Pisanie exploitów, które działają na wolności

Każdy błąd ma swoją historię. Robak rodzi się, żyje, a następnie umiera, często nie będąc nigdy odkrytym ani wykorzystanym. Dla hakera każdy błąd jest doskonałą okazją do stworzenia exploita, magicznego zaklęcia, które zamienia każdą wrażliwą ścianę w drzwi. Ale co innego stworzyć zaklęcie, które działa w laboratorium, a zupełnie co innego stworzyć takie, które działa w elektrycznej dżungli, jaką jest nowoczesny Internet. Ten rozdział koncentruje się na tworzeniu exploitów, które można z powodzeniem wykorzystać na wolności.

Czynniki zawodności

Ta część opisuje różne powody, dla których twój exploit może nie działać niezawodnie w środowisku naturalnym. Pamiętaj, że chociaż istnieje wiele powodów, dla których twój exploit nie działa, jak mówi Anakata, „nawet ślepy kurczak od czasu do czasu znajduje nasionko".

Magiczne liczby

Niektóre luki w zabezpieczeniach, takie jak przepełnienie stosu RealServer, nadają się do niezawodnego wykorzystania. Inne, takie jak sterta dtlogin double-free, są prawie niemożliwe do niezawodnego wykorzystania. Jednak nie można wiedzieć, jak niezawodny możesz zrobić z danego exploita, dopóki go nie wypróbujesz. Ponadto wykorzystywanie coraz trudniejszych podatności to jedyny sposób na poznanie nowych technik. Samo czytanie o technice nigdy tak naprawdę nie da ci niezbędnej wiedzy o tym, jak korzystać z tej techniki. Z tych powodów zawsze powinieneś dokładać wszelkich starań, aby Twoje exploity były jak najbardziej niezawodne. W niektórych przypadkach będziesz mieć doskonale dobry exploit, który działa przez 100 procent czasu w laboratorium, ale tylko przez 50 procent czasu w środowisku naturalnym, i będziesz musiał przepisać go od nowa, aby ulepszyć jego działanie. niezawodność w realnym świecie. Kiedy tworzysz pierwszego exploita dla określonej luki, może on działać tylko na twoim komputerze. Jeśli tak, prawdopodobnie zakodowałeś w nim kilka ważnych rzeczy, najprawdopodobniej adres zwrotny lub adres geteip. Kiedy możesz nadpisać wskaźnik funkcji lub przechowywany adres powrotu, będziesz musiał gdzieś przekierować wykonanie - ta lokalizacja jest prawdopodobnie zależna od wielu czynników, z których tylko niektóre kontrolujesz. Na potrzeby tego rozdziału tę sytuację nazwiemy wykorzystaniem jednego czynnika. Podobnie możesz mieć miejsce w swoim exploicie, w którym znajduje się wskaźnik do ciągu. Program docelowy używa tego wskaźnika, zanim uzyskasz kontrolę nad wykonaniem. Aby skutecznie przejąć kontrolę, konieczne może być ustawienie tego wskaźnika (część ciągu ataku) w nieszkodliwym miejscu w pamięci. Ten krok dodałby dodatkowy czynnik, o którym musiałbyś wiedzieć, aby skutecznie wykorzystać swój cel. Większość łatwych exploitów to exploity jedno- lub dwuskładnikowe. Na przykład podstawowe przepełnienie stosu zdalnego to zazwyczaj jeden czynnik wykorzystujący lukę - wystarczy tylko zgadnąć, adres twojego shellcodeu w pamięci. Ale w miarę postępów w przepełnianiu sterty, które są zazwyczaj dwuskładnikowymi exploitami, musisz zacząć szukać sposobów, które pomogą zmniejszyć poziom chaosu w systemie.

Wersjonowanie

Jednym z ważnych problemów, z jakimi się spotkasz podczas uruchamiania exploita na wolności, jest to, że rzadko będziesz wiedzieć, co jest załadowane na drugą maszynę. Być może jest to komputer z systemem Windows 2000 Advanced Server, taki jak ten w twoim laboratorium, a może korzysta z ColdFusion i przemieszcza pamięć. A może jest załadowany uproszczonym chińskim systemem Windows 2000. Może zawierać poprawki wykraczające poza najnowszy dodatek Service Pack; niektóre z nich mogły zostać zainstalowane ręcznie lub nawet nieprawidłowo. Możliwe jest również, że zdalny system jest komputerem z Linuksem działającym na Alfie lub że system jest urządzeniem SMP, co może

mieć wpływ na sposób, w jaki działa serwer, który atakujesz. Wiele publicznych exploitów Microsoft RPC Locator nie działa na komputerach SMP lub komputerach z procesorami Xeon, które według systemu Windows są urządzeniami z dwoma procesorami. Tego rodzaju problemy są bardzo trudne do zdalnego śledzenia. Ponadto, gdy uruchomisz exploity związane z korupcją sterty, będziesz mieć problemy z powstrzymaniem innych osób przed korzystaniem z usługi podczas uszkadzania sterty. Innym częstym problemem związanym z przepełnieniem sterty jest to, że nadpisują one wskaźniki do funkcji, które są zależne od konkretnej wersji libc. Ponieważ każda wersja Linuksa ma nieco inną bibliotekę libc, oznacza to, że exploit musi być specyficzny dla niektórych dystrybucji Linuksa. Niestety, żadna dystrybucja Linuksa nie ma większości udziałów w rynku, więc zakodowanie tych wartości na sztywno w exploity nie jest tak łatwe, jak w przypadku Windowsa lub komercyjnych komputerów Unix. Należy pamiętać, że wielu dostawców udostępnia wiele wersji Uniksa pod ten sam numer wersji. Twój dysk CD z systemem Solaris 8 będzie się różnić od innego dysku CD z systemem Solaris 8, w zależności od tego, kiedy każdy z nich został zakupiony. Twoja wersja może zawierać łatki, których nie ma na innej płycie CD i na odwrót.

Problemy z kodem powłoki

Niektórzy programiści spędzają tygodnie na pisaniu swoich shellcodów. Inni używają spakowanych shellcodów z Packetstorm . Jednak bez względu na to, jak zaawansowany jest twój shellcode, nadal jest to program napisany w języku asemblerowym i wykonywany w niestabilnym środowisku. To znaczy

że sam shellcode często jest punktem niepowodzenia. W laboratorium Ty i Twoja skrzynka docelowa znajdujecie się w tym samym koncentratorze Ethernet. Jednak na wolności twój cel może znajdować się na innym kontynencie, pod kontrolą kogoś, kto założył własną sieć zgodnie z własnymi zachciankami. Może to oznaczać, że ustawili swoje MTU na 512, że blokują ICMP, że przekierowali porty IIS do swojego komputera z systemem Windows ze swojej zapory linuksowej lub że mają filtry wychodzące na zaporze ogniowej lub inne komplikacje. Podzielmy problemy z shellcodem na następujące kategorie.

Związane z siecią

MTU (maksymalna jednostka transmisji) lub problemy z routingiem mogą stanowić problem podczas uruchamiania kodu powłoki. Czasami zaatakujesz jeden adres IP i oddzwoni do ciebie z innego adresu IP lub interfejsu. Częstym problemem jest również filtrowanie ruchu wychodzącego. Twój shellcode powinien zostać czysto zamknięty, jeśli nie może do Ciebie wrócić z powodu filtrowania. Możesz chcieć dołączyć kod powłoki UDP lub ICMP wywołania zwrotnego.

Związane z przywilejami

W systemie Windows określony wątek może działać bez uprawnień wymaganych do załadowania ws2_32.dll. Zwykłym rozwiązaniem jest kradzież wątku, w którym się znalazłeś, założenie, że ws2_32.dll jest już załadowany lub wywołanie RevertToSelf(). W niektórych wersjach Linuksa (SELinux itd.) możesz napotkać ten sam rodzaj problemów z uprawnieniami. W niektórych rzadkich przypadkach nie będziesz mógł wykonywać połączeń przez gniazdo lub nasłuchiwać na porcie. W takich przypadkach możesz chcieć zmodyfikować przepływ wykonywania oryginalnego programu (na przykład wyłączyć normalne uwierzytelnianie procesu docelowego, aby umożliwić sobie dalszą manipulację, zmodyfikować plik, z którego odczytuje, dodać do listy użytkowników itd.) lub znaleźć jakiś sposób, aby Twój kod powłoki mógł wykorzystać swój dostęp bez kontaktu z zewnątrz.

Związane z konfiguracją

Błędna identyfikacja systemu operacyjnego może spowodować umieszczenie w exploitie nieprawidłowego kodu powłoki lub adresu zwrotnego. Trudno jest zdalnie określić Alpha Linux od SPARC Linux; mądrze może być po prostu spróbować obu. Jeśli proces docelowy jest chrootowany, /bin/sh może nie istnieć. To kolejny dobry powód, aby nie używać standardowego kodu powłoki exeve(/bin/sh). Czasami baza stosu zmienia się w zależności od tego, który procesor atakujesz. Ponadto nie wszystkie instrukcje działają na każdym typie procesora. Być może twój cel ma na przykład stary układ Alpha, a testowałeś swój kod powłoki tylko na nowym. A może nowa maszyna SGI, którą atakujesz, ma dużą pamięć podręczną instrukcji, która nie jest czyszczona podczas ataku.

Powiązane z identyfikatorem hosta

chroot, LIDS, SELinux, jail() BSD, gresecurity, Papillion i inne warianty motywu mogą powodować problemy w shellcodezie na wielu poziomach. Ponieważ te technologie stają się coraz bardziej popularne, spodziewaj się, że będziesz z nimi radzić sobie w swoim shellcodezie. Jedynym sposobem, aby dowiedzieć się, czy wpłyną one na Ciebie, jest ich zainstalowanie i samodzielne przetestowanie.

Okena i Entercept zarówno przechwytują wywołania systemowe, jak i wykonują profilowanie w oparciu o to, jakie wywołania systemowe zwykle wykonuje ta aplikacja. Dwa sposoby na pokonanie tego profilowania to modelowanie normalnego zachowania aplikacji i próba pozostania w tym zakresie lub próba pokonania samego podpinania się wywołań systemowych. Jeśli masz exploit jądra, teraz masz szansę użyć go bezpośrednio z kodu powłoki.

Powiązane z wątkiem

W przypadku przepełnienia sterty inny wątek może się obudzić, aby obsłużyć odpowiedź, spróbować wywołać free() lub malloc(), a następnie zawiesić proces, gdy stwierdzi, że sterta jest uszkodzona. Inny wątek może obserwować Twój wątek, aby dowiedzieć się, czy zakończy się na czas. Ponieważ przejąłeś swój wątek, może cię to zabić, aby odzyskać proces. Sprawdź wątki pulsu z kodu powłoki i spróbuj emulować ich sygnały, jeśli to możliwe. Twój exploit może polegać na adresach zwrotnych, które są ważne tylko dla jednego wątku, co jest zwykle spowodowane słabymi testami z Twojej strony.

Środki zaradcze

Istnieje wiele sposobów, w jakie twoja próba wykorzystania może stać się niestabilna lub całkowicie nie zadziałać. Istnieje jednak wiele sposobów na zrekompensowanie tych problemów. Należy pamiętać, że piszesz aplikację, która nigdy nie powinna była istnieć - exploit. Exploity istnieją tylko z powodu błędów w innym oprogramowaniu. Dlatego tworzenie niezawodnych exploitów nie jest kwestią prostego korzystania z inżynierii oprogramowania. Przez cały czas trwania procesu powinieneś nieustannie próbować znaleźć alternatywne metody rozwiązywania pojawiających się problemów. W razie wątpliwości pomyśl: „Co zrobiłby John McDonald?” Oto fragment z Phrack (numer 60, grudzień 2002), który przedstawia jego filozofię. Pamiętaj o jego słowach, gdy wpadniesz w kłopoty. PHRACKSTAFF: Znalazłeś sporo błędów w przeszłości i stworzyłeś dla nich kod exploita. Niektóre luki wymagały nowych koncepcji kreatywnego wykorzystania, które nie były wówczas znane. Co skłania Cię do kwestionowania wykorzystania skomplikowanych błędów i jakich metod używasz? John McDonald: Cóż, moje motywacje zdecydowanie się zmieniły z biegiem czasu. Mogę podać kilka dodatkowych powodów, które kierowały mną w różnych momentach mojego życia, a są to zarówno egoizm, jak i altruizm. Ale myślę, że tak naprawdę sprowadza się to do przymusu wymyślenia tego wszystkiego. Jeśli chodzi o metody, staram się być nieco systematyczny w swoim podejściu. Przeznaczam sporą porcję czasu na samo przeczytanie programu, próbując wyczuć jego architekturę oraz sposób myślenia i techniki jego autorów. To również wydaje się pomagać w przygotowaniu mojej podświadomości. Lubię zaczynać od niższych warstw programu lub systemu i szukać wszelkiego rodzaju potencjalnych

nieoczekiwanych zachowań, które mogłyby przenikać w górę. Udokumentuję każdą funkcję i przeprowadzę burzę mózgów o wszelkich potencjalnych problemach, które z nią zobaczę. Od czasu do czasu robię sobie przerwę od dokumentacji i robię znacznie przyjemniejszą pracę polegającą na prześledzeniu niektórych moich teorii, aby sprawdzić, czy się sprawdzą. Jeśli chodzi o pisanie exploitów, zazwyczaj staram się ograniczyć lub wyeliminować liczbę rzeczy, które należy odgadnąć. Kiedy twój exploit jest prawie ukończony, ale wydaje się, że się nie rozwija, stań się kimś innym. Napisz swój exploit w stylu „Halvar" - spędź dużo czasu w IDA Pro, badając szczegółowo dokładną lokalizację awarii i wszystko, co program robi od tego momentu. Rzuć się w to szaleńczo z super długimi strunami. Sprawdź, co robi program, gdy nie umiera z powodu twojego exploita. Być może możesz znaleźć inny błąd, który będzie bardziej niezawodny. Często przydatne jest poznanie technik eksploatacji używanych na platformach innych niż te, które znasz. Techniki Windows mogą się przydać w systemie Unix i na odwrót. Nawet jeśli nie są przydatne, mogą dostarczyć potrzebnej inspiracji dla tego, czego potrzebuje twój ostatni exploit, aby odnieść sukces.

Przygotowanie

Zawsze bądź przygotowany. W rzeczywistości zawsze miej stos dysków twardych dostępnych z każdym systemem operacyjnym w każdym języku, z każdym dostępnym dodatkiem Service Pack i poprawką, i bądź przygotowany na odwoływanie się do adresów między nimi, aby określić, który zestaw adresów działa na wszystkich twoich celach. VMWare jest bardzo pomocne w tym przypadku, chociaż VMWare i OllyDbg czasami nie dogadują się, co może być kłopotliwe. Odwoływanie się do bazy danych wszystkich możliwych adresów może również skrócić czas korzystania z brutalnego wymuszania.

Brutalne zmuszanie

Czasami najlepszym sposobem na wzmocnienie swojego exploita jest wyczerpanie zakresu możliwych magicznych liczb. Jeśli masz ogromną listę potencjalnych adresów powrotu do ebx, być może powinieneś po prostu przejrzeć je wszystkie. W każdym razie brutalne wymuszanie jest często ostatecznością, ale jest to całkowicie słuszna ostateczność . Istnieje jednak kilka sztuczek, które mogą powstrzymać Cię przed marnowaniem czasu i pozostawieniem większej liczby dzienników, niż potrzebujesz. Określ, czy możesz sprawdzić więcej niż jeden adres naraz, gdy stosujesz brutalne wymuszanie. Przechowuj wszystkie prawidłowe wyniki w pamięci podręcznej, aby móc je najpierw sprawdzić. Maszyny w dowolnej sieci są zwykle konfigurowane w ten sam sposób, więc jeśli twoja technika zadziałała raz, prawdopodobnie zadziała ponownie. Wysyłanie absurdalnie dużych buforów kodu powłoki czasami może dać ci rozsądną szansę na prawidłowe trafienie magicznej liczby. A jeśli to możliwe, postaraj się, aby Twoje magiczne liczby były ze sobą powiązane. Jeśli wiesz, że jeden adres, którego będziesz potrzebować, zawsze będzie blisko drugiego, będzie ci znacznie lepiej niż wtedy, gdy będą one całkowicie niezależne od siebie. Wycieki pamięci często mogą znacznie ułatwić brutalne wymuszanie. Czasami nie potrzebujesz nawet prawdziwego wycieku pamięci, aby zapełnić pamięć shellcodeem. Na przykład w exploicie CANVAS IIS ColdFusion nawiązujemy 1000 połączeń ze zdalnym hostem, z których każde wysyła 20 000 bajtów kodu powłoki i NOP. Ta procedura szybko zapełnia pamięć kopiami shellcodu. Wreszcie, nie odłączając żadnego z naszych innych gniazd, wysyłamy przepełnienie sterty. Musi odgadnąć lokalizację naszego shellcodeu, ale prawie zawsze zgaduje poprawnie, ponieważ większość pamięci procesu jest nim wypełniona. Wypełnianie pamięci procesu jest łatwe, gdy proces jest wielowątkowy, podobnie jak IIS. Nawet jeśli proces nie jest wielowątkowy, wyciek pamięci może osiągnąć prawie to samo. A jeśli nie możesz znaleźć wycieku pamięci, możesz znaleźć zmienną statyczną, która przechowuje ostatni wynik zapytania i jest zawsze w tym samym miejscu. Jeśli spojrzysz na cały program, aby zobaczyć, czy zawiera jakieś operacje, którymi możesz manipulować, aby osiągnąć tego rodzaju cel, prawie zawsze znajdziesz coś przydatnego.

Lokalne exploity

Nie ma powodu, aby mieć niewiarygodny lokalny exploit. Kiedy mapujesz siebie do przestrzeni procesu, kontrolujesz prawie wszystko - przestrzeń pamięci, sygnalizację, zawartość dysku i lokalizację bieżącego katalogu. Wiele osób stwarza więcej problemów niż jest to konieczne z lokalnymi exploitami; to znak początkującego mieć lokalny exploit, który nie działa za każdym razem. Na przykład, podczas pisania prostego przepełnienia lokalnego bufora Linux/Unix, użyj exeve(), aby określić dokładne środowisko dla procesu docelowego. Teraz możesz dokładnie obliczyć, gdzie w pamięci będzie twój shellcode, i możesz napisać swój exploit jako atak typu powrót do libc bez żadnych zgadywanek. Osobiście lubimy wracać do strcpy() i kopiować nasz kod powłoki do sterty, a następnie go tam wykonywać. Możemy użyć dlopen() i dlsym(), aby znaleźć adres strcpy() podczas exploita. Ten rodzaj wyrafinowania sprawi, że twoje wyczyny będą działać na wolności. Jak zauważył Sinan Eren (znany szerzej jako noir), atakując jądro, możesz mapować pamięć do dowolnej potrzebnej lokalizacji, umożliwiając ustawienie adresu zwrotnego dokładnie w miejscu, w którym zaczyna się twój shellcode, nawet jeśli możesz użyć tylko jeden znak, do którego wrócić. (Innymi słowy, 0x00000000 może być całkowicie prawidłowym adresem zwrotnym podczas pisania ataku na lokalne jądro).

Odcisk palca systemu operacyjnego/aplikacji

Z wielu powodów odciski palców, które mogą dostarczyć narzędzia takie jak Nmap lub Xprobe, stanowią tylko część obrazu. Kiedy wykorzystujesz aplikację, musisz wiedzieć więcej niż tylko system operacyjny, na który celujesz. Musisz również wiedzieć, co następuje:

■■ Architektura (x86/SPARC/inne)

■■ Wersja aplikacji

■■ Konfiguracja aplikacji

■■ Konfiguracja systemu operacyjnego (stos non-exec/PaX/DEP itd.)

W wielu innych przypadkach identyfikacja systemu operacyjnego jest całkowicie bezużyteczna, ponieważ jesteś przesyłany z jednego hosta do drugiego. A może po prostu nie chcesz wysyłać dziwacznych pakietów identyfikacyjnych systemu operacyjnego do hosta, ponieważ odciskałoby się to na każdym nasłuchującym IDS sieci. Dlatego, aby pisać niezawodne exploity, często musisz znaleźć unikalne sposoby na odciski palców zdalnego hosta, które znajdują się w granicach całkowicie normalnego ruchu. Zawsze najlepiej jest móc wykonać odcisk palca na tym samym porcie, na który w końcu będziesz atakować. Poniższy przykład jest używany w exploicie MSRPC firmy CANVAS. Możesz zobaczyć, że po prostu używając portu 135 (usługa docelowa), możemy precyzyjnie zawęzić system operacyjny, na który kierujemy. Najpierw dzielimy XP i Windows 2003 od NT 4.0 i Windows 2003. Następnie dzielimy 2003 od XP (używając innej funkcji, której tutaj nie pokazano). Następnie oddzieliliśmy Windows 2000 od NT 4.0. Cała ta funkcja wykorzystuje publicznie dostępne interfejsy na porcie 135 (TCP), co jest dobre, ponieważ może to być jedyny otwarty port. Korzystając z tej techniki, nasz exploit może zawęzić celowanie do właściwej platformy za pomocą zaledwie kilku prostych połączeń.

```
def runTest(self):

UUID2K3=”1d55b526-c137-46c5-ab79-638f2a68e869”

callid=1

error,s=msrpcbind(UUID2K3,1,0,self.host,self.port,callid)
```

```
if error==0:
errstr="Could not bind to the msrpc service for 2K3,XP - assuming
NT 4 or Win2K"
self.log(errstr)
else:
if self.testFor2003(): #Simple test not shown here.
self.setVersion(15)
self.log("Test indicated connection succeeded to msrpc service.")
self.log("Attacking using version %d:
%s"%(self.version,self.versions[self.version][0]))
return 1
self.setVersion(1) #default to Win2K or XP
UUID2K="000001a0-0000-0000-c000-000000000046"
#only provided by 2K and above
callid=1
error,s=msrpcbind(UUID2K,0,0,self.host,self.port,callid)
if error==0:
errstr="Could not bind to the msrpc service for 2K and above -
assuming NT 4"
self.log(errstr)
self.setVersion(14) #NT4
else:
self.log("Test indicated connection succeeded to msrpc service.")
self.log("Attacking using version %d:
%s"%(self.version,self.versions[self.version][0]))
return 1 #Windows 2000 or XP
callid=0
#IRemoteDispatch UUID
UUID="4d9f4ab8-7d1c-11cf-861e-0020af6e7c57"
error,s=msrpcbind(UUID,0,0,self.host,self.port,callid)
#error is reversed, sorry.
```

```
if error==0:

errstr="Could not bind to the msrpc service necessary to run the

attack"

self.log(errstr)

return 0

#we assume it's vulnerable if we can bind to it

self.log("Test indicated connection succeeded to msrpc service.")

self.log("Attacking using version %d:

%s"%(self.version,self.versions[self.version][0]))

return 1
```

Wycieki informacji

Przeszliśmy już epokę, w której każdy exploit był po prostu rakietą typu „wystrzel i zapomnij". W dzisiejszych czasach dobry autor exploitów szuka sposobów na skierowanie ataku bezpośrednio na cel. Istnieją metody uzyskiwania informacji, często określonych adresów pamięci, od celów. Oto niektóre z nich:

■■ Odczytywanie i interpretowanie danych wysyłanych przez cel. Na przykład pakiety MSRPC często zawierają wskaźniki, które są kierowane bezpośrednio z pamięci. Te wskaźniki mogą służyć do przewidywania miejsca w pamięci twojego procesu docelowego.

■■ Użycie funkcji przepełnienia sterty do zapisania danych przed ich wysłaniem z powrotem pozwala określić, gdzie w pamięci znajduje się bufor.

■■ Użycie funkcji przepełnienia sterty w stylu frontlink() w celu zapisania adresu zmiennych wewnętrznych malloc w danych przed ich wysłaniem z powrotem pozwala określić, gdzie w pamięci znajdują się wskaźniki funkcji malloc za pomocą prostego obliczenia.

■■ Nadpisanie pola długości często umożliwia wysłanie dużej części pamięci serwera (pomyśl o przepełnieniu BIND TSIG).

■■ Wykorzystanie niedomiaru lub innego podobnego ataku może pozwolić na przesłanie części pamięci serwera. FX firmy Phenoelit z powodzeniem wykorzystuje tę metodę z pakietami echa dla swojego exploita Cisco HTTPD. Jego praca jest znakomitym przykładem połączenia dwóch exploitów w jeden bardzo niezawodny exploit.

Analiza informacji o czasie może być cennym sposobem na uzyskanie wglądu w rodzaje błędów, na które napotyka Twój exploit. Czy wysłał ci pakiet resetowania od razu, czy przekroczył limit czasu, a następnie wysłał ci reset? Halvar Flake powiedział kiedyś: „Żaden dobry haker nie szuka tylko jednego błędu". Wyciek informacji może sprawić, że nawet trudny błąd będzie możliwy. Nawet PaX (zaawansowana łatka chroniąca pamięć opartą na jądrze) można łatwo pokonać dzięki wystarczająco dobremu wyciekowi informacji.

Podsumowanie

Załóżmy, że piszesz exploit dla niestandardowego serwera sieci Web Win32. Po dniu pracy exploit, proste przepełnienie stosu, działa idealnie pięć razy na sześć. Wykorzystuje standardową technikę „nadpisywania struktury obsługi wyjątków", która wskazuje na przestrzeń pamięci procesów. To z kolei wskazuje na powrót pop w segmencie .text. Jednak ponieważ docelowy proces jest wielowątkowy, czasami inny wątek nadpisuje shellcode i atak się nie udaje. Więc przepisujesz exploit, używając znacznie mniejszego ciągu, który pozwala oryginalnej funkcji na bezpieczny powrót i ostatecznie uzyskanie kontroli za pomocą zapisanego wskaźnika powrotu w ramce stosu kilka zwrotów dalej. Ta technika, chociaż ogranicza rozmiar shellcodu, którego możesz użyć, jest znacznie bardziej niezawodna. Chodzi o to, że czasami nie można polegać nawet na bardzo stabilnych technikach – czasami trzeba przetestować kilka różnych metod wykorzystania błędu, a następnie wypróbować każdą metodę na dowolnej liczbie platform testowych, aż znajdziesz najlepsze rozwiązanie. Kiedy utkniesz, spróbuj wydłużyć lub maksymalnie skrócić ciąg ataku lub wstrzyknąć znaki, które mogą spowodować, że wydarzy się coś innego. Jeśli masz kod źródłowy, spróbuj dokładnie śledzić swoje dane, gdy przepływają przez program. Ogólnie rzecz biorąc, nie poddawaj się. Musisz mieć dużą dozę pewności siebie, aby pozostać w tej grze, ponieważ dopóki twój exploit nie zadziała, nigdy nie będziesz wiedział, czy odniesiesz sukces. Zapewniamy, że Twoja wytrwałość jest tego warta. Ale musisz pogodzić się z faktem, że czasami nigdy nie dowiesz się, dlaczego twój exploit nie działa na wolności.