Instrumentalne dochodzenie: podejście ręczne

Przy całej tej rozmowie o fuzzingu możesz uwierzyć, że w świecie współczesnego łowcy owadów nie ma miejsca na ręczne dochodzenie. Celem tego rozdziału jest pokazanie, dlaczego tak nie jest i że ręczne polowanie na błędy ma się dobrze. Zaczniemy od omówienia techniki (takiej jaka jest), a następnie przejdziemy przez kilka przykładów procesów myślowych i technik stojących za odkryciem pewnych błędów. Po drodze zajmiemy się również ogólnie walidacją danych wejściowych i porozmawiamy o ciekawych sposobach jej ominięcia, ponieważ walidacja danych wejściowych często utrudnia proces badawczy, a nieco głębsze zrozumienie może pomóc zarówno wzmocnić ataki, jak i zwiększyć zrozumienie defensywej techniki

Filozofia

Ideą stojącą za naszym podejściem jest uproszczenie spojrzenia badacza na system, co pozwala mu skupić się na strukturze i zachowaniu systemu z punktu widzenia bezpieczeństwa technicznego, zamiast kierować się jakąś predefiniowaną ścieżką przez dokumentację dostawcy lub kod źródłowy. Jest to bardziej postawa i podejście niż konkretna technika, chociaż będziesz potrzebować pewnych podstawowych umiejętności. Z doświadczenia wynika, że takie podejście prowadzi do odkrycia błędów „niemożliwych do pomyślenia" przez zespoły programistów - ponieważ były zbyt oczywiste lub zasłonięte przez kod źródłowy (na przykład złożone definicje makr w języku C) lub ponieważ po prostu nie pomyślano o interakcji między komponentami systemu. Można powiedzieć, że wyrzucenie podręcznika jest rzeczą wyzwalającą. Podejście można najlepiej podsumować w następujący sposób:

* Spróbuj zrozumieć system bez odwoływania się do jego dokumentacji i kodu źródłowego.

* Zbadaj prawdopodobne obszary słabości. Podczas dochodzenia skorzystaj z narzędzi do śledzenia systemu, aby dowiedzieć się więcej o zachowaniu systemu i zwróć uwagę, gdzie zachowania się różnią - te punkty mogą nie być oczywiste.

* Tam, gdzie obserwuje się różne zachowania, spróbuj wspólnych form ataku i obserwuj reakcję.

* Kontynuuj, aż omówisz wszystkie zachowania.

Być może konkretny przykład wyjaśniłby ten proces.

Przepełnienie exproc Oracle

Oracle wydał Security Alert 57 dotyczący przepełnienia extproc. We wrześniu 2002 r. Next Generation Security Software przeprowadziło rygorystyczne badanie Oracle RDBMS w poszukiwaniu luk w zabezpieczeniach, ponieważ autorzy uważali, że Oracle DBMS został niedostatecznie zbadany przez resztę społeczności zajmującej się bezpieczeństwem. David Litchfield znalazł wcześniej błąd w mechanizmie extproc, więc postanowiliśmy przyjrzeć się temu ogólnemu obszarowi. Ważne jest, aby zrozumieć szczegóły architektoniczne tutaj, które odnoszą się do tego, jak odkryto ten pierwszy błąd. Ogólnie rzecz biorąc, zaawansowane DBMS obsługują niektóre dialekty Structured Query Language (SQL), które pozwalają na tworzenie bardziej złożonych skryptów, a nawet procedur. W SQL Server nazywa się to Transact-SQL i umożliwia takie rzeczy, jak pętle WHILE, instrukcje IF i tak dalej. SQL Server umożliwia również bezpośrednią interakcję z systemem operacyjnym za pośrednictwem tego, co Microsoft nazywa „rozszerzonymi procedurami składowanymi" - są to niestandardowe funkcje zaimplementowane w bibliotekach DLL, które można napisać w C/C++. W przeszłości w rozszerzonych procedurach składowanych SQL Server występowało wiele przepełnień bufora, więc logiczne jest przypuszczenie, że analogiczne mechanizmy w innych DBMS będą cierpieć z powodu podobnych problemów. Wprowadź Oracle. Oracle oferuje mechanizm, który jest znacznie bogatszy niż mechanizm

rozszerzonych procedur składowanych SQL Server, ponieważ umożliwia wywoływanie dowolnych funkcji bibliotecznych, a nie tylko funkcji zgodnych z jakąś predefiniowaną specyfikacją. W Oracle wywołania bibliotek zewnętrznych nazywane są procedurami zewnętrznymi i są wykonywane w procesie wtórnym, extproc. extproc jest oferowany jako dodatkowa usługa Oracle i może być podłączony w podobny sposób do samej usługi bazy danych. Kolejną ważną rzeczą do zrozumienia jest protokół Transparent Network Substrate (TNS). Jest to część architektury, która zarządza komunikacją procesu Oracle z klientami oraz z innymi częściami systemu. TNS to protokół tekstowy z nagłówkiem binarnym. Obsługuje wiele różnych poleceń, ale głównym celem jest uruchamianie, zatrzymywanie i inne zarządzanie usługami Oracle. Przyjrzeliśmy się temu mechanizmowi extproc i postanowiliśmy go oprzyrządować, aby zobaczyć, co robi. Używaliśmy Oracle na platformie Windows, więc wzięliśmy wszystkie standardowe wywołania gniazd w procesie Oracle i określiliśmy je na przerwanie - connect, accept, recv, recvfrom, readfile, writefile i tak dalej. Następnie wykonaliśmy szereg wezwań do procedur zewnętrznych. David Litchfield odkrył, że gdy Oracle wywołał rozszerzoną procedurę składowaną, użył serii wywołań TNS, a następnie prostego protokołu do wykonania wywołania. Nie było absolutnie żadnego uwierzytelnienia; extproc po prostu założył, że na drugim końcu połączenia musi być Oracle. Konsekwencją tego jest to, że jeśli możesz (jako zdalny napastnik) skierować extproc do wywołania wybranej przez ciebie biblioteki, możesz łatwo naruszyć serwer, np. uruchamiając funkcję systemową w libc lub msvcrt.dll w systemie Windows. Istnieje wiele czynników łagodzących, ale w przypadku instalacji domyślnej (przed naprawieniem tego błędu przez Oracle) tak właśnie było. Zgłosiliśmy to firmie Oracle i pracowaliśmy z nimi, aby opublikować łatkę. Alert Oracle (numer 29) można znaleźć pod adresem otn.oracle.com/deploy/security/pdf/lsextproc_alert.pdf. Porady Davida Litchfielda są dostępne pod adresem www.ngssoftware.com/advisories/oraplsextproc.txt. Ponieważ ten obszar zachowania Oracle jest tak wrażliwy (pod względem bezpieczeństwa systemu), postanowiliśmy ponownie przejrzeć wszystkie zachowania związane z procedurami zewnętrznymi, aby zobaczyć, czy możemy znaleźć coś więcej. Sposób wykonania powyższego wywołania jest dość prosty - polecenia TNS można zobaczyć, debugując Oracle i uruchamiając następujący skrypt (z doskonałego artykułu Litchfield „HackProofing Oracle Application Server”, który można znaleźć na stronie www.ngssoftware.com/papers/ hpoas.pdf):

```
Rem

Rem oracmd.sql

Rem

Rem Uruchamiaj polecenia systemowe za pośrednictwem serwerów baz danych Oracle

Rem

Rem Bugs do david@ngssoftware.com

Rem

CREATE OR REPLACE LIBRARY exec_shell AS

‘C:\winnt\system32\msvcrt.dll’;

/

show errors

CREATE OR REPLACE PACKAGE oracmd IS

PROCEDURE exec (cmdstring IN CHAR);
```

end oracmd;

/

show errors

CREATE OR REPLACE PACKAGE BODY oracmd IS

PROCEDURE exec(cmdstring IN CHAR)

IS EXTERNAL

NAME “system”

LIBRARY exec_shell

LANGUAGE C; end oracmd;

/

show errors

Następnie uruchamiasz procedurę

exec oracmd.exec (‘dir > c:\oracle.txt);

aby rozpocząć właściwe wykonywanie. Zaczynając od początku, wypróbowaliśmy zwykłe rzeczy w instrukcji create lub replace przez ręczne podłączenie zapytań i zobaczenie, co się stało (w debuggerze i FileMon). Co zaskakujące, gdy podaliśmy zbyt długą nazwę biblioteki:

CREATE OR REPLACE LIBRARY ext_lib IS ‘AAAAAAAAAAAAAAAAAAAAAAAA...’;

a następnie wywołał w nim funkcję:

CREATE or replace FUNCTION get_valzz

RETURN varchar AS LANGUAGE C

NAME “c_get_val”

LIBRARY ext_lib;

select get_valzz from dual;

stało się coś dziwnego, nie z samym Oracle, ale najwyraźniej gdzieś indziej – połączenie było resetowane – co normalnie wskazywałoby na jakiś wyjątek. Dziwne było to, że nie wystąpiło to w procesie Oracle. Po chwili przyjrzenia się FileMon, zdecydowaliśmy się debugować proces TNS Listener (tnslsnr) (który obsługuje protokół TNS i jest pośrednikiem między Oracle a extproc podczas wywoływania procedur zewnętrznych). Ponieważ proces tnslsnr uruchamia extproc, użyliśmy WinDbg, który pozwala na łatwe śledzenie procesów podrzędnych. Proces był trochę skomplikowany:

1) Zatrzymaj wszystkie usługi Oracle

2) Uruchom usługę bazy danych Oracle („OracleService<nazwa hosta>”)

3) Z sesji wiersza poleceń na interaktywnym pulpicie uruchom windbg -o tnslsnr.exe

Powoduje to, że WinDbg debuguje odbiornik TNS i wszelkie procesy, które uruchamia odbiornik TNS. Nasłuchiwanie TNS działa na interaktywnym pulpicie. Rzeczywiście, kiedy to zrobiliśmy, zobaczyliśmy

magiczny wyjątek w WinDbg: wyjątki pierwszej szansy są zgłaszane przed jakąkolwiek obsługą wyjątków. Ten wyjątek może być oczekiwany i obsłużony.

eax=00000001 ebx=00ec0480 ecx=00010101 edx=ffffffff esi=00ebbfec

edi=00ec04f8

eip=41414141 esp=0012ea74 ebp=41414141 iopl=0 nv up ei pl zr na

po nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000

efl=00010246

41414141 ?? ???

Wskazywało to na przepełnienie stosu waniliowego w extproc.exe. Szybki zestaw testów ujawnił, że problem dotyczył nie tylko platformy Windows. Mamy jeden wektor wyzwalający błąd - poprzez instrukcję create library. Ale nawiązując do oryginalnego zalecenia Davida dotyczącego extproc, przypomnieliśmy sobie, że możliwe jest wykonywanie połączeń do extproc bezpośrednio jako zdalny atakujący. Następnie zakodowaliśmy szybką wiązkę do zdalnego wyzwalania przepełnienia i odkryliśmy, że działa to w ten sam sposób. Znaleźliśmy zdalne nieuwierzytelnione przepełnienie stosu w Oracle. Tyle o „Niezniszczalnym”! Najwyraźniej luka ta została wprowadzona w łatce do poprzedniego błędu - wprowadzona funkcjonalność rejestrowania żądania uruchomienia zewnętrznej procedury jest podatna na przepełnienie. Podsumowując proces związany z odkryciem tych dwóch błędów:

* Zdawaliśmy sobie sprawę z prawdopodobnej słabości architektury, ponieważ wiedzieliśmy, że SQL Server ma problemy w tym ogólnym obszarze funkcjonalności. Dlatego uznaliśmy za prawdopodobne, że Oracle będzie miał podobny problem, ponieważ dostęp do procedur przechowywanych jest trudny do bezpiecznego wykonania.

* Dokładne oprzyrządowanie i śledzenie zachowania Oracle doprowadziło nas do możliwości wykonywania zewnętrznych procedur bez uwierzytelniania - błąd numer jeden.

* Wracając do tego obszaru funkcjonalności, odkryliśmy, że coś dziwnego stało się ze zbyt długimi nazwami bibliotek (od czasu łatki do pierwszego błędu extproc).

* Użyliśmy debuggerów i narzędzia do monitorowania plików (doskonały FileMon Russinovicha i Cogswella) i zidentyfikowaliśmy kwestionowane komponenty.

* Podczas debugowania omawianych komponentów zauważyliśmy wyjątek wskazujący na przepełnienie stosu — błąd numer dwa.

W żadnym momencie niczego nie zautomatyzowaliśmy; wszystko opierało się na uważnym przyjrzeniu się konstrukcji testowanego systemu i zlekceważeniu dokumentacji, woląc zamiast tego zrozumieć system pod kątem jego oprzyrządowanego zachowania. Jako przypis do tego ćwiczenia firma Oracle zamieściła teraz doskonały zestaw informacji o obejściu tych błędów w Alert 57, a także poprawkę, która w pełni rozwiązuje oba problemy.

Powszechne awarie architektoniczne

Jak widzieliśmy w poprzednim przykładzie, rzeczy zawodzą w podobny sposób. Po codziennym przeglądaniu porad przez kilka lat zaczynasz dostrzegać wzorce, a następnie podążasz za nimi we

własnych badaniach. Prawdopodobnie warto zatrzymać się na chwilę i zastanowić nad tymi wzorcami, ponieważ mogą one dostarczyć pomysłów na przyszłe badania.

Problemy zdarzają się na granicach

Chociaż nie jest to powszechnie prawdą, zazwyczaj problemy z bezpieczeństwem pojawiają się, gdy następuje pewnego rodzaju przejście: z jednego procesu do drugiego, z jednej technologii do drugiej lub z jednego interfejsu do drugiego. Oto kilka przykładów.

Proces wywołujący proces zewnętrzny na tym samym hoście

Dobrymi przykładami tego problemu są opisany wcześniej problem z Oracle 57 oraz problem z przejęciem kontroli nad nazwanym potokiem odnaleziony przez Andreasa Junestama i opisany w biuletynie Microsoft MS03-031. Aby zobaczyć kilka interesujących problemów z podnoszeniem uprawnień, użyj HandleEx lub Process Explorer (od Sysinternals), aby przyjrzeć się uprawnieniom przypisanym do obiektów globalnych (takich jak sekcje pamięci współdzielonej) w systemie Windows. Wiele aplikacji nie chroni przed atakami lokalnymi. W Uniksie znajdziesz całą rodzinę problemów związanych z analizowaniem opcji wiersza poleceń i zmiennych środowiskowych, gdy proces woła do innego procesu, aby wykonał jakąś funkcję. Po raz kolejny oprzyrządowanie jest pomocne, jeśli szukasz w tym obszarze. W tym przypadku ltrace/strace/truss są prawdopodobnie najlepszym sposobem.

Proces odwołujący się do zewnętrznej, dynamicznie ładowanej biblioteki

Ponownie, Oracle i SQL Server dostarczają wielu przykładów tego problemu — jednym z nich jest oryginalny błąd extproc wykryty przez Davida Litchfielda (Alert 29 Oracle), a drugim jest wiele rozszerzonych przepełnień procedur przechowywanych w SQL Server. Ponadto istnieje wiele problemów z filtrami ISAPI w Microsoft IIS, w tym ze składnikiem Commerce Server, filtrem ISM.DLL, filtrem SQLXML, filtrem .printer ISAPI i wieloma innymi. Jednym z powodów, dla których pojawiają się tego rodzaju problemy, jest to, że chociaż ludzie intensywnie kontrolują podstawowe zachowania demona sieciowego, mają tendencję do pomijania obsługi rozszerzalności. IIS nie jest w tym odosobniony. Spójrz na błąd Apache mod_ssl off-by-one, a także problemy w mod_mylo, mod_cookies, mod_frontpage, mod_ntlm, mod_auth_any, mod_access_referer, mod_jk, mod_php i mod_dav. Jeśli przeprowadzasz audyt nieznanego systemu, zwykle można znaleźć słaby punkt w tego rodzaju obszarze funkcjonalnym.

Proces wywołujący funkcję na zdalnym hoście

To również jest pole minowe, chociaż ludzie są bardziej świadomi zagrożeń. Niedawny błąd Microsoft Windows RASMAN RPC (MS06-025) pokazuje, że tego rodzaju problem wciąż istnieje. Większość błędów RPC pasuje do tej kategorii, jak Sun UDP PRC DOS, przepełnienie usługi lokalizatora, liczne przepełnienia MS Exchange znalezione przez Dave'a Aitela i stary ulubiony błąd ciągu formatu statd odnaleziony przez Daniela Jacobowitza.

Problemy pojawiają się podczas tłumaczenia danych

Kiedy dane są przekształcane z jednej formy w drugą, często jest możliwość ominięcia kontroli. Jest to właściwie podstawowy problem związany z tłumaczeniem między gramatykami. Powodem, dla którego ten rodzaj problemu (często nazywany błędem kanonizacji) jest tak powszechny, jest to, że wyjątkowo trudno jest stworzyć system, w którym programowalne interfejsy stają się mniej złożone gramatycznie w miarę zagłębiania się w drzewo wywołań. Formalnie moglibyśmy to ująć tak: Funkcja f() implementuje zestaw zachowań F. f() implementuje te zachowania wywołując funkcję g(), przekazując jej część danych wejściowych do f(). g implementuje zestaw zachowań G. Niestety, zestaw

G zawiera zachowania, których nie jest pożądane ujawnianie za pomocą f(). Te złe zachowania nazywamy Gbad. Dlatego f() musi zaimplementować jakiś mechanizm, aby zapewnić, że F nie zawiera żadnego elementu Gbad. Jedynym sposobem, w jaki realizator f() może to zrobić, jest pełne zrozumienie całego G i walidacja danych wejściowych do f(), aby upewnić się, że żadna kombinacja danych wejściowych nie skutkuje żadnym elementem Gbad.

Jest to problem z dwóch powodów:

* Rzeczy prawie zawsze stają się bardziej złożone, im niżej schodzisz w dół drzewa wywołań, więc f() zajmuje się zbyt wieloma przypadkami.

* g() ma ten sam problem, co h(), i(), j() itd. drzewo połączeń.

Weźmy na przykład funkcje systemu plików Win32. Być może masz program, który akceptuje nazwę pliku. O ile rozumie pojęcie nazw plików, zakłada, co następuje:

* Nazwa pliku może mieć na końcu rozszerzenie. Rozszerzenia mają zwykle, ale nie zawsze, trzy znaki długości i są oznaczone ostatnim znakiem kropki (.) w nazwie pliku.

* Nazwa pliku może być pełną ścieżką. Jeśli tak, zaczyna się od litery dysku, po której następuje znak dwukropka (:).

* Nazwa pliku może być ścieżką względną. Jeśli tak, będzie zawierać znaki ukośnika odwrotnego (\).

* Każdy znak odwrotnego ukośnika sygnalizuje przejście do katalogu podrzędnego.

Można to traktować jako gramatykę nazw plików, jeśli chodzi o program. Niestety, gramatyka zaimplementowana przez podstawowe funkcje systemu plików (takie jak Win32 API CreateFile) zawiera wiele innych potencjalnie niebezpiecznych konstrukcji, takich jak poniższe (nie jest to wyczerpująca lista)

* Nazwa pliku może zaczynać się od sekwencji podwójnego ukośnika odwrotnego. W takim przypadku pierwsza nazwa katalogu oznacza hosta w sieci, a druga nazwę udziału SMB. Interfejs API systemu plików podejmie próbę nawiązania połączenia z tym udziałem przy użyciu (możliwych do przeszukania) poświadczeń bieżącego użytkownika.

* Nazwa pliku może również zaczynać się od sekwencji \\?\, co oznacza, że jest to ścieżka pliku Unicode i może przekroczyć normalne limity długości narzucone przez interfejs API systemu plików.

* Nazwa pliku może zaczynać się od \\?\UNC, co spowoduje również wyzwolenie opisane wcześniej zachowanie połączenia Microsoft Share.

* Nazwa pliku może zaczynać się od \\.\PHYSICALDRIVE< n >, gdzie < n > jest indeksem liczonym od zera dysku fizycznego, który ma zostać otwarty. To otworzy dysk fizyczny do surowego dostępu.

* Nazwa pliku może zaczynać się od \\.\pipe\<nazwa potoku>. Nazwany potok < nazwa potoku > zostanie otwarty.

* Nazwa pliku może zawierać znak dwukropka (:) (po początkowej sekwencji liter dysku). Oznacza to alternatywny strumień danych w systemie plików NTFS, który jest faktycznie traktowany jako odrębny plik, ale który nie jest wyświetlany oddzielnie na listach katalogów. Strumień pliku :$DATA jest zarezerwowany dla normalnej zawartości pliku.

* Nazwa pliku może zawierać (jako nazwę katalogu) .. lub . sekwencja. Pierwszy przypadek sygnalizuje przejście do katalogu nadrzędnego, drugi oznacza, że nie należy wykonywać przejścia do katalogu.

Możliwych jest wiele innych, równie dziwacznych zachowań. Chodzi o to, że jeśli nie będziesz ostrożny w walidacji danych wejściowych, wprowadzisz problemy, ponieważ bazowy interfejs API prawdopodobnie zaimplementuje zachowania, których nie jesteś świadomy. Dlatego z punktu widzenia napastnika sensowne jest zrozumienie tych podstawowych zachowań i próba dotarcia do nich poprzez obronne mechanizmy walidacji danych wejściowych. Niektóre z rzeczywistych błędów, które występują z powodu tego rodzaju problemu (niestety nie wszystkie szelkody) obejmują błąd Unicode IIS, błąd podwójnego dekodowania IIS, problem wstrzykiwania SMTP CDONTS.NewMail, zachowanie http:// nazwy pliku w PHP (możesz może otworzyć plik na podstawie adresu URL) oraz lukę w zabezpieczeniach kodu źródłowego Macromedia Apache (jeśli dodasz zakodowaną spację na końcu adresu URL, otrzymasz kod źródłowy). Jest ich znacznie więcej. Prawie każdy błąd ujawniający kod źródłowy pasuje do tej kategorii. Jeśli się nad tym zastanowić, walidacja danych wejściowych jest w rzeczywistości powodem, dla którego przepełnienia są tak szkodliwe. Dane wejściowe funkcji są interpretowane w pewnym podstawowym kontekście. W przypadku przepełnienia stosu dane, które przepełniają bufor, są traktowane jako część ramki stosu zawierająca dane, wskaźniki wirtualne (VPTR), zapisane adresy powrotne, adresy obsługi wyjątków i tak dalej. To, co możesz nazwać frazą w jednej gramatyce, jest interpretowane jako fraza w innej. Prawie wszystkie ataki można podsumować jako próby skonstruowania fraz, które są poprawne w wielu gramatykach. Jest to związane z pewnymi interesującymi implikacjami obronnymi w dziedzinie teorii informacji i teorii kodowania, ponieważ jeśli możesz upewnić się, że dwie gramatyki nie mają wspólnych zwrotów, może (ewentualnie) być w stanie zapewnić, że żaden atak nie będzie możliwy w oparciu o translację między nimi. Pomysł kontekstów interpretacyjnych jest przydatny, zwłaszcza jeśli masz do czynienia z celem, który obsługuje różne protokoły sieciowe – na przykład serwer sieci Web, który wysyła wiadomości e-mail lub przesyła dane do serwera usług sieci Web przy użyciu dziwnego formatu XML. Pomyśl tylko „Co analizuje dane wejściowe?” a jeśli potrafisz poprawnie odpowiedzieć na to pytanie, możesz być na dobrej drodze do znalezienia błędu.

Klaster problemów w obszarach asymetrii

Ogólnie rzecz biorąc, programiści mają tendencję do stosowania technik defensywnych w całym obszarze zachowań, używając takich rzeczy, jak limity długości, sprawdzanie ciągów formatu lub inne rodzaje walidacji danych wejściowych. Doskonałym sposobem na znalezienie problemów jest znalezienie obszaru asymetrii i zbadanie go, aby dowiedzieć się, co go wyróżnia. Być może pojedynczy nagłówek HTTP obsługiwany przez serwer sieci Web wydaje się mieć inny limit długości niż wszystkie inne, a może zauważysz dziwną odpowiedź, gdy dołączysz określony symbol do danych wejściowych. Lub ewentualnie określenie ostatnio zaimplementowanej metody sieci Web w Apache wydaje się zmieniać komunikaty o błędach. Być może twoja próba wykonania pliku przez wadliwy serwer WWW nie powiedzie się, gdy zażądasz cmd.exe, ale powiedzie się na ftp.exe. Zwrócenie uwagi na różne obszary może wskazać obszary produktu, które są gorzej chronione.

Problemy pojawiają się, gdy uwierzytelnianie i autoryzacja są pomylone

Uwierzytelnianie to weryfikacja tożsamości, nic więcej. Autoryzacja to proces określania, czy dana tożsamość powinna mieć dostęp do danego zasobu. Wiele systemów bardzo dba o to pierwsze i zakłada, że następuje to drugie. Co gorsza, w niektórych przypadkach pozornie nie ma żadnego związku między tymi dwoma - jeśli możesz znaleźć alternatywną drogę do danych, możesz uzyskać do nich dostęp. Prowadzi to do kilku interesujących sytuacji podniesienia uprawnień, takich jak przykład Oracle extproc. Można go również zobaczyć w Lotus Domino z błędem obejściu ACL widoku oraz w Oracle mod_plsql z obejściem uwierzytelniania. Innym dobrym przykładem tego, co dzieje się, gdy inna trasa jest dostarczana do poufnych danych, była luka htaccess w Apache, która nie uwzględnia wielkości liter. Ten rodzaj problemów można również zaobserwować w wielu aplikacjach internetowych.

Ponieważ HTTP jest z natury bezstanowy, mechanizm używany do utrzymywania stanu (identyfikator sesji) zwykle niesie ze sobą stan uwierzytelniania. Jeśli możesz w jakiś sposób odgadnąć lub odtworzyć identyfikator sesji, możesz pominąć etap uwierzytelniania.

Problemy pojawiają się w najgłupszych miejscach

Jeśli konkretne polowanie na błąd staje się zbyt techniczne i to był długi dzień, nie bój się wypróbować naprawdę oczywistego. Zbyt długie nazwy użytkowników były przyczyną wielu błędów:

* www.ngssoftware.com/advisories/sambar.txt

* otn.oracle.com/deploy/security/pdf/2003Alert58.pdf

* www.ngssoftware.com/advisories/ora-unauthrm.txt

* www.ngssoftware.com/advisories/ora-isqlplus.txt

* www.ngssoftware.com/advisories/steel-arrow-bo.txt

* cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0891

* www.kb.cert.org/vuls/id/322540

Ogólnie rzecz biorąc, faza uwierzytelniania protokołu jest dobrym celem badania przepełnień i ciągów formatowania z oczywistego powodu, ponieważ jeśli możesz uzyskać kontrolę przed uwierzytelnieniem, nie potrzebujesz nazwy użytkownika ani hasła, aby złamać zabezpieczenia serwera. Kolejną parą klasycznych, nieuwierzytelnionych zdalnych błędów głównych jest błąd hello znaleziony przez Dave'a Aitela (cve.mitre.org/cgi-bin/cvename.cgi? name=CAN-2002-1123) oraz błędy SQL-UDP znalezione przez Davida Litchfielda (www.ngssoftware.com/advisories/mssql-udp.txt). Znaleźliśmy również produkty, w których niejasne części różnych protokołów były dostępne bez uwierzytelnienia – w niektórych przypadkach po prostu pominięto sprawdzenie uwierzytelnienia. W jednym spektakularnym przypadku uznano, że brak stanu uwierzytelnienia czyni użytkownika superużytkownikiem („uid to 0, jeśli go nie ma", w połączeniu z „uid 0 oznacza, że jesteś rootem"). Te obszary są świetnymi celami z oczywistych powodów.

Omijanie walidacji danych wejściowych i wykrywania ataków

Zrozumienie walidacji danych wejściowych i wiedza, jak ją ominąć, to podstawowe umiejętności łowcy błędów. Przedstawimy krótki przegląd tematu, aby pomóc Ci zrozumieć, gdzie popełniane są błędy, i przedstawić przydatne techniki omijania walidacji.

Usuwanie złych danych

Ludzie często używają wadliwych wyrażeń regularnych, aby ograniczyć (lub wykryć) potencjalne ataki. Jednym z powszechnych zastosowań jest usuwanie danych wejściowych, o których wiadomo, że są złe - jeśli bronisz się przed wstrzyknięciem SQL, możesz na przykład napisać filtr, który będzie usuwał zastrzeżone słowa SQL, takie jak select, union, gdzie, z i tak dalej. Na przykład ciąg wejściowy

‘ nazwa union select, hasło z sys.user$--

może stać się

‘ nazwa, hasło sys.user$--

Powoduje to błąd. Czasami możesz ominąć ten błąd, rekursywnie włączając w siebie złe dane, na przykład:

‘ uniunionon selselect nazwa, hasło z sys.user$--

Każdy zły termin zawiera się sam w sobie. Gdy wartości są usuwane, dołączone złe dane są odtwarzane, pozostawiając nam dokładnie te dane, których szukaliśmy. Oczywiście działa to tylko wtedy, gdy znane złe terminy składają się z co najmniej dwóch różnych znaków.

Korzystanie z kodowania alternatywnego

Najbardziej oczywistym sposobem na obejście sprawdzania poprawności danych wejściowych jest użycie alternatywnego kodowania danych. Na przykład może się okazać, że sposób, w jaki zachowuje się serwer sieci Web lub środowisko aplikacji sieci Web, zależy od sposobu kodowania danych formularza. Dobrym przykładem jest specyfikator kodowania IIS Unicode %u. W IIS te dwa są równoważne:

* www.example.com/%c0%af

* www.example.com/%uc0af

Innym dobrym przykładem jest traktowanie białych znaków. Może się okazać, że aplikacja traktuje znaki spacji jako ograniczniki, ale nie traktuje znaków tabulacji, powrotu karetki ani znaku końca wiersza. W przypadku przepełnienia Oracle TZ_OFFSET spacja zakończy specyfikator strefy czasowej, ale znak TAB nie. Napisaliśmy exploita na ten błąd, który uruchamiał polecenie i mieliśmy problem z określeniem parametrów w exploicie. Szybko zmodyfikowaliśmy exploita, aby zmienić wszystkie spacje na tabulatory, co działało dobrze, ponieważ większość powłok traktuje zarówno spacje, jak i tabulatory jako ograniczniki. Innym klasycznym przykładem był filtr ISAPI, który próbował ograniczyć dostęp do katalogu wirtualnego IIS na podstawie określonych poświadczeń. Filtr uruchomi się, jeśli zażądasz czegokolwiek w katalogu /downloads (www.example.com/downloads/hot_new_file.zip). Oczywiście pierwszą rzeczą, którą należy spróbować, aby to ominąć, jest:

www.example.com/Pobieranie/hot_new_file.zip

co nie działa. Następnie spróbuj tego:

www.example.com/%64ownloads/hot_new_file.zip

a filtr jest pominięty. Masz teraz pełny dostęp do katalogu pobierania bez uwierzytelniania.

Korzystanie z funkcji obsługi plików

Niektóre z technik przedstawionych w tej sekcji dotyczą tylko systemu Windows, ale zwykle można znaleźć sposób na obejście tego rodzaju problemów również na platformach Unix. Chodzi o to, aby oszukać aplikację, aby:

* Uważa, że w ścieżce pliku znajduje się wymagany ciąg.

* Uważa, że w ścieżce pliku nie ma niedozwolonego ciągu.

* Stosuje niewłaściwe zachowanie do pliku, jeśli obsługa pliku jest oparta na rozszerzeniu pliku.

Wymagany ciąg jest obecny w ścieżce

Pierwszy przypadek jest łatwy. W większości sytuacji, w których możesz przesłać nazwę pliku, możesz przesłać nazwę katalogu. W przeprowadzonym przez nas audycie napotkaliśmy sytuację, w której skrypt aplikacji internetowej obsłużyłby pliki pod warunkiem, że znajdowały się one na danej stałej liście. Zostało to zaimplementowane poprzez zapewnienie, że nazwa jednego z określonych plików:

* data/foo.xls

* dane/bar.xls

* data/wibble.xls

* data/wobble.xls

pojawił się w parametrze file_path. Typowe żądanie może wyglądać tak:

www.example.com/getfile?file_path=data/foo.xls

Interesujące jest to, że gdy większość systemów plików napotyka specyfikator ścieżki nadrzędnej, nie zadają sobie trudu, aby sprawdzić, czy wszystkie katalogi, do których się odnoszą, istnieją. W związku z tym byliśmy w stanie ominąć walidację, wysyłając żądania takie jak: www.example.com/getfile?file_path=data/foo.xls/../../../etc/passwd

Zabroniony ciąg nie występuje na ścieżce

Ta sytuacja jest nieco trudniejsza i znowu dotyczy katalogów. Załóżmy, że skrypt obsługujący plik, o którym mowa w poprzedniej sekcji, umożliwia nam dostęp do dowolnego pliku, ale zabrania używania specyfikatorów ścieżki nadrzędnej (/../) i dodatkowo zabrania dostępu do prywatnego katalogu danych, sprawdzając ten ciąg w parametrze file_path:

data/private

Możemy ominąć tę ochronę, wysyłając takie żądania, jak:

www.example.com/getfile?file_path=data/./private/accounts.xls

ponieważ specyfikator /./ nie robi nic w kontekście ścieżki. Podwojenie liczby ukośników („dane//prywatne”) może czasami osiągnąć podobny wynik.

Nieprawidłowe zachowanie na podstawie rozszerzenia pliku

Załóżmy, że administratorzy witryn sieci Web mają dość ludzi, którzy pobierają arkusze kalkulacyjne z ich kont i decydują się na zastosowanie filtra, który zabrania wszelkich parametrów file_path kończących się na .xls. Więc staramy się:

www.example.com/getfile?file_path=data/foo.xls/../private/accounts.xls

i to się nie udaje. Następnie próbujemy:

www.example.com/getfile?file_path=data/./private/accounts.xls

i to również zawodzi. Jednym z najbardziej interesujących aspektów systemu plików Windows NT NTFS jest obsługa alternatywnych strumieni danych w plikach, które są oznaczone dwukropkiem (:) na końcu nazwy pliku, a następnie nazwą strumienia. Możemy wykorzystać tę koncepcję, aby uzyskać dane konta. Po prostu prosimy:

www.example.com/getfile?file_path=data/./private/accounts.xls::$DATA

a dane zostaną nam zwrócone. Powodem tego jest to, że „domyślny” strumień danych w pliku nazywa się ::$DATA. Żądamy tych samych danych, ale nazwa pliku nie kończy się rozszerzeniem .xls, więc aplikacja na to pozwala. Aby zobaczyć to na własne oczy, uruchom następujące polecenie na polu NT (w woluminie NTFS):

echo foobar > foo.txt

Następnie uruchomić:

more < foo.txt::$DATA

i zobaczysz foobar. Oprócz możliwości mylenia walidacji danych wejściowych technika ta zapewnia również doskonały sposób na ukrycie danych. Błąd w usługach IIS sprzed kilku lat umożliwia odczytanie źródła stron ASP, żądając czegoś takiego:

www.example.com/foo.asp:::$DATA

To zadziałało z tego samego powodu.

Inną sztuczką związaną z rozszerzeniami plików w systemach Windows jest dodanie jednej lub więcej końcowych kropek do rozszerzenia. To sprawiłoby, że nasza prośba do skryptu obsługującego pliki stałaby się:

www.example.com/getfile?file_path=data/./private/accounts.xls.

W niektórych przypadkach otrzymasz te same dane. Czasami aplikacja uzna, że rozszerzenie jest puste; czasami pomyśli, że rozszerzenie to .xls. Ponownie możesz to szybko zaobserwować, uruchamiając

echo foobar > foo.txt

następnie

type foo.txt.

lub

notepad foo.txt...

Unikanie sygnatur ataku

Większość systemów IDS opiera się na jakiejś formie rozpoznawania ataków na podstawie sygnatur. W dziedzinie szelkodów ludzie opublikowali już wiele informacji na temat równoważności nop, ale chciałbym pokrótce poruszyć tę kwestię, ponieważ jest ona ważna. Kiedy piszesz szelkod, możesz wstawić prawie nieskończoną różnorodność instrukcji, które nic nie robią między instrukcjami, które mają znaczenie dla twojego exploita. Należy pamiętać, że te instrukcje nie muszą w rzeczywistości nic nie robić — po prostu nie muszą robić nic, co jest istotne dla stanu twojego exploita. Na przykład możesz wstawić do swojego szelkodu złożoną serię manipulacji stosem i ramką, przeplatając instrukcje z rzeczywistymi instrukcjami, które składają się na twój exploit. Możliwe jest również wymyślenie niemal nieskończonej liczby sposobów wykonania danego zadania w szelkodzie, takich jak wrzucanie parametrów na stos lub ładowanie ich do rejestrów. Dość łatwo jest napisać generator, który przyjmuje jedną formę asemblera dla exploita i wypluwa funkcjonalnie identyczny exploit bez wspólnych sekwencji kodu.

Pokonywanie ograniczeń długości

W niektórych przypadkach dany parametr aplikacji jest obcinany do stałej długości. Ogólnie jest to próba ochrony przed przepełnieniem bufora, ale czasami jest używany w aplikacjach internetowych jako ogólny mechanizm obrony przed wstrzyknięciem SQL lub wykonywaniem poleceń. Istnieje wiele technik, które mogą pomóc w takiej sytuacji.

Dane małp morskich

W zależności od charakteru danych, możesz być w stanie przesłać jakąś formę danych wejściowych, która rozwija się w aplikacji. Na przykład w większości aplikacji internetowych kończysz kodowanie znaków podwójnego cudzysłowu jako:

"

czyli stosunek sześciu znaków do jednego. Każdy znak, który prawdopodobnie zostanie „uciekła” w danych wejściowych, jest dobrym kandydatem do tego rodzaju rzeczy – pojedyncze cudzysłowy, ukośniki odwrotne, znaki pionowe i symbole dolara są pod tym względem całkiem dobre. Jeśli możesz przesłać sekwencje UTF-8, prześlij zbyt długie sekwencje, ponieważ mogą być traktowane jako pojedynczy znak. Możesz mieć szczęście i natknąć się na aplikację, która traktuje wszystkie znaki spoza zestawu ASCII jako 16-bitowe. Możesz następnie przepełnić go, nadając mu znaki, które są dłuższe niż to, w zależności od tego, jak oblicza długość ciągu.

%2e is the URL encoding for (.). However:

%f0%80%80%ae

and

%fc%80%80%80%80%ae

są również kodowaniami (.).

Szkodliwe obcięcie - zerwanie znaków ucieczki

Najbardziej oczywistym zastosowaniem tej techniki jest SQL Injection, chociaż mając na uwadze wcześniejszą dyskusję na temat kanonizacji, możliwe jest wymyślenie wielu interesujących sposobów zastosowania tej techniki wszędzie tam, gdzie używane są dane rozdzielane lub dane ze znakami ucieczki. Uruchamianie poleceń w perlu jest dobre do ewentualnego wstrzykiwania do strumienia SMTP. Zasadniczo, jeśli dane są jednocześnie pomijane i obcinane, czasami można wyrwać się z wyznaczonego obszaru, upewniając się, że obcięcie nastąpi w środku sekwencji ucieczki. Istnieje oczywisty przykład SQL Injection: Jeśli aplikacja, która unika pojedynczych cudzysłowów przez ich podwojenie, akceptuje nazwę użytkownika i hasło, nazwa użytkownika jest ograniczona do (powiedzmy) 16 znaków, a hasło jest również ograniczone do 16 znaków, następująca nazwa użytkownika/ kombinacja hasła spowoduje wykonanie polecenia zamknięcia, które spowoduje zamknięcie programu SQL Server:

Nazwa użytkownika: aaaaaaaaaaaaa”

Hasło: ‘wyłączenie

Aplikacja próbuje uniknąć pojedynczego cudzysłowu na końcu nazwy użytkownika, ale ciąg jest następnie cięty do 16 znaków, usuwając „uciekający” pojedynczy cudzysłów. W rezultacie pole hasła może zawierać trochę kodu SQL, jeśli zaczyna się od pojedynczego cudzysłowu. Zapytanie może wyglądać tak: wybierz * spośród użytkowników, gdzie nazwa użytkownika = „aaaaaaaaaaaaaaa” i hasło = „”

zamknąć

W efekcie nazwa użytkownika w zapytaniu stała się:

aaaaaaaaaaaaa” i hasło =”

więc uruchamiany jest końcowy kod SQL, a program SQL Server zostaje zamknięty. Ogólnie rzecz biorąc, ta technika ma zastosowanie do wszelkich danych o ograniczonej długości, które zawierają sekwencje specjalne. Istnieją oczywiste zastosowania tej techniki w

świat perla, ponieważ aplikacje perla mają tendencję do wywoływania zewnętrznych skryptów.

Wiele prób

Nawet jeśli wszystko, co możesz zrobić, to zapisać pojedynczą wartość gdzieś w pamięci, możesz normalnie przesłać i wykonać kod powłoki. Nawet jeśli nie masz miejsca na dobry exploit (być może przepełniasz 32-bajtowy bufor, chociaż to wystarczy na execve lub winexec, gdy pozostało miejsce), nadal możesz wykonać dowolny kod, zapisując mały ładunek w jakieś miejsce w pamięci. Tak długo, jak możesz to zrobić wiele razy, możesz zbudować swój exploit w dowolnym miejscu w pamięci, a następnie (po przesłaniu całości) go uruchomić, ponieważ już wiesz, gdzie on jest. Ta technika jest bardzo podobna do doskonałej techniki wykorzystywania niewykonywalnego stosu podczas wykorzystywania błędów ciągu formatującego. Ta metoda może nawet mieć zastosowanie w sytuacji przepełnienia sterty, chociaż proces docelowy musiałby być bardzo dobry w obsłudze wyjątków. Po prostu używasz prymitywu „zapisz wszystko w dowolnym miejscu" z powtarzającymi się próbami budowania ładunku, a następnie wyzwalasz go, nadpisując wskaźnik funkcji, adres obsługi wyjątków, VPTR lub cokolwiek innego.

Bezkontekstowe ograniczenia długości

Czasami dany element danych można przesłać wiele razy w danym zestawie danych wejściowych, z ograniczeniem długości stosowanym do każdego wystąpienia danych, ale dane są następnie łączone w pojedynczy element, który przekracza długość. Dobrym tego przykładem jest pole nagłówka hosta HTTP w kontekście technologii Web Intrusion Prevention. Nie jest niczym niezwykłym, że te rzeczy traktują każdy nagłówek oddzielnie od innych. Apache (na przykład) połączy następnie nagłówki hosta w jeden długi nagłówek hosta, skutecznie omijając limit długości nagłówka hosta. IIS robi coś podobnego. Możesz użyć tej techniki w dowolnym protokole, w którym każdy element danych jest identyfikowany za pomocą nazwy, takim jak SMTP, parametry HTTP, pola formularzy i zmienne plików cookie, atrybuty tagów HTML i XML oraz (w rzeczywistości) dowolny mechanizm wywoływania funkcji, który akceptuje parametry wg nazwy.

Windows 2000 SNMP DOS

Chociaż nie jest to wyjątkowo ekscytujący błąd, to wydanie dość dobrze ilustruje zasady instrumentalnej techniki śledczej. Odpowiedni artykuł z bazy wiedzy Microsoft Knowledge Base można znaleźć pod adresem support.microsoft.com/default.aspx?scid=kb;en-us;Q296815, a poradę NGS można znaleźć pod adresem www.ngssoftware.com/advisories/snmp-dos/ . W chwili znudzenia podczas testowania jakiegoś krótkiego kodu SNMP (wspólnego dla implementatorów SNMP), postanowiliśmy sprawdzić, czy nie możemy spowodować przepełnienia demona SNMP SNMP. Przeszliśmy przez zwykły proces dołączania debugera, RegMon i FileMon; szybkie zajrzenie do HandleEx, aby zobaczyć, jakie zasoby otworzył demon SNMP; i używając Monitora wydajności, aby śledzić, z jakich zasobów korzystał proces SNMP - a następnie przeprowadziliśmy kilka szybkich testów, uruchamiając kilka ręcznych żądań ze zniekształconą strukturą BER (długości nie odpowiadają poprawnie i tak dalej). Wyglądało na to, że niewiele się wydarzyło, więc przyjrzeliśmy się, jakie identyfikatory SNMP OID były obecne, gdy spacerowaliśmy po całym drzewie. Znowu nic strasznie interesującego nie wydawało się obecne, ale kiedy wróciliśmy do Monitora wydajności, zauważyliśmy, że demon najwyraźniej przydzielił około 30 MB pamięci. Uruchamiając kolejny spacer SNMP, proces SNMP ponownie przydzielił dużą ilość pamięci. Następnie przeszliśmy przez kod spaceru SNMP,

uważnie obserwując ilość pamięci przydzielonej przez proces SNMP. Odkryliśmy, że problem pojawił się podczas żądania wartości związanych z drukarką w MIB LanMan. Okazuje się, że pojedyncze żądanie SNMP (czyli pojedynczy pakiet UDP) powoduje przydział 30 MB. Zużycie w ten sposób całej dostępnej pamięci jest śmiesznie łatwe (i bardzo szybkie), przy kilku tysiącach pakietów, a cały serwer jest uszkodzony. Żadne nowe procesy nie zostaną uruchomione, nie zostaną utworzone żadne nowe okna, a jeśli ktoś spróbuje się zalogować (na przykład w celu próby zamknięcia usługi SNMP lub samego serwera), zakończy się niepowodzeniem, ponieważ Microsoft Graphical Identification and Authentication (GINA), biblioteka DLL, która kontroluje logowanie, nie ma wystarczającej ilości dostępnej pamięci, aby utworzyć okna dialogowe potrzebne do uzyskania poświadczeń użytkownika. Jedynym wyjściem jest wyłączenie zasilania. Tak więc w tym przypadku wykrycie błędu opierało się na uważnej obserwacji wykorzystania pamięci w docelowym procesie. Gdybyśmy nie przyglądali się zużyciu pamięci, nigdy nie zobaczylibyśmy błędu.

Znajdowanie ataków DOS

Poprzedni przykład ilustruje kolejną doskonałą technikę wyszukiwania ataków DOS - ścisłe monitorowanie wykorzystania zasobów. Wyeliminowanie wycieków zasobów jest trudnym problemem, a większość dużych aplikacji ma wycieki w takiej czy innej formie. Jest to rodzaj błędu, który jest łatwy do wykrycia przy dobrym oprzyrządowaniu i prawie niemożliwy bez niego. Jak więc mamy monitorować tego rodzaju rzeczy? W Linuksie drzewo proc jest dość pouczające (man proc); zawiera listę plików, które proces ma otwarte (fd), obszarów pamięci zmapowanych przez proces (mapy) oraz rozmiaru pamięci wirtualnej w bajtach (stat/vsize). statm jest również nieco przydatny; dostarcza informacje o stanie pamięci oparte na stronach. W systemie Windows historia jest nieco inna. Standardowy menedżer zadań może być pomocny w zorientowaniu się w wykorzystaniu zasobów, ponieważ możesz dość łatwo zmienić kolumny wyświetlane w zakładce procesów. Przydatnymi rzeczami, na które należy zwrócić uwagę, są liczba uchwytów, użycie pamięci i rozmiar maszyny wirtualnej. Lepszym sposobem monitorowania zasobów w procesie (jeśli poważnie podchodzisz do oprzyrządowania) jest Monitor wydajności systemu Windows, który można uruchomić, uruchamiając perfmon.msc w systemie Windows 2000 lub za pomocą opcji menu Start Narzędzia administracyjne. Monitor wydajności jest doskonałym źródłem informacji liczbowych o procesach, ponieważ umożliwia tworzenie niestandardowych histogramów zawierających wszystkie elementy, które chcesz monitorować w procesie. Daje to wgląd w wykorzystanie zasobów w czasie, a nie tylko liczbę miejsc, co ułatwia dostrzeżenie wzorców. Przydatne liczniki do dodania do wykresu podczas testowania określonego procesu znajdują się zazwyczaj w obiekcie wydajności procesu — takie jak liczba obsługi, liczba wątków i statystyki użycia pamięci. Jeśli będziesz monitorować te liczby w czasie, będziesz znacznie bardziej skłonny znaleźć problemy z wyciekiem zasobów DOS.

SQL-UDP

Robak Slammer wykorzystał ten błąd SQL-UDP. Poradnik NGS na ten temat można znaleźć pod adresem www.ngssoftware.com/advisories/mssql-udp.txt. W trakcie konsultingu dla klienta firma NGS została poproszona przez klienta o przyjrzenie się różnym protokołom obsługiwanym przez SQL Server, ponieważ stanowiły one przedmiot zainteresowania klienta. W szczególności klient widział ruch UDP w sieci i był świadomy możliwości sfałszowania pakietów UDP. Klient był zaniepokojony konsekwencjami dla bezpieczeństwa tego dziwnego protokołu opartego na UDP i chciał jasno ustalić, czy powinien blokować ten ruch w sieci. Zespół zaczął analizować protokół. Na podstawie informacji opublikowanych przez Chipa Andrewsa dotyczących jego wspaniałego narzędzia sqlping, zespół był świadomy, że wysyłając jednobajtowy pakiet UDP zawierający bajt 0x02, docelowy serwer SQL odpowiadałby szczegółowymi protokołami używanymi do łączenia się z różne instancje SQL Server działające na hoście. Oczywistym punktem wyjścia było zatem sprawdzenie, co zrobiły inne wiodące

bajty w pakiecie (0x00, 0x01, 0x03 itd.). Zespół oprzyrządował różne instancje SQL Server za pomocą FileMon, RegMon, debuggerów itd. i zaczął wysyłać żądania.

David zauważył (poprzez RegMon), że gdy pierwszy bajt pakietu UDP miał wartość 0x04, SQL Server próbował otworzyć klucz rejestru w postaci:

HKLM\Software\Microsoft\Microsoft SQL

Server\<contents_of_packet>\MSSQLServer\CurrentVersion

Następną rzeczą do zrobienia było wyraźne dodanie dużej liczby bajtów do pakietu. Rzeczywiście, SQL Server przewrócił się z waniliowym przepełnieniem stosu. W tym momencie było całkiem jasne, że klient naprawdę powinien pomyśleć o zablokowaniu UDP 1434 w całej sieci. Zespół kontynuował, dochodzenie trwało do tej pory około pięciu minut. Kilka innych wiodących bajtów wykazywało interesujące zachowania. 0x08 wywołało przepełnienie sterty, gdy po bajcie wiodącym nastąpił długi ciąg, dwukropek, a następnie liczba. 0x0a powodowało, że SQL Server odpowiadał pakietem zawierającym pojedynczy bajt 0x0a - w związku z tym można było łatwo ustawić odmowę usługi wykorzystania sieci poprzez sfałszowanie adresu źródłowego jednego SQL Servera i wysłanie pakietu z 0x0a do innego SQL Serwer.

Wniosek

Odchodząc na chwilę do kwestii społecznych związanych z badaniem podatności, przerażającą rzeczą w błędzie SQL-UDP była szybkość, z jaką wykrywano przepełnienie stosu; wystarczyło dosłownie pięć minut śledztwa. Było dla nas oczywiste, że gdybyśmy mogli znaleźć błąd tak szybko, inni, być może mniej odpowiedzialni, ludzie również by go znaleźli i prawdopodobnie wykorzystaliby do złamania zabezpieczeń systemów. Zgłosiliśmy błąd do firmy Microsoft w zwykły sposób i zarówno my, jak i firma Microsoft bardzo głośno mówiliśmy o próbach nakłonienia organizacji do zastosowania poprawki i zablokowania UDP 1434 (jest używany tylko wtedy, gdy klient SQL nie jest pewien, jak połączyć się z serwerem SQL instancja). Niestety, wiele organizacji nie zrobiło nic w sprawie błędu, a następnie, dokładnie sześć miesięcy po opublikowaniu łaty, pewna (nieznana jeszcze) osoba zdecydowała się napisać i wydać robaka Slammer, powodując znaczne przeciążenie Internetu i przysparzając administracyjnego bólu głowy. tysiące organizacji. Chociaż prawdą jest, że robak Slammer mógł być znacznie gorszy, to wciąż przygnębiające było to, że ludzie nie chronili się wystarczająco, by go udaremnić. Trudno jest przewidzieć, co firmy zajmujące się bezpieczeństwem mogą zrobić, aby zapobiec występowaniu tego rodzaju problemów w przyszłości. We wszystkich najczęściej zgłaszanych przypadkach — Slammer, Code Red (na podstawie błędu IIS .ida wykrytego przez innego współautora tej książki, Rileya Hassela), Nimda (ten sam błąd) i robaka Blaster (na podstawie RPC). -Błąd DCOM wykryty przez grupę The Last Stage of Delirium) — zaangażowane firmy współpracowały odpowiedzialnie z dostawcami, aby zapewnić dostępność poprawki i dobrej jakości informacji o obejściu przed opublikowaniem informacji o lukach. Jednak w każdym przypadku ktoś zbudował robaka, który wykorzystał błąd i wypuścił go, powodując ogromne szkody. Kuszące jest, aby przestać badać błędy w oprogramowaniu, gdy coś takiego się dzieje, ale alternatywa jest znacznie gorsza. Badacze nie tworzą błędów, tylko je znajdują. Firma Microsoft wydała 72 różne poprawki zabezpieczeń w 2002 r., 51 w 2003 r., 45 w 2004 r., 55 w 2005 r. i 78 w 2006 r. Wiele z tych poprawek naprawiło wiele błędów zabezpieczeń. Jeśli jesteś użytkownikiem Linuksa, nie lekceważ tych problemów. Według przeprowadzonej na koniec roku ankiety US-CERT 2005, Linux miał w tym roku więcej luk w zabezpieczeniach niż Windows — chociaż liczba ta jest nieco trudna do ustalenia, ponieważ zależy to od tego, jak kategoryzujesz problemy. Błędy SSH i Apache SSL oraz błędy kodowania fragmentarycznego są dobrymi przykładami problemów z Linuksem. Jeśli jesteś użytkownikiem Macintosha, nadal nie możesz odrzucić Microsoftu z jego wirusami i robakami oraz Linuksa z jego

błędami SSH i SSL oraz mnóstwem problemów związanych z podnoszeniem uprawnień. Liczba osób aktywnie wyszukujących i publikujących błędy na platformie Mac jest obecnie niewielka, ale to, że nikt nie szuka błędów, nie oznacza, że ich tam nie ma. Czas pokaże. Jeśli wyobrazisz sobie świat, w którym nikt nie przeprowadzał żadnych badań nad lukami w zabezpieczeniach – ani z powodów prawnych, ani dlatego, że po prostu nie można im było zawracać głowy – wszystkie te fenomenalnie niebezpieczne błędy nadal byłyby dostępne i dostępne dla każdego, kto chciałby przejąć kontrolę naszych maszyn i sieci z dowolnego powodu. Nie mielibyśmy nadziei na obronę przed przestępcami, rządami, terrorystami, a nawet konkurentami komercyjnymi z powodu braku informacji. Ponieważ ludzie znaleźli te błędy, sprzedawcy musieli je naprawić i dlatego mieliśmy pewien stopień obrony. Badanie podatności to po prostu proces zrozumienia, co działa na twoim komputerze. Naukowcy nie tworzą wad tam, gdzie wcześniej ich nie było; po prostu rzucają światło na to, co my (i nasi klienci) uruchamiamy w naszych sieciach. Mam nadzieję, że ta książka pomoże ci zrozumieć problemy i dalej naświetlić temat.