Metody automatycznego wytwarzania sygnatur zagrożeń sieciowych

Piotr Kijewski
CERT Polska/NASK
e-mail: piotr.kijewski (at) cert.pl

Wstęp

Wytwarzanie sygnatur zagrożeń sieciowych, które są stosowane w systemach wykrywania lub zapobiegania włamaniom, jest w większości przypadków procesem ręcznym, a co za tym idzie podatnym na błędy i powolnym. Tymczasem reakcja na nowo pojawiające się zagrożenia musi być szybka, jeśli ma być skuteczna, a jednocześnie dobrze ukierunkowana, tak, aby nie nieść ze sobą niepożądanych, wręcz szkodliwych skutków ubocznych. Próba osiągnięcia powyższego efektu jest celem metod automatycznego wytwarzania sygnatur zagrożenia, opracowywanych w ramach projektu ARAKIS zespołu CERT Polska.

Definicja sygnatury zagrożenia

Sygnaturę można zdefiniować jako zbiór lub podzbiór cech charakteryzujących dane zagrożenie. Cechy mogą być bardzo różne np. informacje z pól nagłówka pakietów, zawartość payloadu pakietów, analizy częstości wystąpień znaków, wywołań funkcji systemowych czy zależności czasowe pomiędzy zdarzeniami. Z punktu widzenia bezpieczeństwa, najistotniejszą cechą danego zagrożenia jest sposób ataku, a nie działania po uzyskaniu dostępu do systemu. Nowy, nieznany sposób propagacji sprawia, że zagrożenie ma większy zasięg. W związku z tym, w artykule pisząc o sygnaturze zagrożenia mamy na myśli **sygnaturę ataku**, z którego dane zagrożenie korzysta, aby uzyskać dostęp do systemu.

Cechy dobrej sygnatury

Poza **szybkością** wytwarzania, **wykrywaniem ataku** i **niskim współczynnikiem fałszywych alarmów**, z praktycznego punktu widzenia istotne jest, aby sygnatura była niezależna od protokołów warstw aplikacji. Oznacza to, że do skutecznego zastosowania sygnatury nie będzie konieczne zrozumienie przez system wykrywania lub zapobiegania włamaniom protokołu aplikacji, której dotyczy. Pozwala to na podwójną korzyść wynikającą z **uniwersalności** sygnatury: a) sygnatura może być zastosowana w większej liczbie systemów b) może być wykorzystana względem zupełnie nowych protokołów.

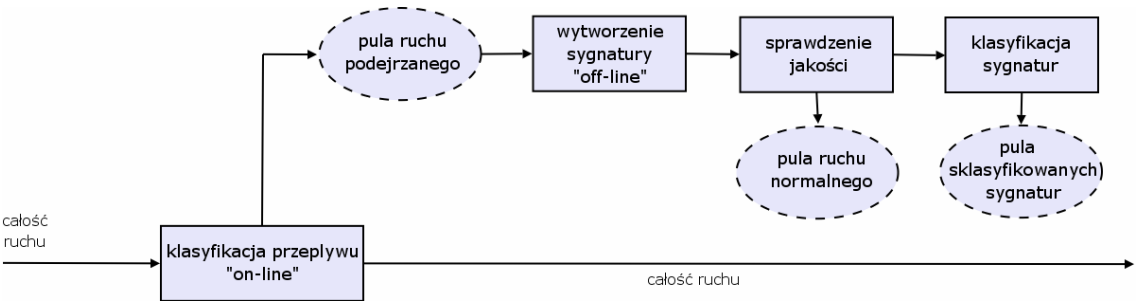
Z punktu widzenia prewencji ataku pożądane jest, aby sygnatura ataku wyodrębniała podatność, którą wykorzystuje. Dysponujemy wtedy sygnaturą bardziej ogólną, niezależną od exploita użytego do przeprowadzenia ataku. Jednakże, z punktu widzenia informacyjnego wskazane jest również uzyskanie

sygnatury jednoznacznie identyfikującej exploita. Pierwszy rodzaj sygnatur jest trudniejszy do otrzymania.

Kryterium szybkości i uniwersalności sugeruje wykorzystanie „standardów de facto" w zakresie sygnatur, będących podstawowym elementem dzisiejszych systemów wykrywania i prewencji włamań. Oznacza to, że aby możliwe było praktyczne podejście do zagadnienia, stosować należy sygnatury oparte o ciąg bajtów, które charakteryzują zagrożenie.

Ogólny model systemu wytwarzania sygnatur

Podstawą wytwarzania sygnatur jest identyfikacja ruchu sieciowego, który ma być poddany procesowi przetwarzania. Pierwszym etapem jest wykrycie anomalii w ruchu w sieci i sklasyfikowanie przepływu jako atak bądź nie. Istotna jest **nowość** danego zjawiska oraz jego **powtarzalność**. Zakłada się, że nowe zagrożenie – w szczególności zautomatyzowane (robak, bot itp.) cechuje się powtarzalnością działań niezbędnych do propagacji. Jeżeli dysponujemy zbiorem przepływów związanych z danym zagrożeniem możemy przystąpić do prób opisu cech charakterystycznych danego zagrożenia, czyli wygenerować sygnaturę. Oddzielenie tego procesu od procesu wykrywania anomalii zezwala na wykorzystanie bardziej złożonych algorytmów wytwarzania, które ze względów wydajnościowych nie sprawdzą się w trybie „on-line". Wynik procesu wytwarzania sygnatury jest następnie poddawany weryfikacji względem próbek ruchu, o którym wiadomo, że jest „normalny", niezwiązany z żadnym atakiem. Jest to miara jakości wytworzonej sygnatury. Kolejnym krokiem jest próba klasyfikacji sygnatury na podstawie podobieństwa sygnatury z poprzednio wytworzonymi sygnaturami, zakładając, że poprzednio wytworzone sygnatury posiadają etykiety.



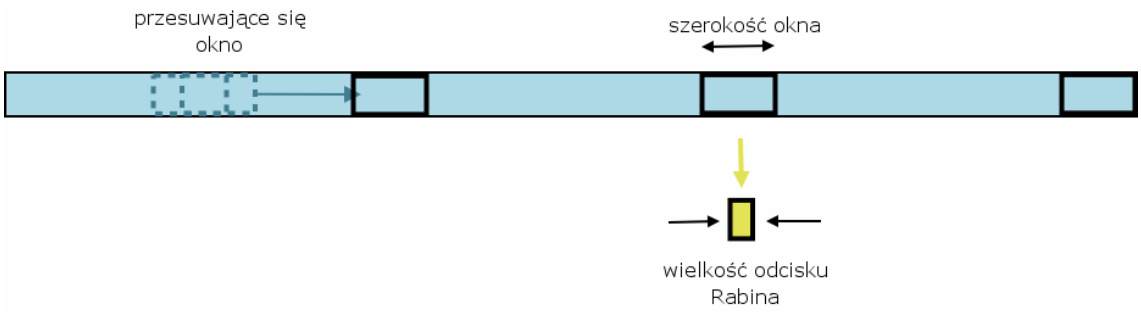
Rys 1. Model systemu wytwarzania sygnatur

Porównywanie przez odciski, czyli identyfikacja i wytworzenie sygnatury w jednym

Najprostszą metodą identyfikacji ataków jest porównywanie i katalogowanie pakietów w sieci za pomocą skrótów kryptograficznych - na przykład MD5. Często powtarzające się pakiety z różnych źródeł do różnych hostów docelowych mogą być oznaką nowo propagującego się zagrożenia. W tym wypadku skrót MD5 staje się sygnaturą zagrożenia. Uruchomienie obserwacji na produkcyjnej sieci o dużej przepustowości wiąże się jednak z problemami

wydajnościowymi oraz z dużą liczbą fałszywych alarmów, które wynikają z tego, że w większości monitorowany jest ruch legalny. Zastosowanie metody staje się bardziej efektywne w środowisku honeynet'owym, gdzie ruch jest mniejszy oraz w większości związany z zagrożeniami. Metoda ta jest często stosowana, np. w projekcie Internet Motion Sensor (IMS) [1]. Podejście nie jest jednak bez wad – każda drobna modyfikacja ataku powoduje wytworzenie nowego skrótu MD5. Nie zostaje wyodrębniona sekwencja bajtów stanowiąca istotę ataku ale cały pakiet.

W celu wyodrębnienia ataku i lepszej identyfikacji wariantów - zamiast uwzględniać cały pakiet - możliwe jest zastosowanie mechanizmu przesuwającego się okna. Na stale przesuwającym się oknie o określonej długości w obrębie tego samego pakietu liczony jest odcisk, który następnie jest porównywany między pakietami. Po przekroczeniu pewnego progu, określanego za pomocą unikalnej liczby źródeł i hostów docelowych, sygnalizowany jest alarm o potencjalnym wykryciu zagrożenia. W tym celu dla każdego pakietu liczonych jest wiele odcisków, których liczba zależna jest od długości pakietu i szerokości okna (jeżeli s to wielkość pakietu w bajtach, a β to długość okna, liczba odcisków dla pakietu wynosi $s - \beta + 1$).



Rys 2. Przesuwające się okno na przepływie

Konieczności wyliczenia dużej liczby skrótów sprawia, że ze względów wydajnościowych stosowanie skrótów kryptograficznych staje się problematyczne. W ich miejsce zastosować można odciski Rabina, które są podstawą algorytmu Rabina-Karpa [2], jednego z najszybszych algorytmów służących do wyszukiwania łańcuchów. Odciski Rabina dobrze nadają się do stosowania w metodyce przesuwającego się okna, ponieważ umożliwiają obliczenie skrótu okna przesuniętego o jeden bajt na podstawie obliczeń poprzedniego okna. Odciski Rabina zostały po raz pierwszy zaproponowane do wykrywania zagrożeń sieciowych w [3].

Aby jeszcze bardziej poprawić wydajność rozwiązania (w szczególności jeśli przechowujemy payload związany z danym odciskiem), możliwe jest zastosowanie maski bitowej do próbkowania odcisków. Próbkowanie odbywa się po wartościach odcisków, a ich końcowa liczba wynosi $1/2^k$ gdzie k to liczba bitów w masce. Jeżeli maska ustalona jest na 4 bity (np. najmniej istotne 4 bity odcisku powinny mieć wartość 0), to spróbkowane zostaje 1 na 16 odcisków.

Powyższy sposób na poprawę wydajności odbywa się jednak kosztem potencjalnego przegapienia zagrożenia, którego odciski nie zostaną

spróbkowane. Prawdopodobieństwo wykrycia zagrożenia o długości β wynosi $½^k$. Prawdopodobieństwo wykrycia zagrożenia o długości x wynosi $1-e^{-f(x-\beta+1)}$ gdzie $f=½^k$ [3]. Przykładowo, zakładając maskę składającą się z 4 bitów, zagrożenie o długości 100b i długość okna równą 32b, prawdopodobieństwo wykrycia wynosi 98,66%.

Im krótszy odcisk, tym większe prawdopodobieństwo wykrycia zdarzenia ale też większa możliwość fałszywego alarmu. Według [4] konieczne jest zastosowanie okna o długości co najmniej 150b w celu uniknięcia fałszywych alarmów (ze względu na długość zapytań w protokole Microsoft RPC).

Innym sposobem na poprawę wydajności przesuwających się okien Rabina, jest monitorowanie tylko jednej strony przepływu. Kiedy interesują nas ataki przekazywane przez stronę inicjującą połączenie jak w przypadku zagrożeń propagujących się przez skanowanie, liczymy odciski tylko na nich. W przypadku, kiedy interesują nas zagrożenia rozprowadzane przez serwer (pasywna metoda propagacji) odciski Rabina liczone są tylko na przepływach od strony serwera.

Zaletą wykorzystania odcisków Rabina, wynikającą z ich wysokiej wydajności jest możliwość ich wykorzystania nie tylko w środowisku honeynetowym ale także w sieciach produkcyjnych.

Porównywanie przez odciski może służyć zarówno do klasyfikacji pakietu (przepływu) jak i do wytworzenia sygnatury. W przypadku zestawu okien Rabina, kiedy wiele odcisków kojarzonych jest z jednym pakietem wskazane staje się ustalenie liczników powtórzeń tak, by były kojarzone ze zbiorem odcisków (np. ustala się, że zbiór odcisków pakietów musi się zgadzać w określonym procencie) a nie pojedynczym odciskiem. Sygnaturą staje się wtedy wybrany odcisk Rabina.

Wytwarzanie sygnatur w trybie „off-line”

Jeżeli odciski Rabina mogą być wytworzone w czasie prawie rzeczywistym, dlaczego bierzemy pod uwagę wydzielenie generatora sygnatur do modułu „off-line”? Wiąże się to z:

- Ograniczeniami wynikającymi z długości okna. Im mniejsza długość okna tym większe prawdopodobieństwo wykrycia ataku, ale tym większa możliwość fałszywych alarmów, jeżeli wykorzystamy wybrany odcisk jako sygnaturę. Z kolei im dłuższe okno, tym większe prawdopodobieństwo, że wyodrębniliśmy atak, ale kosztem przegapienia ataków o krótszych payloadach.
- Ograniczeniami wynikającymi z próbkowania odcisków Rabina. Wylosowane próbki odcisków Rabina mogą nie w pełni wyodrębniać istotę ataku (najistotniejszy element exploita).
- Polimorfizmem. Ataki polimorficzne będą w stanie ominąć wykrycie za pomocą okien Rabina, ponieważ nie będą zawierać w sobie takich samych sekwencji bajtów dostatecznie długich by zapełnić okno.
- Wydajnością. Im więcej obliczeń w trybie „on-line” tym większe opóźnienie oprogramowania monitorującego.

Aby wykryć powtarzające się ciągi znaków wykorzystać można inne metody niż porównywanie przez odciski. Taką możliwość daje algorytm wyszukujący najdłuższy wspólny ciąg znaków między pakietami (przepływami), znany jako LCS (*Longest Common Substring*). Algorytm został po raz pierwszy wykorzystany do generacji sygnatur w systemie honeycomb [5], który działa jako wtyczka honeyd. Problemem honeycomb jest jego słaba wydajność oraz duża liczba sygnatur, które wytwarza i którymi później trudniej jest zarządzać, ponieważ porównywane są wszystkie przepływy ze wszystkimi przechowywanymi w pamięci[1], bez wstępnej klasyfikacji. W tej sytuacji system nie nadaje się do monitorowania większych przestrzeni, niż kilka pojedynczych IP.

W celu zaradzenia powyższym, **proponujemy metodę, która opiera się zarówno na oknach Rabina jak i LCS**. Okna Rabina są wykorzystywane do wstępnej klasyfikacji przepływu na bazie payloadu pakietów. Na bazie tej klasyfikacji odbywa się wykrycie anomalii: zakończone przepływy są grupowane według ich podobieństwa w sensie Rabina za pomocą reguł (np. grupowane są wszystkie zakończone przepływy, które posiadają 20% odcisków podobnych). Dodatkowo każda reguła sprawdza grupy według określonych heurystyk. Przykładowo, sprawdzana jest liczba unikalnych źródłowych IP w określonym przedziale czasu, stanowiących jedną grupę. Jeżeli przekroczony zostanie pewien próg, zestaw przepływów grupy przekazywany jest innym analizom, które ze względów wydajnościowych działają w trybie „off-line".

Przedstawicielem tej grupy analiz jest algorytm LCS. Na grupie podobnych przedstawicieli wyszukany zostaje najdłuższy wspólnych ciąg bajtów. Jeżeli porównujemy sumaryczny payload z różnych przepływów uzyskuje się jedną sygnaturę zagrożenia. Jeżeli porównujemy pakiet z pakietem dla różnych przepływów, uzyskać możemy sygnaturę exploita (być może również i podatności).

Powyższe podejście zezwala na wykorzystanie okien Rabina o niewielkiej długości, zwiększając prawdopodobieństwo wyszukania wspólnych ciągów przepływów, natomiast o końcowej sygnaturze decyduje bardziej elastyczny LCS, wydłużający wykrytą sygnaturę.

Zauważmy, że stosowanie Rabina w tym przypadku nie oznacza braku możliwości wykrycia zagrożeń polimorficznych za ich pośrednictwem, o ile monitorujemy honeynet a nie sieć rzeczywistą. Ataki polimorficzne wykrywane są za pomocą reguły, która, grupując pakiety za pomocą podobieństwa Rabina, przekazuje do dalszej analizy jedynie grupy składające się z pojedynczych przedstawicieli. W tym wypadku ataki polimorficzne są wyróżniane na podstawie braku podobieństwa Rabina.

[1] Porównanie za pomocą biblioteki libstree trwa od paruset milisekund do ponad sekundy, w zależności od długości porównywanych ciągów (do 6kB) na procesorze Pentium IV Xeon 2.8 Ghz

Oczywiście wygenerowanie sygnatury LCS na przepływie polimorficznym jest mało prawdopodobne (w zależności od jakości generatora wykorzystywanego przez zagrożenie polimorficznego). Jak wykazano w [6], możliwe jest jednak wytworzenie sygnatur składających się z rozłącznych ciągów bajtów za pomocą innych algorytmów, gdyż nawet zakładając idealny polimorfizm ataków, istnieją bardzo krótkie sekwencje bajtów, które muszą pozostać niezmiennie, tak aby dany exploit był w stanie funkcjonować. Algorytmy te (w tym algorytm Smitha-Watermana) rozpatrywane są jako kolejni kandydaci do implementacji. Warto jednak przypomnieć, że do tej pory żaden samodzielnie propagujący się kod w sieci nie wykorzystał polimorficznych exploitów. Najbliższy temu był robak Witty [7], który losowo uzupełniał pakiet z exploitem.

Redukcja fałszywych alarmów

Jakość sygnatury nie ma krytycznego znaczenia w sytuacji, w której pełni rolę informacyjną, pozwalającą na zrozumienie kontekstu wystąpienia danego zjawiska w sieci. W przypadku, kiedy chcemy ją jednak wykorzystać do ochrony, zatrudniając w systemie IDS/IPS konieczna jest pewność, że sygnatura nie zablokuje ruchu legalnego. W tym celu po wytworzeniu danej sygnatury należy sprawdzić czy za jej pomocą nie wychwytuje się ruchu legalnego. Konieczne jest przechowywanie puli ruchu, o której wiemy, że nie zawiera przepływów związanych z zagrożeniami. W przypadku wykorzystania algorytmu LCS wydawać by się mogło, że im dłuższa sygnatura, tym większe prawdopodobieństwo, że zawiera ona atak - stąd tym mniejsza szansa na fałszywy alarm. Jednakże w przypadku, kiedy protokół składa się z długich nagłówków (jak np. w przypadku zapytań HTTP z przeglądarek), możliwe jest, że LCS wykryje właśnie te stałe protokołu jako najdłuższy wspólny łańcuch znaków. Najdłuższy ciąg nie musi wcale oznaczać najlepszego.

Klasyfikacja sygnatur

Klasyfikacja wytworzonej sygnatury jest ważną częścią oceny nowego zjawiska występującego w sieci. Proces ten można zautomatyzować poprzez porównanie nowo wytworzonej sygnatury z uprzednio sklasyfikowaną sygnaturą. Proces porównania sygnatur może odbywać się zaraz po jej wytworzeniu. Sprawdzane jest, czy wytworzona sygnatura już istnieje i jest sklasyfikowana, czy jest podzbiorem istniejącej sygnatury, a jeśli nie, to czy jest podobna do którejś istniejącej. Najbardziej złożony jest proces porównywania podobieństwa z istniejącą sygnaturą. Po pierwsze, należy określić funkcję podobieństwa, którą się posługujemy. Po drugie, biorąc pod uwagę liczbę sygnatur, wskazane jest wybranie tylko reprezentantów poszczególnych klas do porównania.

Funkcje podobieństwa zdecydowaliśmy się oprzeć na idei *q-gramów* [8] Pomysł jest podobny do stosowanych wcześniej okien Rabina, z tą różnicą, że nie liczony jest odcisk okna. Sygnatura LCS poddawana jest przesuwającemu się oknu o krótkiej długości 3-4 znaków, nazwanego przez nas tokenem. Stopień podobieństwa między sygnaturami LCS jest określany na podstawie części wspólnej wszystkich okien z różnych sygnatur.

Aby wybrać reprezentantów grup do porównania, wykorzystujemy algorytmy klasteryzacji. Przykładowym algorytmem przez nas zaimplementowanym jest znany algorytm *dbscan* [9]. Zbiór historycznych sygnatur jest okresowo dzielony na klastry, zgodnie z eksperymentalnymi metryką. Sygnatura tworząca rdzeń danego klastra, wybierana jest jako jego reprezentant.

Implementacja

Implementacja powyższego modelu znajduję się obecnie w fazie testów. Założeniem przyjętym podczas implementacji systemu wytwarzania sygnatur jest oparcie się o powszechnie uznane oprogramowanie open-source. Za oprogramowanie bazowe uznano *snort* [10] oraz *Apache*. Odciski Rabina zaimplementowano jako wtyczkę (plugin) snorta, bazującego na pluginie *flow* oraz *stream4*. Plugin odpowiedzialny za odciski Rabina jest z kolei podstawą plugina odpowiedzialnego za klasyfikowanie przepływów. Kiedy reguły klasyfikatora przepływów wykryją zagrożenie, przepływy związane z zagrożeniem są przekazywane do modułu Apache, wraz z listą analiz, którym powinny być poddane. Komunikacja pomiędzy snort a Apache odbywa się za pośrednictwem protokołu opartego o mechanizm gniazd TCP, co umożliwia rozdzielenie zadań związanych z klasyfikacją przepływów a wytworzeniem sygnatury pomiędzy oddzielnymi maszynami. Analizy off-line (np. LCS) są zaimplementowane w postaci modułów Apache. Późniejszy proces grupujący sygnatury (dbscan) zaimplementowany jest w języku Perl. Klasyfikacja sygnatur wspierana jest przez definicje sygnatur projektu *bleeding snort* [11].

Podsumowanie

Przedstawione w artykule metody wytwarzania sygnatur zagrożeń sieciowych stanowią zaledwie niewielką część możliwych podejść do zagadnienia. Proponowane metody zostały wybrane pod kątem możliwości implementacji i funkcjonowania w rzeczywistym środowisku sieciowym. Duży nacisk kładziony jest na wydajność poszczególnych rozwiązań. Istotnym problemem nie poruszonym w artykule jest sposób zarządzania wytworzonymi sygnaturami. W przypadku algorytmu LCS nie ma pewności, czy wytworzona sygnatura rzeczywiście jest najlepsza i czy nadaje się ona do zastosowania w systemie IDS/IPS działającym w sieci produkcyjnej. W związku z tym trwają pracę nad zastosowaniem innych algorytmów w ramach opisanego modelu. Niezależnie od jakości sygnatur zaprezentowane metody pozwalają na znacznie lepsze zrozumienie zdarzeń w sieci i służyć mogą jako element systemu wczesnego ostrzegania o nowych zagrożeniach.

Literatura

[1] University of Michigan Internet Motion Sensor Project http://ims.eecs.umich.edu/

[2] L. Banachowski, K. Diks, W.Rytter, “Algorytmy i struktury danych”, Wydawnictwo Naukowo-Techniczne, Warszawa 1996

[3] Sumeet Singh, Cristian Estan, George Varghese, Stefan Savage, “Automated Worm Fingerprinting”. In *Proceedings of the 6th ACM/USENIX Syposium on Operating System Design and Implementation (OSDI)*, 2004

[4] P. Akritidis, K. Agnagnostakis, E.P.Markatos, “Efficient Content-Based Detection of Zero-Day Worms”. In *Proceedings of the IEEE International Conference on Communications (ICC)*, May 2005.

[5] Christian Kreibich, Jon Crowcroft, “Honeycomb - Creating Intrusion Detection Signatures Using Honeypots”. In *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*. Cambridge Massachusetss: ACM SIGCOMM, 2003. http://www.cl.cam.ac.uk/~cpk25/honeycomb/

[6] James Newsome, Brad Karp, and Dawn Song, “Polygraph - Automatically Generating Signatures for Polymorphic Worms”, In *IEEE Security and Privacy Symposium*, May 2005

[7] Colleen Shannon, David Moore, “The Spread of the Witty Worm”, CAIDA, 2004, http://www.caida.org/analysis/security/witty/

[8] Luis Gravano, Panagiotis G, Ipeirotis H. V, Jagadish, Nick Koudas S. Muthukrishnan, Lauri Pietarinen and Divesh Srivastava, "Using q-grams in a DBMS for Approximate String Processing", In *IEEE Data Engineering Bulletin*, 2001

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", *Proc. 2nd int. Conf. on Knowledge Discovery and Data Mining (KDD ‘96)*, Portland, Oregon, 1996

[10] Snort Intrusion Detection System, http://www.snort.org

[11] Bleeding Snort The Aggregation Point for Snort Signatures and Research http://www.bleedingsnort.com