora. W tym celu należy zlokalizować adres stosu, pod którym będzie ten kod kopiowany i odpowiednio podmienić nadpisywany adres powrotu z funkcji.

Poprzednie dwa zadania nie wymagały dokładnego analizowania kodu i wykorzystywały środowisku Visual Studio jedynie do przechwytywania komunikatu błędu z dokładnym adresem jego wystąpienia. W celu realizacji tego zadania należy zlokalizować w kodzie dostarczonego serwera podatną funkcję oraz przeanalizować jej zachowanie podczas analizy odebranych danych zawierających exploit.

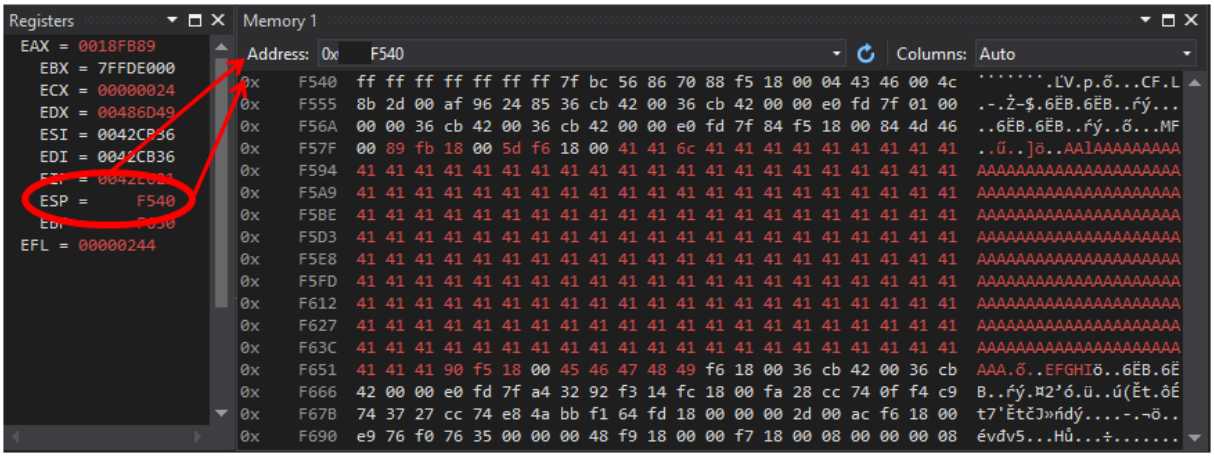
W tym celu należy:

- Otworzyć program VulnSrv.exe w Visual Studio (File -> Open -> Project/Solution -> .exe)
- Rozpocząć debugowanie od pierwszej instrukcji programu (Debug -> Step into)

- Zlokalizować podatną funkcję w programie VulnSrv.exe
- Przeanalizować zachowanie podatnej funkcji podczas analizy wysłanego exploit-a. Zlokalizować w tej sytuacji adres stosu oraz zawartość "górnej" części stosu i na tej podstawie określić adres, którym należy nadpisać adres powrotu z funkcji w celu wykonania tego zadania.

- Przydatne dodatkowe okna ułatwiające wykonanie zadania
    a. Rejestry (Debug -> Windows -> Registers)
    b. Pamięć (Debug -> Windows -> Memory -> Memory 1)
    c. Wynik desasemblacji aktualnie wykonywanego kodu (Debug -> Windows -> Disassembly)
    d. Postawione pułapki. Uwaga !!! Nowsze wersje środowiska czasem wyłączają pułapki podczas kolejnych uruchomień programu (Debug -> Windows -> Breakpoints)

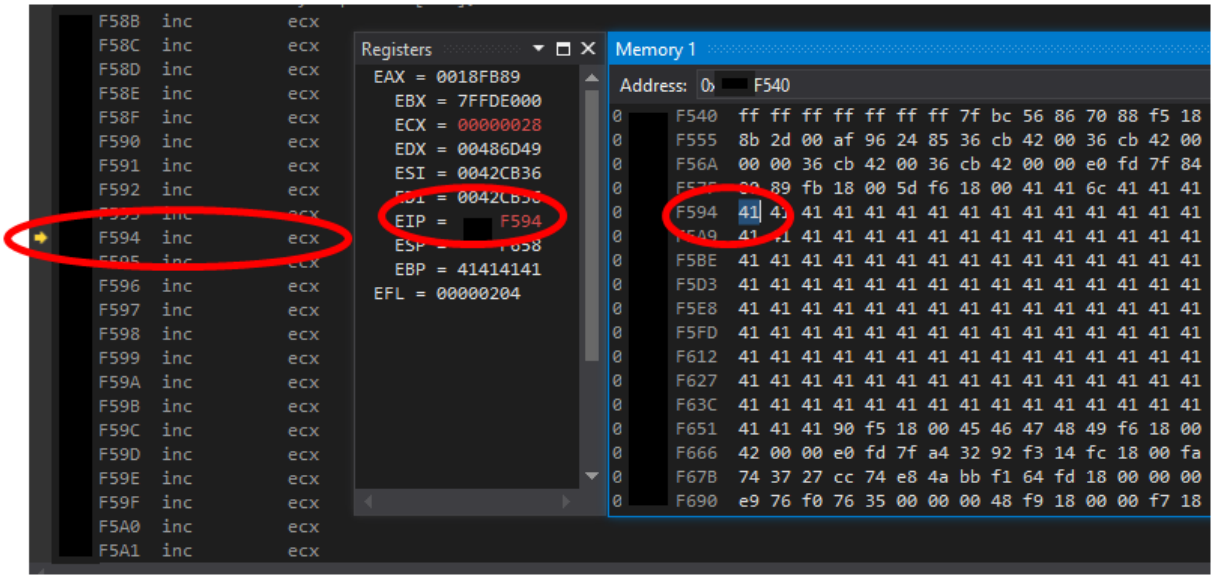
Widok rejestrów (patrz rejestr ESP) oraz pamięci pod tym adresem tuż przed wykonaniem instrukcji ret znajdującej się na końcu podatnej funkcji. Widoki te umożliwiają znalezienie odpowiedniego adresu w pamięci, który powinien nadpisać oryginalny adres powrotu z podatnej funkcji.



Zadanie do wykonania

Przygotować exploit-a, którego wysłanie spowoduje wykonanie wysłanych w nim instrukcji ***inc ecx***. Zaprezentować prowadzącemu pod kontrolą debuggera skok do przesłanego kodu oraz jego wykonanie.

Poniższy zrzut ekranu prezentuje efekt działania exploit-a. Rejestr EIP zawiera adres kolejnej instrukcji do wykonania. Na widoku zdeassemblowanego kodu widać instrukcję (***inc ecx***) przewidzianą do wykonania (żółta strzałka przed nią). Widok pamięci pokazuje, że jest to wysłany w exploit-cie bajt reprezentujący znak 'A'.



Zadanie 4 - dodatkowe

Celem tej części ćwiczenia jest przygotowanie exploit-a, który uruchamia przykładowo program ***calc.exe***.

W tej części ćwiczenia należy skorzystać z możliwości manipulowania nadpisywanym adresem powrotu z funkcji i spowodować wykonania własnego kodu. W celu uruchomienia programu można posłużyć się, po uprzednim uaktualnieniem odpowiednich adresów, kodem:

```
0xbc,0x00,0xF4,0x12,0x00, //mov esp,0x0012F400 -> rezerwacja pamięci, poniżej danych
0x68,0x64,0xf5,0x12,0x11, //push 0x1112f564 -> adres parametru
0xBB,0xC2,0xC6,0x22,0x00,//mov ebx, 0022C6C2 -> adres funkcji
0xFF,0xD3,//call ebx -> wywołanie funkcji
```

Wywoływana funkcja systemowa, umożliwiająca uruchomienie zadanego programu jest wykorzystywana w podatnej aplikacji do wyświetlenia nazwy i wersji systemu operacyjnego.

Ponieważ kod atakującego zostaje umieszczony na stosie, wywołanie funkcji systemowej może go nadpisać. Z tego powodu pierwsza wywoływana instrukcja ustawia wartość adresu wierzchołka stosu w taki sposób, aby znajdował się on poniżej (miał mniejszy adres) niż najmniejsze wykorzystywane podczas wykonania exploit-a instrukcje lub wykorzystywane dane. Kolejna instrukcja umieszcza na stosie parametr wywołania funkcji - adres, pod którym po przesłaniu exploit-a znajdzie się ciąg znaków zawierający nazwę wywoływanego programu, przykładowo ***calc.exe***. Ciąg ten należy umieścić w dowolnym, znanym miejscu wysyłanego exploita. Dwie ostatnie instrukcje służą do wykonania skoku do funkcji systemowej. Pierwsza instrukcja umieszcza w rejestrze ***ebx*** adres funkcji, a druga wykonuje skok do funkcji o adresie znajdującym się w rejestrze ***ebx***. Taki zestaw instrukcji wykorzystany jest ze względu na pewne problemy związane z bezpośrednim skokiem pod wskazany adres.

Zadanie do wykonania

Przygotowanie exploit-a dla aplikacji VulnSrv w dostarczonym projekcie Exploit01, którego wysyłanie spowoduje uruchomienie programu calc.exe.