автоматический пен-тестинг – мертвый сезон

крис касперски, ака мыщъх, a.k.a. nezumi, a.k.a souriz, a.k.a. elraton, no-email

в последнее время появилось множество утилит для автоматического поиска уязвимостей, не требующих от хакера никакой квалификации и подвластных даже юзерам. просто запусти программу и пожинай камнепад дыр, сыплющийся со всех сторон, словно перезрелые червивые яблоки. однако… если все было так просто, интернет уже давно бы загнулся и медленно умирал в агонии дымящихся руин. автоматизированный поиск имеет свои ограничения, о которых полезно знать, прежде чем хачить, поэтому немного теории не помешает.

введение

Ручной поиск дыр — крайне трудоемкое занятие, требующее обширных знаний в самых различных областях, причем целенаправленный поиск крайне неэффективен и практически все крупные уязвимости были обнаружены случайно. Ну, не совсем случайно, конечно. Тысячи хакеров не отрываются от дизассемблера, но фортуна стоит к ним задом и удача улыбается лишь немногим.

Зачастую даже не самым умным, талантливым, продвинутым… просто более везучим. Здесь все как у золотоискателей, даже хуже. Ареал поиска настолько широк, что совершенно непонятно куда рыть, где искать и самое главное, _что_ искать. Природа дыр довольно разнообразна и за исключением классических случаев (например, ошибки переполнения или синхронизации) четкой классификации нет.

короткий исторический экскурс

Попытки механизировать, тьфу, то есть автоматизировать процесс добычи дыр из недр программного кода предпринимались неоднократно. Уже в самом начале 90х годов количество отрытых уязвимостей исчислялось сотнями, что позволило задействовать математический аппарат статистического анализа. Выяснилось, что многие дыры имеют сходную природу и программисты зачастую совершают одни и те же ошибки, например, выделяют буфер фиксированного размера, куда копируют полученную по сети строку, забыв предварительно проверить ее длину. Такие ошибки могут быть обнаружены чисто механически, путем передачи серверу строк различной длины в надежде, что случится переполнение и произойдет крах системы.

В 1993 году появился первый автоматический сканер уязвимостей, предназначенный для удовлетворения потребностей системных администраторов и ориентированный на мирные цели, однако, хакеры быстро осознали какие перспективы открывает это мощное оружие и прибрали его к своим рукам.

Речь идет о знаменитой утилите S.A.T.A.N. (Security Administrator Tool for Analyzing Networks — Инструмент Администратора для Анализа Безопасности Сетей), созданной Дэном Фармером (Dan Farmer) и Вице Винемом (Wietse Venema). Проект развивался вплоть до 1995 года, а потом был заброшен и переименован в SANTA (Дед Мороз), в попытке откреститься от хакерской составляющей.

Сегодня любой желающий может скачать исходные тексты (смесь Perl'a с Си) с http://www.porcupine.org/satan и вывести Сатану на орбиту, но… реальной опасности она уже не представляет. Так, чисто музейный экспонат.

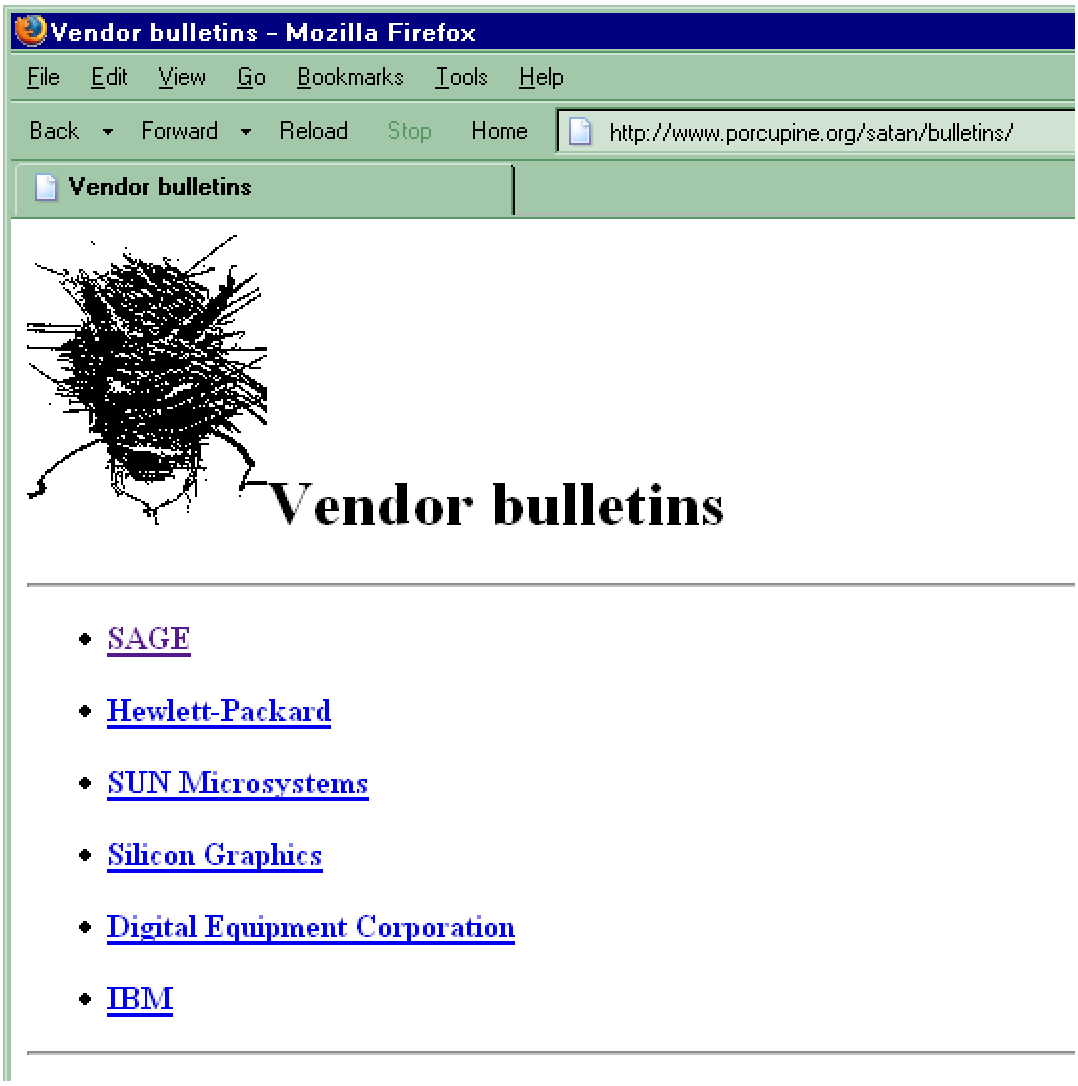

Рисунок 1 отсюда можно скачать исходные тексты легендарного сканера безопасности S.A.T.A.N.

Кстати говоря, к аресту известного хакера Кэвина Митника, Сатана имеет самое непосредственное отношение. Тсутому Шимомура (Tsutomu Shimomura) – эксперт по безопасности — получил заказ от военных ведомств США на разработку улучшенной модели Сатаны, с помощью которой военные надеялись контролировать Интернет, что технически вполне возможно, главное — дыры иметь (достаточно вспомнить Червя Морриса).

Вот в поисках исходных текстов этого монстра Кэвин и вломился к Шимомуре на компьютер, который, обнаружив, что его поимели, в панике поднял военных по тревоге и они прищемили Кевина как молодого. Впрочем, существенно улучшенной модели Сатаны так и не появилось. Автоматизированный поиск дыр оказался недостаточно эффективным — сканер обнаруживал лишь единичные уязвимости. Попытка взять Интернет под контроль угасла как бычок в писсуаре, а Шимомура всю оставшуюся жизнь зарабатывал рассказами на тему как круто он щемил Митника.

Рисунок 2 Тсутому Шимомура собственной персоной

Современные утилиты автоматизированного поиска, конечно же, ушли далеко вперед, но реальной угрозы они не представляют. Так, развлечение для пионеров, атакующих пионерские сайты, создатели которых совершенно не озабочены безопасностью (хинт: что мешает им скачать тот же самый сканер и проверить самих себя на вшивость?!).

автоматизированный поиск дыр без сорцов

В большинстве случаев хакерам приходится атаковать узлы без прикрытия со стороны танков и артиллерии, то есть без исходных текстов, что существенно усложняет задачу автоматического сканирования. Даже если на целевой узел завязан на Open Source проектах, очень трудно (а порой и невозможно) определить какая версия программы там установлена и с какими заплатками. А уж про то, что при перекомпиляции другой версией компилятора или другими ключами часть дыр может исчезать (допустим, использован ключ, форсирующий проверку границ буферов в реальном времени) лучше вообще промолчать.

Короче, мы будем исходить из того, что исходных текстов у нас нет. Что мы можем сделать? Атакуемая программа представляет собой черный ящик со входом и выходом. Устройство черного ящика неизвестно и потому нам приходится искать золото без карты в полной темноте да еще с завязанными глазами и связанными руками. Просто поразительно, что коммерческие (да и некоммерческие) сканеры безопасности ухитряются в таких условиях находить дыры. Как же они, черт возьми, это делают?!

А никак. То есть _вообще_ никак. Движок, дающий основной выхлоп, основан на базе уже известных уязвимостей и потому для эффективной работы требует постоянной дозаправки, в смысле скачки свежих обновлений. Ведь старые дыры постепенно затыкаются и мощь сканера неуклонно падает. Написать подобный сканер много ума не надо и потому можно выбирать любой из десятков готовых (лично мыщъх предпочитает XSpider, но навязывать свои вкусы никому не собирается).

Ленивость системных администраторов (не говоря уже о домашних пользователях) приводит к тому, что количество уязвимых узлов в любой произвольно взятый момент времени исчисляется миллионами! Достаточно заправить сканер свежим топливом, выбрать диапазон IP адресов пошире и… немного подождать. В смысле покурить. А пока мы курим, сканер будет методично перебирать все IP один за другим, проверяя их на предмет установки всех обновлений. Кто не подсуетился, тот уже хрустит у нас на зубах.

Ну и кто же там у нас хрустит? Домашние пользователи — это понятно. Их будет большинство. Они завсегда идут косяками, даже если не сезон (в смысле давно не появлялось свежих дыр). Да только что с домашних пользователей возьмешь? Конфиденциальных документов у них нет, вычислительные мощности слабые, сетевые каналы узкие как медицинская игла… Короче, жопа полная.

Попадется и какое-то количество серверов различных организаций, обычно государственных, где на месте администратора сидит вообще не понятно кто… или что… Тут уже можно развернуться и пошерстить в локальной сети на предмет поиска чего-нибудь интересного. А широта каналов позволит использовать их в самых разных целях от DDoS атак до банальной рассылки спама.

А вот сервера крупных компаний таким методом уже не взломать. Там за безопасностью следят весьма сурово и хотя ленивые (тупые) администраторы все-таки встречаются, эти встречи носят разовый характер. Так что вся надежда на алгоритмы, позволяющие обнаруживать ранее неизвестные дыры, против которых еще не существует вакцины и заплатки еще не написаны.

Самое простое (но не самое умное), что может сделать сканер это искать ошибки переполнения (благо, они самые распространенные) посылая серверу запросы с полям различной длины. Почему различной? Разве не логично бомбардировать сервер строками сумасшедшей длинны, чтобы сразу выловить все ошибки переполнения? Увы… Обработка запросов занимает намного больше одной стадии и данные в процессе продвижения по пищеварительному тракту проходят через множество буферов, часть из которых выполняет проверку, а часть, возможно, и нет. В результате чего, ошибка переполнения проявляется только при обработке строк определенной длины. Более короткие строки полностью помещаются в буфер (и переполнения не происходит), более длинные отлавливаются другими проверками.

Чем больше полей в запросе и чем сложнее структура самого поля, тем больше вариаций мы получаем. Сканер вынужден перебирать огромное количество комбинаций, а это не только трафик, но еще и время. Плюс подозрительно возросшая сетевая активность, регистрируемая системами обнаружения вторжений. Вот чисто для примера, возьмем простейшее поле "FROM" вида "john.smith@domen.com <John Smith>". Как нетрудно догадаться, в процессе синтаксического анализа, сервер разделит одну строку на несколько частей "john.smith", "domen.com" и "<John Smith>" каждая из которых проходит через свою цепочку буферов и потому, чтобы найти ошибку переполнения (даже при условии, что она там есть) необходимо перетрахать все подполя.

Мыщъх реально сталкивался с ситуаций, когда почтовый сервер ограничивал длину поля FROM 1024 байтами, и копировал имя пользователя, заключенное в угловые кавычки в буфер длинной 1000 байт без проверки на переполнение, причем, чтобы добраться до адреса возврата (который и являлся главным объектом атаки) требовалось поместить в буфер еще 12 байт. Плюс 4 байта самого адреса возврата. Итого, на все остальные поля отводится 1024-(1000+12+4) = 8 байт, то есть их длинна должна быть сокращена до минимума, иначе мы просто не влезем в отведенные нам лимиты. Другими словами, чтобы найти строку, вызывающую переполнение, требуется перебрать просто гигантское количество вариантов. Неудивительно, что большинство подобных ошибок годами остаются необнаруженными!

Рисунок 3 атака на переполняющиеся буфера

Более продуктивным оказывается полуслепой поиск, опирающийся на спецификацию конкретного протокола. Сканер просто раскуривает спецификацию, скрупулезно проверяя каждый пункт RFC на соответствие реализации. Возвращаясь к нашему примеру с полем "FROM". Поставим себя на место программиста, которому поручили распарсить строку, то есть расчленить ее на составляющие. Встретив открывающуюся угловую скобку, программист, со всей очевидностью, будет искать закрывающую. А что, если там ее не окажется?! Если программист лось (а большинство программистов в рогами — ну посудите сами, что делать их бедным женам, пока их мужья зависают над монитором?! Правильно — пока программист имеет секс с клавой, его жена имеет секст с нормальным мужчиной), так вот, если программист лось, то он забудет обработать такую ситуацию и будет шпарить по адресному пространству до самого конца, пока не вылетит за пределы выделенной памяти и операционная система, охренев от такой наглости, просто замочит сервер в унитазе. В смысле завершит его выполнение в принудительном режиме.

Количество перебираемых комбинаций при "полуслепом" тесте по-прежнему велико, но все-таки сокращается на несколько порядков и (в зависимости от навороченности протокола) колеблется от сотен тысяч до десятков миллионов запросов. Допустим, обработка каждого запроса занимает 0,01 сек, тогда на сканирование уйдет от 10 минут до суток — вполне приемлемая величина! Единственная проблема в том, что трудоемкость разработки полуслепого сканера чрезвычайно велика. Вместо тупого перебора строк разной длинны в цикле, необходимо тщательно проштудировать горы RFC и для каждого пункта спецификации написать отдельную процедуру тестирования — а это километры строк на Си.

До сих пор не существует ни одного полуслепого сканера, проверяющего _все_ пункты спецификации даже самых простейших протоколов (таких как, например, POP3 или SMTP). Реально проверяется лишь наиболее "перспективные" пункты спецификации, да и то не в полной мере. Тем не менее, осознавая перспективность полуслепых сканеров, разработчики постоянно добавляют все новые и новые методы сканирования и потому разница между старой и новой версией полуслепого сканера практически всегда оказывается весьма существенной, чего нельзя сказать о слепых сканерах, которые от версии к версии только обрастают жиром и побочным функционалом, типа генерации отчетов.

Стоит так же упомянуть эвристические алгоритмы. Как они работают скорее всего не понимают даже их авторы… В общем случае, они вообще никак не работают и не дают никакого выхлопа, но лэйбл "эвристический сканер безопасности" звучит весьма стильно и активно используется для продвижения продукта на рынок. Если принцип эвристического алгоритма не описан в документации, то с вероятностью близкой к единице можно утверждать, что это просто лажа, особенно если разработчики ссылаются за "закрытые запатентованные технологии", забыв о том, что закрытых патентов не существует и не может существовать по определению и текст любого патента можно бесплатно скачать из Интернета на совершенно легальной основе.

Еще существуют (и активно используется) стресс тесты, демонстрирующие реакцию сервера на пиковую загрузку. К безопасности они имеют весьма косвенное отношение, ну разве что позволяют устраивают DoS атаки, которыми развлекаются пионеры, но… В том-то и дело, что завалить рядовой сервер небольшой компании можно и без всякого теста. Как правило для этого достаточно запустить программу, создающую зеркало сайта и качающую сразу в несколько потоков (например, в десять или сто). Забавно, но многие домашние пользователи уже имеют более широкие каналы, чем различные организации и если администратор не ограничит скорость отдачи для каждого соединения (с учетом максимально разрешенного количества соединений), то DoS можно организовать и легальными методами с помощью обычного браузера, заняв 100% пропускной способности канала и оттеснив других посетителей сайта в глухую очередь в которой они зависнут надолго.

Но это ладно. Серьезные сервера таким способом не обломать, особенно, если администратор заранее позаботился о балансировании нагрузки и воздвиг распределенную систему, подключенную к нескольким сетевым каналам с шириною в целую магистраль. Вот тут стресс тесты и выручают. Самые примитивные из них просто забрасывают сервер ворохом бессмысленных запросов, надеясь, что от этого ему сильно поплохеет, однако, вслепую подобрать "правильные" запросы крайне маловероятно. А вот если заранее спланировать атаку… Возьмем, например, Java-апплеты. Известно, что перед запуском они в обязательном порядке проходят через верификатор, проверяющих все инструкции виртуальной машины одну за другой. Причем, если очередная проверяемая инструкция воздействует на предшествующую, то верификатор выполняет повторный цикл проверки с учетом "вновь открывшихся обстоятельств". При желании можно написать такой Java-апплет из N команд, верификация которого потребует N^N итераций, то есть "подвесит" сервер на очень долгий срок. Причем, это не баг, а фича. Обойти ее возможно только квотированием процессорного времени, но не каждый сервер это позволяет, а если даже позволяет, его еще требуется настроить.

Другой пример целенаправленного стресс теста: упаковываем несколько гигабайт нулей по gzip алгоритму (который поддерживают практически все web-сервера) и смотрим что из этого получится. С некоторой вероятностью сервер либо впадает в задумчивость, граничащую с нирваной, либо рушится.

Целенаправленные стресс тесты действительно весьма эффективны, однако, толку с них… DoS атаки это чистейшей воды вандализм, причем преследуемый и наказуемый. И самое главное, если владелец сервера не полный лох, то он просто заблокирует IP злоумышленника или даже всю подсеть его провайдера. И хотя останется возможность работы через proxy (вот только далеко не все утилиты для стресс тестирования поддерживают proxy), блокируются и они. Чтобы завалить сайт конкурентов на длительное время, требуется доступ ко множеству Интернет-каналов, что требует нехилых вложений. Дешевле банду гопников послать.

автоматизированный поиск дыр с сорцами

Наличие исходных текстов существенно упрощает поиск дыр, поскольку из черного ящика программа превращается в белый. В большинство современных компиляторов уже встроены более или менее серьезные верификаторы, обнаруживающие значительную часть ошибок. Достаточно задействовать максимальный уровень предупреждений, задаваемый ключом командой строки, описанным в документации на компилятор. По умолчанию компилятор ругается только на откровенную лажу, за которую расстреливать надо, но… большинство программ (в том числе и коммерческих) вызывают ругательства компилятора даже на самом "мягком" уровне предупреждений. А на максимальном уровне половина компиляторов ругается на свои же собственные библиотеки.

В компилятор GCC встроен достаточно мощный верификатор, что позволяет использовать его в качестве бесплатного сканера безопасности. К сожалению, проверка программ, написанных на Microsoft Visual C++ существенно затруднена. Будет чудо, если GCC их вообще откомпилирует — уж слишком много в них системно-зависимого кода и нестандартных языковых расширений, которые GCC в упор не переваривает. Но мы же ведь не компилировать собрались, а сканировать, верно? Вот и выкидываем весь "левый" код на хрен, прогоняя через GCC системно-независимые модули, ответственные за обработку запросов, получаемых сервером из сети.

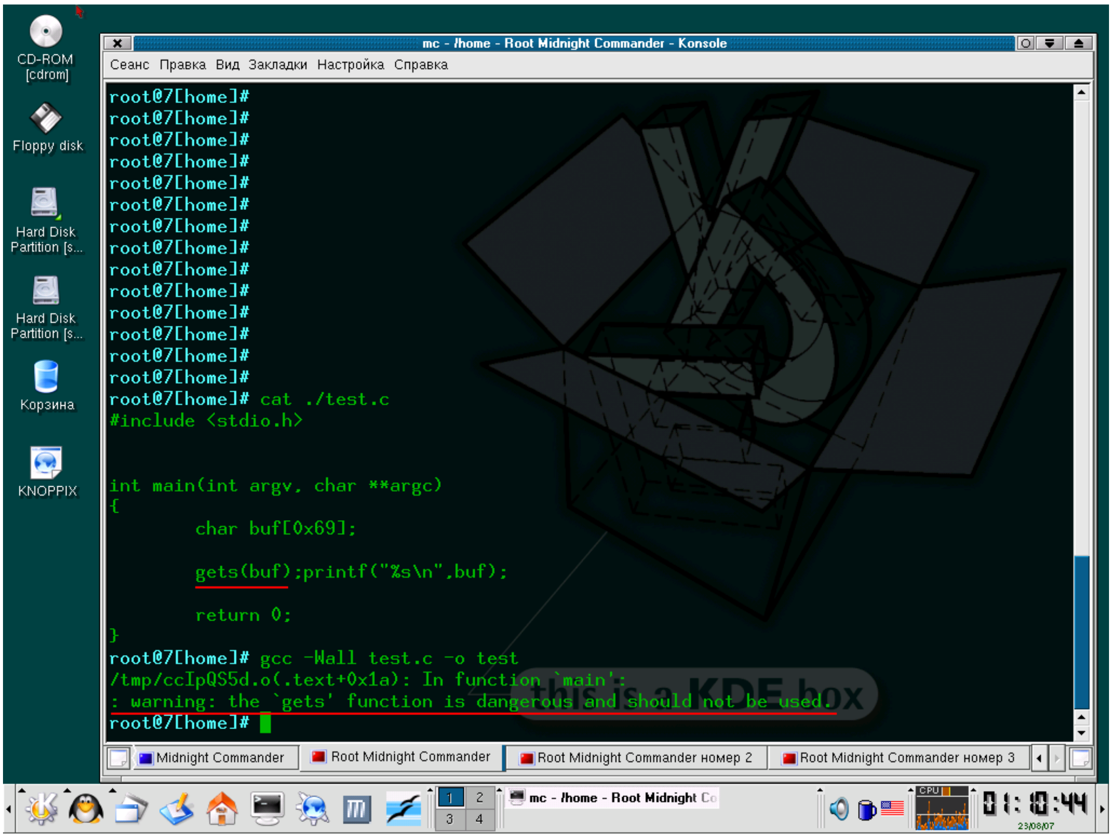

Рисунок 4 пример ругательства компилятора GCC на функцию gets (*внимание: функция 'gets' опасна и не должна использоваться*)

Компилятор легко обнаруживает отсутствие проверок на длину строки перед копированием ее в буфер, а так же выявляет ряд потенциально небезопасных конструкций. Самое главное, что теперь возможно загнать все ветвления на графы и перебирать аргументы именно в тех диапазонах, в которых осуществляется их проверка. В грубом приближении это выглядит так: сканер находит функцию, копирующую строку (или ее фрагмент) в буфер фиксированного размера без предварительной проверки ее длины. Он прослеживает траекторию передачи аргументов: от входных данных до уязвимой функции и смотрит: возможно ли подобрать такую длину строки, которая бы вызывала переполнение в уязвимой функции и не отсекалась бы предшествующими ей проверками.

Подобное сканирование требует совсем немного времени и дает надежный результат. Но еще круче сканирование с обратной связью, при котором проверяются все возможные состояния программы. Движок таких сканеров основан на инструментах, измеряющих покрытие кода (coverage). Задача сканера состоит в том, чтобы покрыть 100% всего кода, проверив его работоспособность. Для каждой строки кода выполняется так называемый "обратный расчет аргументов", то есть сканер анализирует код и пытается найти такие входные данные, обработка которых привела бы функцию в заданное состояние. С точки зрения чистой математики эта задача не имеет решения, однако, если к сканеру прикрутить трассировщик, то мы получим возможность выполнять обратную трассировку, то есть как бы пустить время вспять. На самом деле, это невозможно, поскольку выполнение программы представляет собой однонаправленный процесс, связанный с необратимой потерей данный. В частности, инструкция XOR EAX,EAX обнуляет регистр EAX и мы не можем установить его оригинальное содержимое. Поэтому, обратная трассировка дает не один вариант выполнение, а целое множество. Грубо говоря, образуется система уравнений, которую сканер должен решить. Вот пускай и решает!

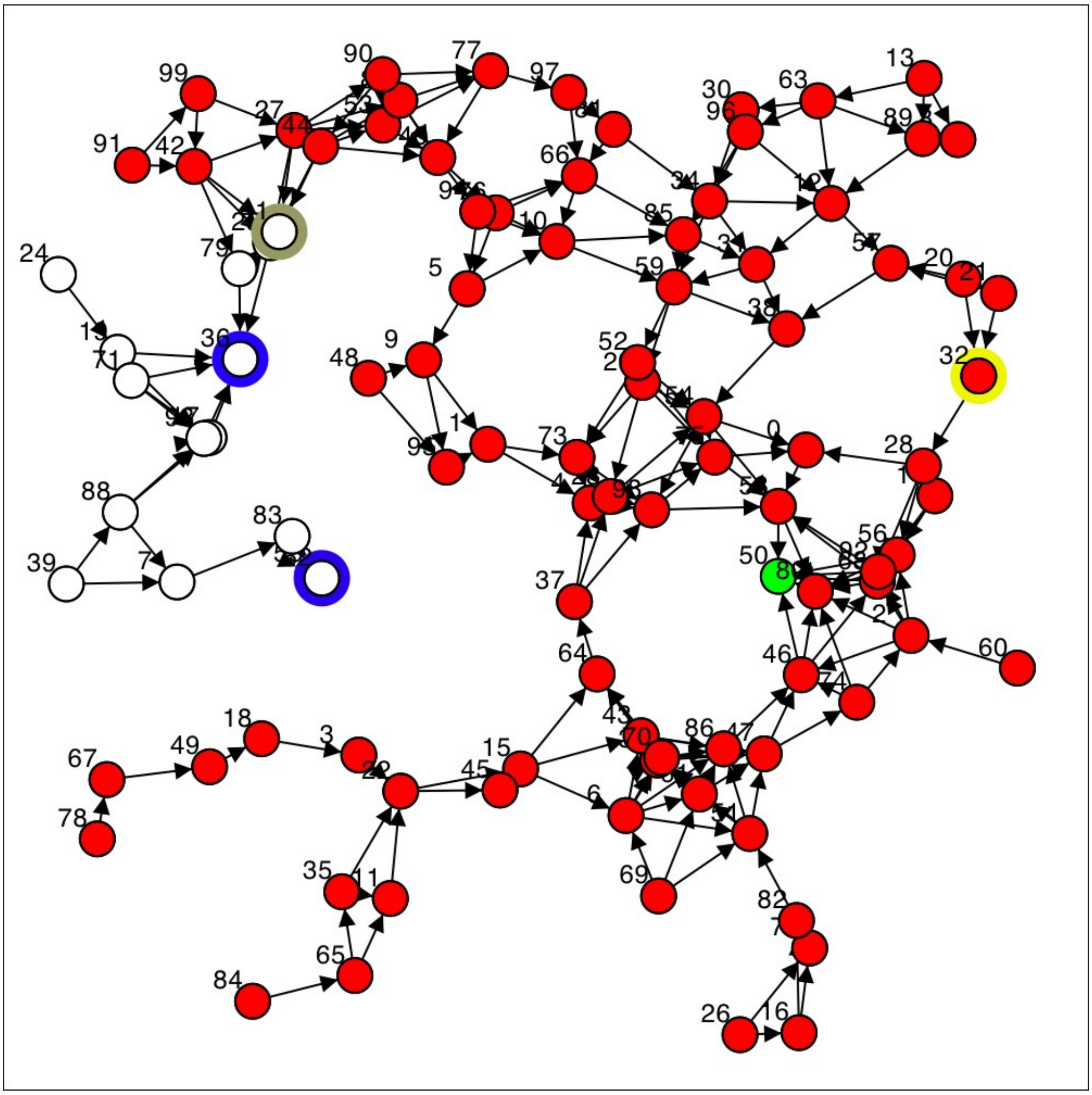

Рисунок 5 примерно так выглядит код, загнанный на граф

Сканеры с обратной связью появились относительно недавно и в настоящее время существуют в виде лабораторных образцов, не доведенных до рыночных продуктов. Тем не менее, с их помощью уже найдено множество дыр (впрочем, большей частью некритических) и проведен аудит кода популярной операционной системы OpenBSD.

заключение

Если все так хорошо, то почему все так плохо? Почему подавляющее большинство юзабельных дыр (то есть дыр, пригодных для атаки) обнаруживаются именно вручную, а сканеры (даже самые лучше) в основном выявляют мелкие баги? Ответ прост и тривиален: серьезные продукты перед сдачей в эксплуатацию прогоняются через новейшие сканеры безопасности и потому хакер имеет шанс обнаружить дыру только при тестировании древнего продукта новой версией сканера.

Только шансы эти будут весьма невелики, особенно в случае "черного ящика" все состояния которого проверить невозможно. Наличие исходных текстов, подкрепленных авторитетом сканеров с обратной связью, конечно, существенно увеличивает шансы на удачу, только все равно… состояний этих много и сканер проверяет лишь некоторые из них, оставляя нетронутым целый пласт всевозможных комбинаций аргументов.

Самое главное — автоматизированные сканеры действуют по заложенной в них шаблонной схеме. Они не способы мыслить, совершать открытия, рождать качественно новые решения. Все это — удел человека. Дыры неизвестно типа сканеры даже не пытаются искать, поскольку, это не входит в сферу их компетенции. А если бы даже пытались, то все равно хрен бы что нашли.

Так что не нужно строить ложных иллюзий и приписывать сканерам безопасности не свойственные им возможности. Но и отказываться от них тоже не нужно.

>>> врезка XSpider

XSpider – популярный сканер безопасности от Positive Technologies (http://www.ptsecurity.ru/xs7.asp), разработчики которого утверждают, что за последний год они ""проигнорировали" 40% вновь опубликованных уязвимостей, поскольку, интеллектуальные алгоритмы сканера определяли их с самого начала, в результате чего, XSpider позиционируется как "единственный в мире сканер, уже сегодня определяющий завтрашний день".

Ну, на счет "единственного" маркетологи, конечно, загнули, если не сказать, погорячились. Сканеров сейчас развелось… К тому же "интеллектуальность" SXpider'а очень сильно преувеличена. Это шаблонный сканер, действующий методом слепого и полуслепого перебора, что, естественно, ограничивает его возможности.

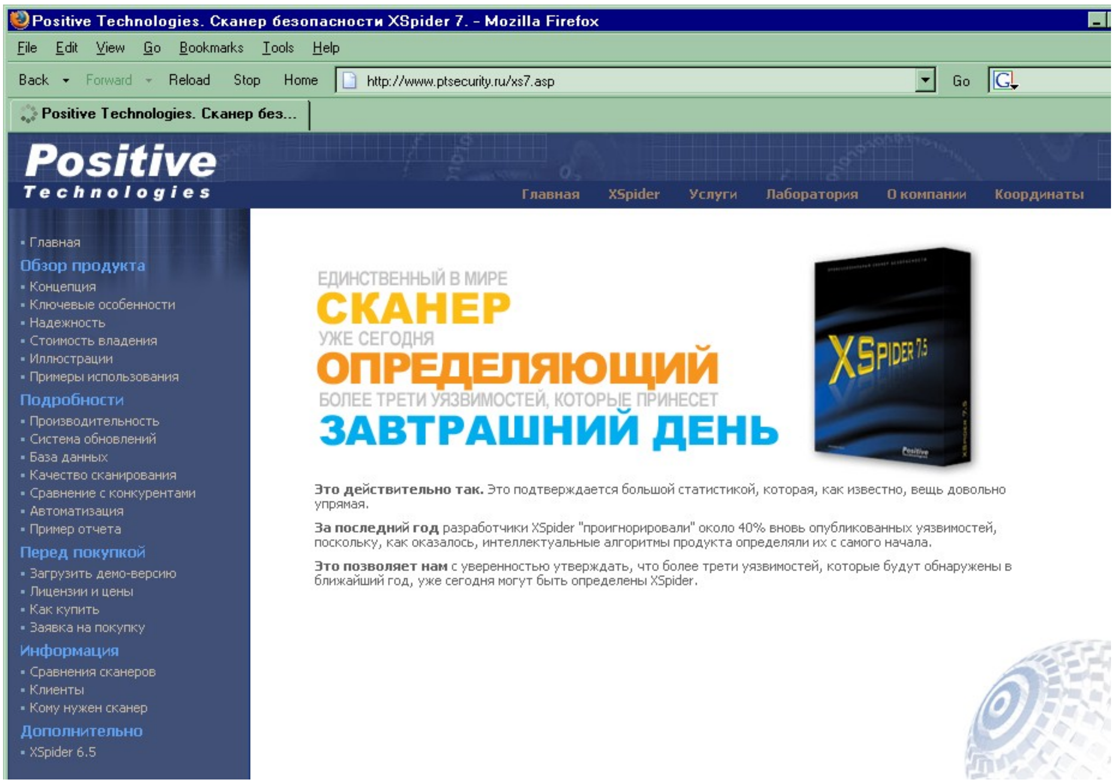

Рисунок 6 XSpider с точки взгляда маркетоидов

Основное его достоинство — обширная и регулярно пополняемая база известных уязвимостей, отслеживающая множество программных продуктов. Весьма полезно просканировать свой компьютер, чтобы убедиться в отсутствии критических дыр. Тоже самое относится и к чужим компьютерам (мы уже говорили, что в настоящее время подавляющее большинство атак совершаются через уже известные дыры).

Это коммерческий продут, распространяющийся на платной основе. Самая дешевая версия, поддерживающая сканирование всего лишь 4х IP-адресов на данный момент стоит 7.400 рублей, но уже для сканирования подсети класса C (256 IP-адресов) придется выложить 59.000 рублей, а B-сеть (65 536 IP-адресов) обойдется аж в 269.800 рублей. Короче, этот сканер не для простых людей. Если бы у меня было 269.800 рублей я бы… Ладно, не будем расстраиваться, благо существует бесплатная демонстрационная версия, отличающая от коммерческой следующими ограничениями:

- Отсутствует система ежедневного онлайнового обновления;
- Отсутствуют потенциально опасные проверки на DoS-уязвимости;
- Проверки содержимого веб серверов на предмет SQL инъекций, инъекций кода, получения файлов и т.д. не содержат детали;
- Отсутствует целый ряд проверок, использующих оригинальные эвристические механизмы;
- Отсутствуют проверки, связанные с использованием различных словарей;
- Планировщик заданий имеет полноценный интефейс, но не сохраняет созданные расписания;
- Генерируемые отчеты содержат только резюме по реальному сканированию, в теле отчета - стандартный демонстрационный фрагмент;
- История сканирований доступна только для общего просмотра, старые результаты нельзя использовать для последующей работы;

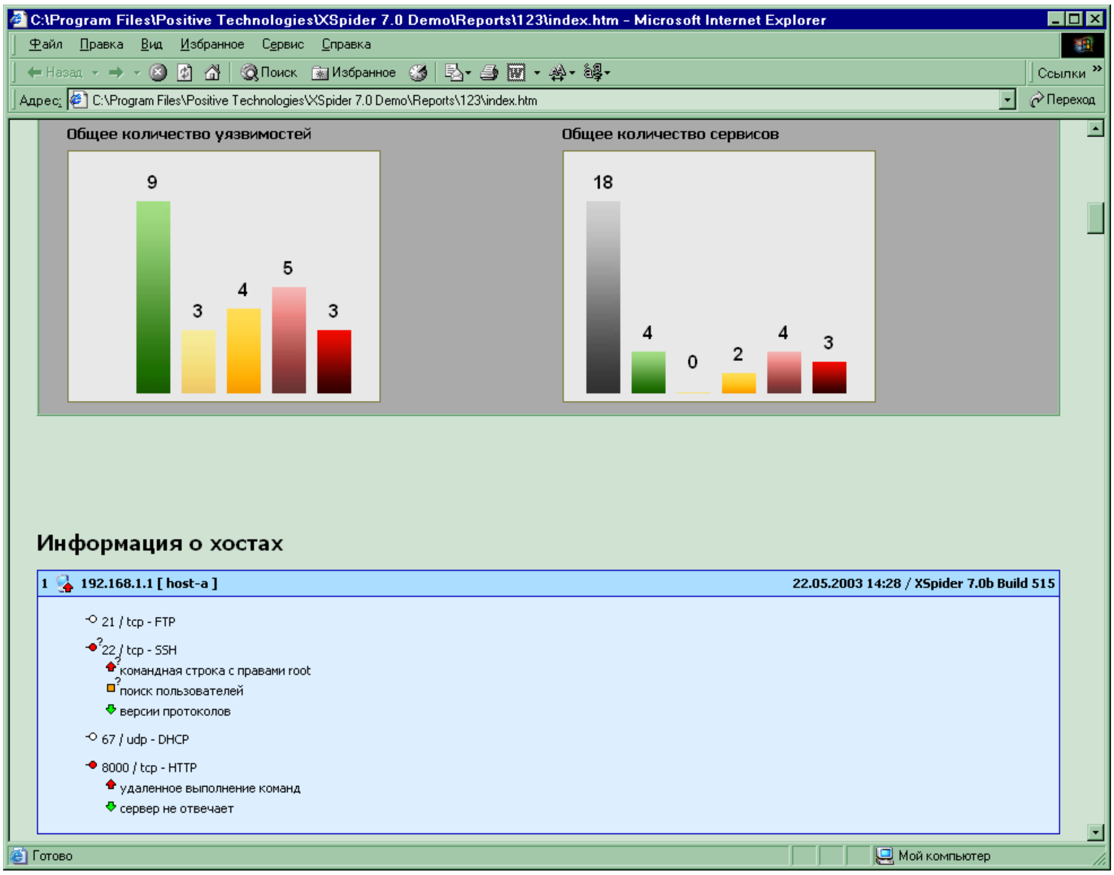

Рисунок 7 пример отчета, сгенерированного программой XSpider