Позади закрытых дверей: Port knocking. Часть 1

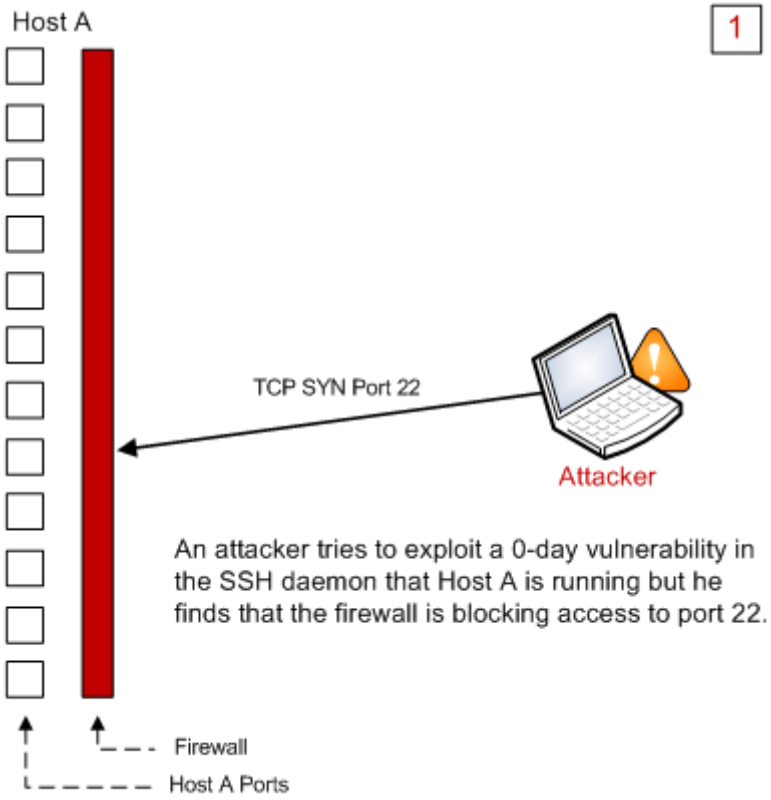
Техника Port knocking уже достаточно известна и распространена, с каждым годом обрастая всё новыми модификациями и сферами применения. Сегодня мы рассмотрим основы её устройства, а также многочисленные реализации

Любые стандартные защитные средства от неправомерного доступа, будучи всегда на виду, хорошо изучены и всем известны, неизбежно и регулярно будут испытываться на прочность и совершенство конкретной реализации. Поэтому в вопросах безопасности особенно важных и критичных сетевых ресурсов, в последнее время ставка делается на различные нестандартные и скрытые формы аутентификации или получения целевого доступа. Для маскировки такого рода механизмов эффективней всего выбирать максимально рядовые и характерные события, типичные для сетевой среды, которые неявно могут нести функцию сообщения/аутентификации, при этом, никак не привлекая к себе внимания потенциальных злоумышленников.

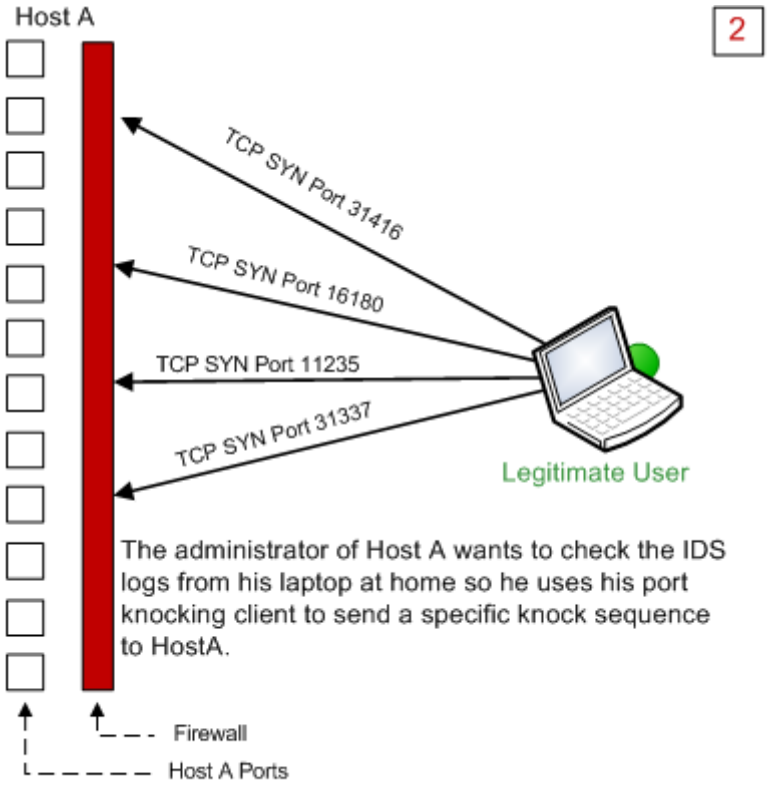
Переходя к делам сетевым, в качестве такого секретного триггера или сигнала уместно выбрать сам TCP/IP поток, выделяя какую-то его специфику и используя её для своих скрытых нужд. В этом и заключается суть метода PK, который имеет огромное количество реализаций и применений. Но перед началом нашего практического экскурса попробуем сформулировать и дать определение технике «*секретного простукивания*» портов.

Port knocking — это неявная форма разрешения доступа к некоему сервису, при условии прохождения предварительно заданной последовательности соединений с различными портами целевого сервера.

Специальное ПО на стороне сервера отслеживает все входящие соединения, и если фиксируется характерная «цепочка подключений» соответствующих ранее заданному «*эталонному стуку*» — временно открывает доступ к закрытому порту (и, соответственно, скрытому сервису на нем).

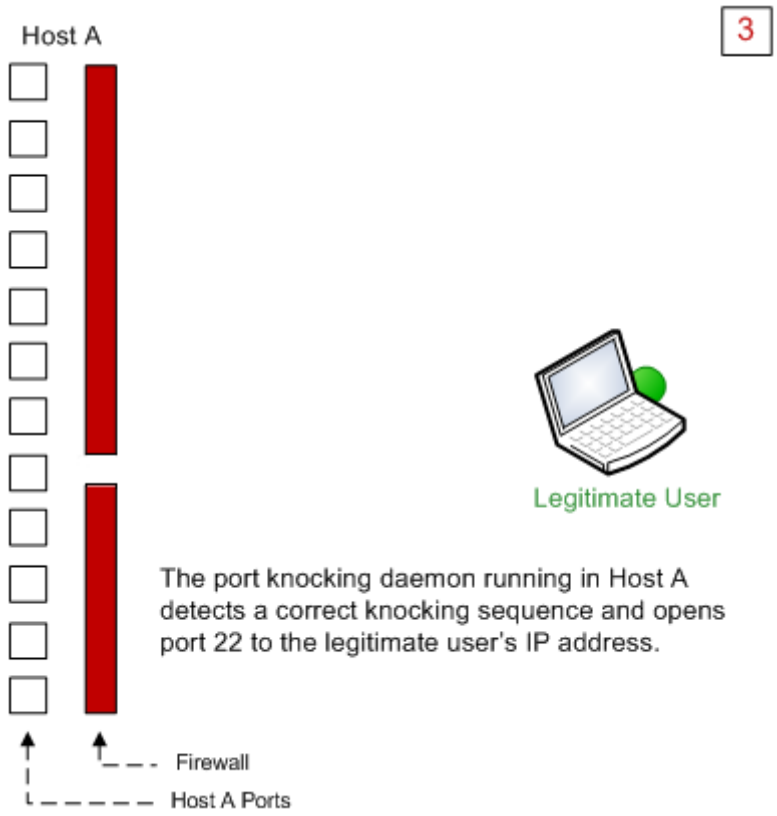


В более общем случае, технология PK позволяет удаленно и скрытно выполнять команды на защищенном сервере, даже если все порты на нем постоянно закрыты.

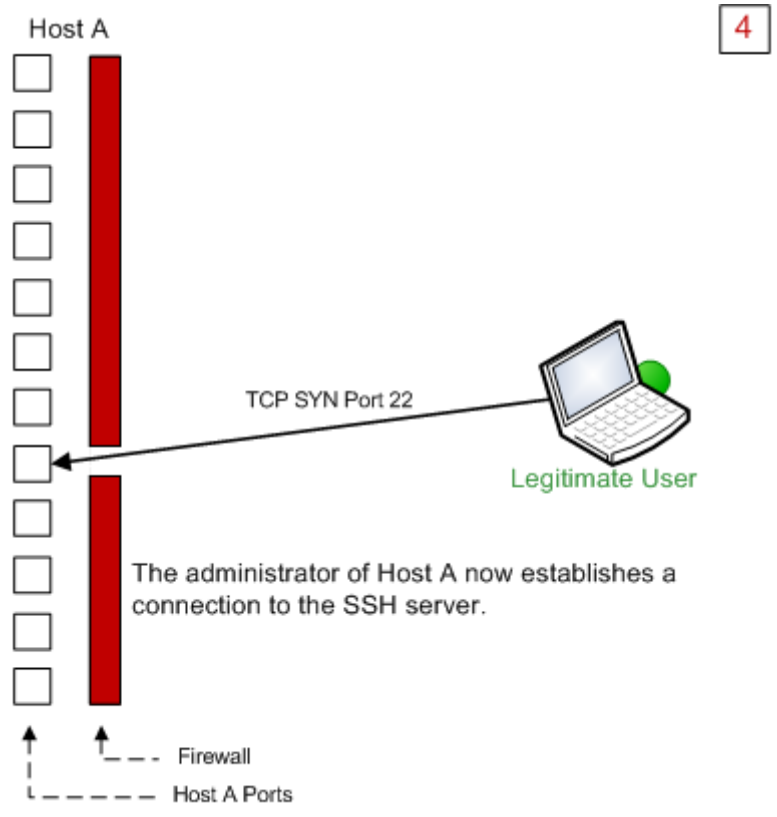


Что касается разнообразия подходов по реализации PK, то условно их все можно выделить в две группы. С одной стороны, это простейшие «самопальные скрипты» следящие за лог-файлом брандмауэра (или, например СУБД MySQL), которые обнаружив там заданные закономерности «стуков извне» — добавляют в него новые правила (или исполняющие условленные системные команды).

Похожий подход иногда реализуется через сочетание работы стандартных tcpdump и sendip, существуют также и другие вариации на вечную сисадминскую тему «сделано на коленке».



Эта статья посвящена подробному рассмотрению доступного арсенала из второй крупной группы в зоопарке решений для РК —*специализированных приложений*, которые гораздо более универсальны и часто технически безупречны.



Стучащему, да откроют

Так уж повелось, что *Port knocking* чаще всего применяют для скрытия сервисов удаленного доступа на компьютер, например для SSH. Чтобы защитится от «плохих парней» некоторые перевешивают SSH на другой, нестандартный порт. Многие сидя за наглухо закрытым брандмауэром «*проковыривают в нем дырочки*» для доступа только из тех подсетей, откуда работают сами. В первом случае это решение слабо защищает от современных атак, в последнем — лишает мобильности.

Я же предлагаю ниже рассмотреть *port knocking*, как наиболее гибкий и универсальный вариант для скрытия подобных сервисов от посторонних. И для первой иллюстрации работы «*кнокеров*» я выбрал классическую реализации такого рода инструментов — knockd.

Настройка `knockd` чрезвычайно проста, и эта одна из причин его популярности. Вся система делится на серверную часть, которая будет слушать поступающие сигналы «из центра», и поставляемую в комплекте клиентскую часть, которая позволит «*сыграть на серверных портах*» нужную «*мелодию*» из сетевых пакетов в момент, когда вам понадобится закрытый со стороны сервера доступ.

Итак, собираем и устанавливаем knockd (я описываю вариант для FreeBSD, но также доступны пакеты и для разных версий Linux).

```
$ cd /usr/ports/security/knock
$ make install clean
```

Перед настройкой демона `knockd` поясню — он выполняет любые ранее заданные команды на стороне сервера в случае обнаружения управляющей последовательности. Какая же функциональность будет вложена в эти заданные триггеры — дело исключительно вашей необходимости и фантазии. Для демонстрации я выбрал защиту доступа к SSH.

Приведу пример настройки главного конфигурационного файла `knockd.conf` :

```
# Глобальная секция настроек
[options]
# Путь к лог файлу
logfile = /var/log/knockd.log
# Какой сетевой интерфейс будет прослушивать knockd?
interface = eth1
[ssh_open]
sequence = 1291:udp, 3614:udp, 1492:tcp, 1090:udp
seq_timeout = 20
tcpflags = syn,ack,!urg
start_command = /usr/sbin/iptables -A INPUT -s %IP% -p tcp –syn –dport 22 -j ACCEPT
[ssh_close]
sequence = 1217:udp, 3114:udp, 1099:udp, 1292:tcp
seq_timeout = 20
tcpflags = syn,ack,urg
command = /usr/sbin/iptables -D INPUT -s %IP% –dport 22 -j ACCEPT
```

Здесь в общем блоке `[options]` задаем общие значения, и далее перечисляем блоки-условия, параметры в которых будет контролировать демон (у нас таких только два, это `ssh_open` и `ssh_close`). В них через ключевое слово sequence мы последовательно указываем порты и протоколы, которые будут прослушиваться, а в `seq_timeout` — максимальное время в секундах, отведенное нами на проигрыш полной «цепочки из портов».

Здесь протоколы можно не указывать явно, тогда по умолчанию будет подразумеваться TCP. Следующий параметр `tcpflags` указывает TCP-пакеты с какими взведенными флагами будут учитываться в качестве TCP-стуков (если таковой протокол вообще используется в `sequence`). Кроме простого перечисления возможно также и явное исключение некоторых флагов в качестве «сигнальных» (через префикс «!»).

В случае выполнения всех условий описанных в блоке запускается его целевая команда (оператор `command`), в нашем случае — это открытие и закрытие порта для брандмауэра. Но ручное закрытие соединения вовсе не обязательно, предусмотрено несколько вариантов для его автозакрытия. Приведем подобный пример с аналогичной функциональностью, где количество выполняемых команд в блоке `ssh_open` расширено с 1 до 3 (а `ssh_close` отпадает за ненужностью):

```
start_command = /usr/sbin/iptables -A INPUT -s %IP% -p tcp –dport 22 –syn -j ACCEPT
cmd_timeout = 20
stop_command = /usr/sbin/iptables -D INPUT -s %IP% -p tcp –dport 22 –syn -j ACCEPT
```

Это значит, что сначала открыв порт для доступа, мы ждем в течение 20 секунд входящих на него подключений, после чего закрываем его. Мы исходим из того, что пакетный фильтр сохраняет уже установленные соединения, поэтому блокировка порта не прервет уже запущенную SSH-сессию, — теперь нам не нужно помнить о необходимости «*уходя закрывать за собой двери*».

Внимательный читатель, вероятно, заметил, что в первом примере для команды использован единственный оператор `command` , тогда как во втором я использую тернарную конструкцию `start_command` и `stop_command` , с разделяющей их `cmd_timeout` (задающей временной шаг срабатывания между ними). Смысл всех этих трех разновидностей `command` эквивалентен — запуск внешней команды.

В заключение этого пункта отмечу, что я не заостряю здесь внимание на деталях взаимодействия с брандмауэром, так как читатель волен использовать любые другие удобные лично ему пакетные фильтры, тем более что эта функциональность тривиальна.

Теперь обратим внимание на слабые места этого механизма: при возможности прослушки трафика, любой злоумышленник вооруженный простейшим `tcpdump` сможет с большой вероятностью обнаружить закономерности подобных повторяющихся действий (пакетов), после которых инициируется успешное соединение с сервером. После этого он может самостоятельно повторить необходимые «стуки», т.е. провести так называемую replay-атаку (sequence replay attack).

Для того чтобы свести опасность этого к минимуму, у `knockd` имеется вариант работы с одноразовыми секретными последовательностями, для демонстрации чего приведем третий вариант настройки блока:

```
[openFTP]
one_time_sequences = /etc/knockd/ftp_sequences.txt
seq_timeout = 20
tcpflags = fin,!ack
command = /usr/sbin/iptables -A input -s %IP% -p tcp –dport 25 -j ACCEPT
```

Единственное новое, что требует пояснение в этом примере — параметр `one_time_sequences` , который указывает на файл-список всех доступных последовательностей, который должен быть подготовлен заранее. После каждой успешно сработавшей последовательности она аннулируется (будет закомментирована символом ’#’ вначале строки), после чего сервер начинает ожидать поступления следующей ключевой последовательности по списку.

Естественно, у сервера и клиента должны быть полностью аналогичные списки, проведение replay-атаки в таком случае затруднительно.

Вот «кусочек» такого списка для наглядности:

```
#3237:udp, 3334:udp, 1569:tcp, 1652:tcp
#1945:udp, 4574, 2849:udp, 1952:udp
1367:tcp, 3176, 1678, 4893
1667:tcp, 3566:udp, 1596:tcp, 1589:tcp
1677, 3774, 1789:udp, 2697
5516:tcp, 5335:udp, 2044:tcp, 3297:tcp
```

Сезам, откройся!

Рассмотрев «дела серверные» самое время переключить наше внимание на сторону клиента и ответить на насущный вопрос: как же выстукивать все эти «сетевые трели» простому смертному?

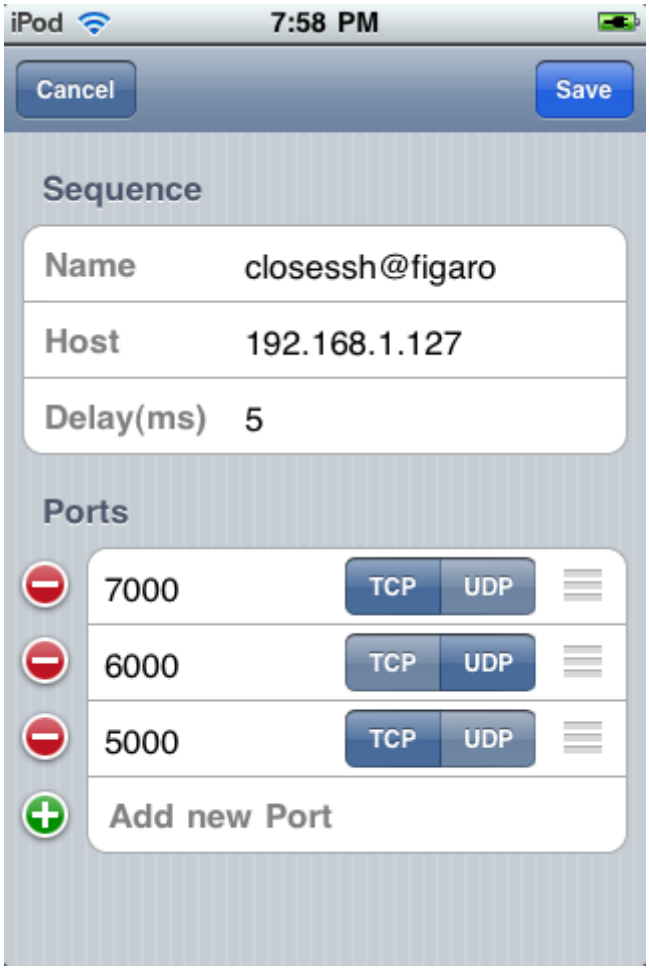
Во-первых, можно использовать штатный клиент поставляемый вместе с самим сервером, вот типичный пример запуска SSH (параметры для подключения аналогичны формату используемому в серверном `knockd.conf` и в режиме подключения по списку):

```
savgor$ knock 1291:udp, 3614:udp, 1492:tcp, 1090:udp & ssh sesame.samag.ru
```

Но, при большой мобильности не всегда удобно иметь дело именно с unix'овым клиентом, и тут на выручку приходят сторонние клиенты, полностью совместимые с оригинальным `knockd` . Прежде чем перечислять их следует отметить, что в качестве универсального решения можно приспособить обычные `telnet` или `netcat` , разве что при этом на сервере следует увеличить тайм-аут ожидания и использовать более простые комбинации стуков.

Для более сложных вариантов удобно использовать сетевые утилиты hping или packit, которые на данный момент доступны на платформах Linux, *BSD, Solaris, MacOs X и Windows. Если же вы применяете замысловатые и длинные комбинации для выстукивания нужных мелодий на удаленном сервере, то для Windows лучше использовать консольные утилиты knock-win32 и It's me . Последний клиент не только поддерживает шифрование и виртуоз в области наиболее сложных и длинных цепочек, но и регулярно обновляется.

Для обладателей мобильных устройств на базе iOS я рекомендую KnockOnD или Port Knock Lite for iPhone (увидеть первый из них можно на рисунке ниже). Кроме того можно использовать Knock-Android для одноименной платформы, ну и в заключение стоит привести пример веб-клиента для `knockd` , который написан на PHP — это phpKnockClient.



Knockd-клиент для iPhone/iPod – KnockOnD

Вот его запуск из командной строки:

```
$ knock.php sesame.samag.ru 1291:udp, 3614:udp, 1492:tcp, 1090:udp
```

Обзор клиентов закончу классическим nmap, который при правильно заданных параметрах может «*выстукивать*» сколь угодные сложные чечетки на удаленном брандмауэре, вот готовый шаблон из ключей для его использования в этих целях:

```
$ nmap -sS -T Polite -p<port1>,<port2>,<portN>... <target>
```

Здесь мы указываем правильный порядок из портов (`port1, port2` и так далее), где `target` — это сетевой адрес целевого сервера. При этом важно указать ключ `Polite` , который гарантирует строго указанный порядок отправки пакетов, более подробное объяснение опций можно посмотреть в документации к nmap.

Позади закрытых дверей: Port knocking. Часть 2

Это продолжение первой статьи про **port knocking** (PK), — это её вторая часть. В первой (обзорной) части мы познакомились с основами и рассмотрели простейшие методы PK, многие из которых (именно в силу своей простоты) содержат различные недостатки.

Поэтому сегодня мы продолжим наше погружение в эту тему и обсудим более поздние и совершенные реализации PK, в том числе *гибридные техники* и алгоритмы с использованием *единственного крипто-пакета* (SPA), после чего подведем черту: сравним и проанализируем совокупные преимущества и недостатки каждого из рассмотренных подходов, чтобы в результате каждый мог выбрать наиболее подходящую ему реализацию «*кнокера*».

Cerberus: OTP в действии

Кроме рассмотренного ранее простого `knockd` имеются и более сложные методики организации portknocking'a, некоторые из которых я хотел бы рассмотреть ниже хотя бы обзорно.

Cerberus, вероятно, одна из первых вариаций на тему PK, но вовсе не самая простая из существующих. Очень долгое время Cerberus использовался для служебных нужд в некоторых правительственных учреждениях США, и только в 2005 году стал доступен для широкой общественности. Это продвинутая «*стучалка*», которая сочетает сразу два подхода: метод одноразовой аутентификации; и классическая аутентификация каждого конкретного пользователя в неявной форме.

Давайте посмотрим, как это реализуется на практике.

Cerberus работает только через протокол ICMP, при этом используя в качестве транспорта наиболее стандартную его форму (тип 12). Как следствие этого, Cerberus позволяет работать через брандмауэр (-ы) в подавляющем большинстве случаев (как с ведома их администраторов, так и без) . Кроме того, системы обнаружения атак спокойно реагируют на пакеты Cerberus, в отличие от `knockd` .

ICMP: А если его тупо запретить?
Конечно, теоретически ICMP можно заблокировать, но этот протокол прямым образом влияет на маршрутизацию и фрагментацию TCP/IP-пакетов (например, именно через ICMP идут посылки достаточно важных запросов «frag needed»), поэтому это является грубым нарушением сетевых стандартов и выполняется только в крайнем случае (например, при неких ICMP-уязвимостях в системах). Блокировка сообщений типа 3 и 4 с большой вероятностью нарушит нормальную фрагментацию пакетов, что внешне проявляется как внезапная остановка передачи данных большого объема, а также периодические загадочные ошибки и разрывы сессий по таймауту. По поводу первобытного страха ICMP-флудинга и прочего — чаще всего это страшилки из прошлого века

(подобные вещи невозможны при правильно настроенной сети), сейчас практически все сохо-маршрутизаторы успешно распознают и блокируют атаки подобного рода, да и DDoS-атаки в основном направлены на web-службы.

Теперь о деталях внутреннего устройства, а значит и о возможностях этого механизма. Пакет Cerberus состоит из заголовка и динамически-вычисляемого одноразового пароля (One time Password, OTP), блок с которым занимает последние 16 символов. Функционально-главное поле заголовка (`ActionID`) содержит код команды, которую следует запустить на стороне сервера.

Вот общая структура описанного пакета:

```
struct {
        2 byte Initiator (0xDEAD)
        1 byte UserID
        1 byte ActionID (Action sequence)
        8 byte One time Password (OTP)
        4 byte IP address (Dotted decimal to Hex)
}
```

В таком пакете *одноразовым паролем* (OTP) выступает MD5-хеш из точного системного времени, ранее полученное случайное число с сервера (server seed), пользовательского пароля, а также целевого IP-адреса.

Структура пакета Cerberus

Проиллюстрируем эту теорию на практическом примере клиента, который отправляет одноразовый пакет с помощью подручных средств. Я привожу команды для FreeBSD, с минимальной адаптацией они доступны и на других платформах.

1. Сначала создаем крипто-составляющую пакета — одноразовый ключ (OTP):

```
date +%d%m%y%k%MSrvseedMypass204.244.123.234 | md5sum | cut -c <nobr>17-32</nobr>
```

2. Получаем результат «f0b70bc031a365e9ccf47bea», который вставляем в общий формат сообщения и отправляем его на сервер:

```
ping -c1 –p «dead4201f0b70bc031a365e9ccf47bea» sesame.samag.ru
```

Здесь переменные `Srvseed` и `Mypass` — это серверное seed-значение и пароль акаунта пользователя на сервере соответственно. На рисунке выше приведена структура сгенерированного нами пробного пакета для её последующей инкапсуляции в ICMP-пакет.

Давайте суммируем основные свойства именно этого протокола:

1. Это многопользовательский протокол, то есть он позволяет обслуживать различных пользователей сервера с индивидуальным набором команд и прав;
2. Этот протокол базируется на одноразовом пароле (OTP), то есть он устойчив к рискам прослушивания трафика и replay-атакам;
3. Протокол использует очень живучий транспорт — ICMP type 8 ping packets, — эти типичные для сети сервисные пакеты пропускает подавляющее большинство файрволов, настроенных для самых разных задач;
4. Гибкость протокола: для доступа к серверу вы можете авторизовать любой IP-адрес для подключения к нему (а не только адрес, с которого пришел пакет, как это реализовано в `knockd`) и исполнить спецкоманду на сервере удаленно;
5. Не нужен специальный клиент для работы с сервером — достаточно стандартных утилит `pingи md5` , которые есть на любой Unix-платформе (в Windows легко найти стороннюю реализацию md5, например);
6. Протокол использует криптоалгоритмы для защиты своего содержимого, что опять же выгодно отличает его от других portknocking-методик;
7. При генерации OTP используются текущие дата и время с точностью до минуты. Некоторые специалисты называют это избыточным усложнением протокола, т.к. дополнительно требуется точная синхронизация времени как на сервере, так и на клиенте, чтобы такой пакет был успешно расшифрован. Тогда как другие, наоборот считают достоинством, считая это дополнительным и неявным уровнем безопасности.

Sig^2 — двунаправленная система

Система Sig^2 (иногда записывается как *sig2knock*) достигает некоторых преимуществ Cerberus, но иным путем, в этом плане по своему устройству скорее представляя концептуально переработанную версию уже классического `knockd` . Для краткого ознакомления с ней приведем базовый алгоритм развертывания системы Sig^2.

Изначально предполагается, что администратор неким заведомо надежным способом передал логин и пароль новому пользователю системы.

1. Когда новый пользователь добавляется в PortKnocking-систему защищенного сервера, Sig^2-демон генерирует полностью случайную *инициализирующую последовательность стуков*(Initial Sequence Numbers, ISNs), после чего она шифруется с помощью хэша от логина и пароля этого пользователя. Администратор отсылает эту зашифрованную строку через любые доступные средства связи клиенту. После чего он самостоятельно сохраняет её на локальный диск в папку программы.
2. Пользователь запускает клиента Sig^2, и вводит в него свой логин и пароль. Программа вычисляет хэш от имени пользователя и его пароля, после чего клиент успешно расшифровывает им строку. После чего запускается механизм «простукивания сервера» полученной последовательностью ISNs. По завершении сам клиент переходит в режим ожидания ответа от сервера.
3. PK-демон на сервере, зафиксировав правильно отыгранную последовательность стуков от клиента, генерирует случайный номер свободного порта для текущей сессии, кроме того вычисляет новую случайную последовательность стуков, упаковывает их в ответное сообщение, шифруя оба значения хэшем на основании логина и пароля пользователя, и отправляет в UDP-пакете по координатам клиента (адрес клиента — это `IP:port` откуда была проиграна *правильная* комбинация стуков). На указанный случайный порт выполняется port mapping для запрошенного сервиса из локальной сети или самого сервера.
4. Клиент сохраняет на диск полученную от сервера новую зашифрованную строку стуков (для следующего сеанса), после чего запускает ранее указанного пользователем внешнего клиента

(например, SSH), передавая ему номер расшифрованного открытого порта. На этом сеанс считается успешно завершенным.
5. Следующий сеанс — повторяет алгоритм, начиная с шага 2.

Несмотря на очень тщательную проработку деталей, этот протокол получился достаточно «*шумным*» — на полный цикл получения допуска генерируются как минимум 2 пакета в двух направлениях.

Мутация *кнокеров*: причины и следствия

В первой и *текущей части* этой статьи мы уже отчасти рассмотрели все классические методы PK, используя приобретенные знания и опираясь на изложенную теоретическую базу, повысим градус сложности и дальше обзорно рассмотрим наиболее перспективные и продвинутые направления уже современного port knocking'а.

Но начать их обзор логично с исходного вопроса — мотивации для их создания — почему эти техники в процессе своей эволюции так существенно усложняются?

Очень сильный недостаток PK в его классическом понимании — это особенности сетевой среды, которая изначально не предназначена для подобных нестандартных операций. Например, если клиент выстукивает *серию-трель* из 8 пакетов, где гарантия того, что 5-ый пакет не придет быстрее 4-го,нарушив всю ключевую последовательность? На самом деле гарантий здесь никаких нет, и чтобы повысить вероятность правильной последовательности — клиенту лучше высылать пакеты с некоторой паузой, например в 4 секунды. Нетрудно посчитать, что авторизация в этом случае растянется на полминуты. Увеличив время таймаута на сервере это можно как-то пережить, но что случится, если в этот момент одновременно начинает стучаться какой-то другой клиент, вклиниваясь в вашу последовательность?

Как в рамках этих достаточно узких возможностей классической техники надежно бороться с replay-атаками (sequence replay attack) или с потенциальной проблемой прозрачного перехвата всего трафика, например в виде атаки «человек посередине» (man in the middle attack)?

Fwknop: прогрессивный SPA

Да, на самом деле подводных камней тут предостаточно, поэтому классический port knocking в процессе поиска решений и эволюционного развития пришел к новой технике — авторизации по одному пакету (Single Packet Authorization, SPA).

И рассмотрим мы её опять на примере.

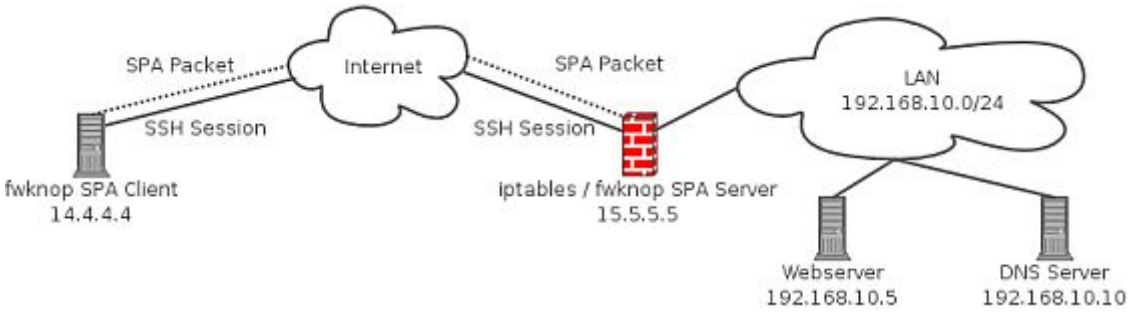
В качестве такового я выбрал fwknop (firewall knock operator) — по моему мнению, одну из самых удачных реализаций SPA. Несмотря на то, что она может работать и в режиме обычного PK, что порой очень удобно из соображений совместимости с другими подобными решениями — в нашем описании мы сосредоточимся *только* на её SPA-составляющей. Утилита написана на Perl и плотно интегрируется с возможностями вашего брандмауэра (поддерживаются `netfilter/iptables` и `ipfw`), при этом по умолчанию SPA-пакеты доставляются через протокол `UDP` (поддерживаются также TCP и ICMP). Для начала работы клиенту нужно заранее получить свой приватный DSA-ключ.

Итак, в терминологии SPA «*стуком*» называется отправка единственного и уникального «пакета идентификации». Давайте снова воспользуемся самым быстрым способом познакомиться с потенциальными возможностями системы — рассмотрим структуру такого пакета с примером его заполнения:

- Случайно сгенерированные числовые данные (16 байт)
- Имя пользователя
- Временная метка локальной системы (Timestamp)
- Версия клиента `fwknop` (для обеспечения обратной совместимости)
- Код режима: запрашивается выполнение команды или доступ к серверу?

- Адрес/порт клиента для удаленного подключения или код команды
- Контрольная сумма-подпись этого пакета (MD5)

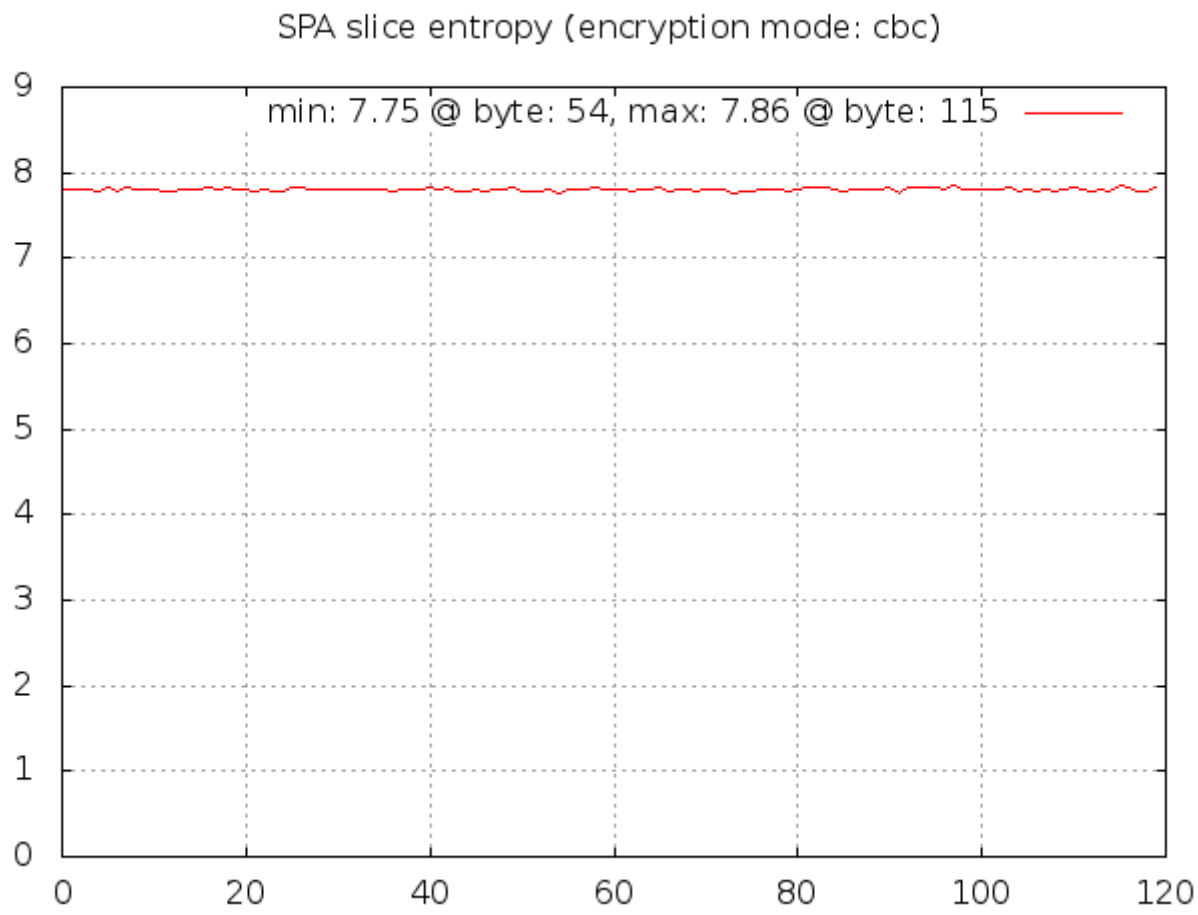
В итоге все поля собираются в единую строку, кодируемую в формате `Base64` , поля которой разделяются символом «:», и на выходе получается упакованная последовательность байтов. Далее она шифруется по алгоритму `AES` в режиме `CBC` (Cipher Block Chain), при размере блока в 128 бита и длине ключа в 256 бит. В заключительной фазе, эта строка инжектируется в выбранный транспортный пакет и отправляется в адрес fwknop-сервера, который прослушивает фиксированный порт (по умолчанию — `66201`) в ожидании новых заданий.



В самой последней версии `fwknop` можно отказаться от постоянного порта, для чего реализована сложная методика рандомизации портов принимающих SPA-пакеты, с которой всем желающим предлагаю самостоятельно ознакомиться вот здесь.

На первый взгляд кажется, что, будучи привязанным к одному случайному порту `fwknop` более заметен со стороны гипотетического сетевого наблюдателя, чем классический PK работающий посредством множества случайных портов. Но на самом деле, такой подход дает возможность отказаться вообще от каких-либо статических *сигнатурных маркеров* для SPA-пакетов: теперь любой приходящий пакет успешно расшифровывается и исполняется, либо он просто отбрасывается, если имеющиеся в системе DSA-ключи не смогли его расшифровать.

Но с точки зрения злоумышленника, не знающего целевой порт сервиса, задача обнаружения таких пакетов в общем сетевом потоке практически невыполнима. Очевидно, что при такой однопакетной авторизации также отпадают многие вышеперечисленные коллизии свойственные классическому PK, что на фоне использования сильного криптоалгоритма идентификации и вовсе выставляет традиционный PK как устаревший метод.



Статистический анализ показывает полную незаметность SPA-пакетов на фоне обычного TCP/IP-потока данных

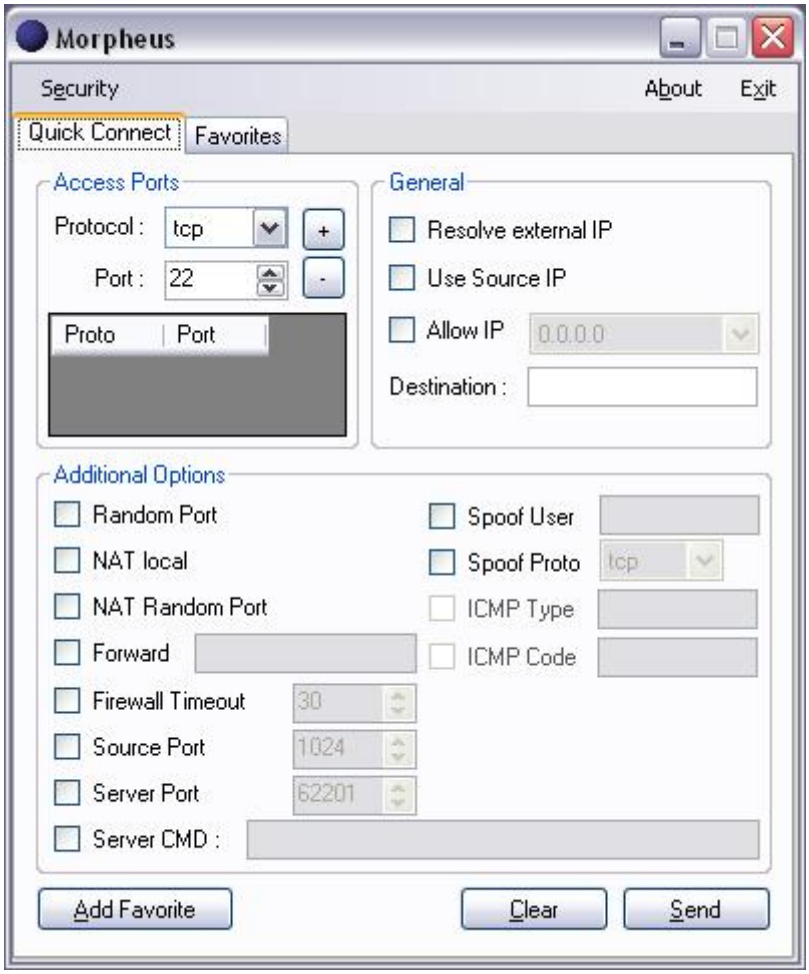
После получения SPA-пакета и его успешной расшифровки, серверный `fwknop` в целях дополнительной проверки на правомерность доступа, может выполнить сканирование системы подключаемого клиента на предмет её особенностей (passive fingerprinting), для чего заранее необходимо установить стороннюю утилиту p0f.

Кстати говоря, после долгого перерыва она, наконец, снова обновилась и продолжила своё развитие. В отличие от известного сканера nmap, p0f использует полностью пассивные механизмы fingerprinting'а, то есть не генерируется никакого дополнительного трафика, который может быть обнаружен исследуемой стороной.

И только если каскад из этих проверок пройден успешно — заданная во входящем fwknop-пакете команда исполняется целевым PK-сервером.

В заключение рассмотрения `fwknop` отмечу три момента. Во-первых, сама криптографическая реализация идентификации базируется на стандарте ISO/IEC 9798-2 для высоконадежных систем. Для UDP-пакетов возможен дополнительный слой ассиметричного шифрования через GPG, кроме того клиент `fwknop` умышленно спроектирован так, что «*тайные команды*» на сервер в целях дополнительной конспирации можно отправлять через анонимную сеть Tor Onion Network (как?). Второй момент — это наличие поля «временных штампов» у каждого SPA-пакета. В отличие от уже рассмотренной выше похожей PK-системы Cerberus, в `fwknop` не требуется синхронизации времени взаимодействующих сторон, и это поле пока никак не используется на стороне сервера.

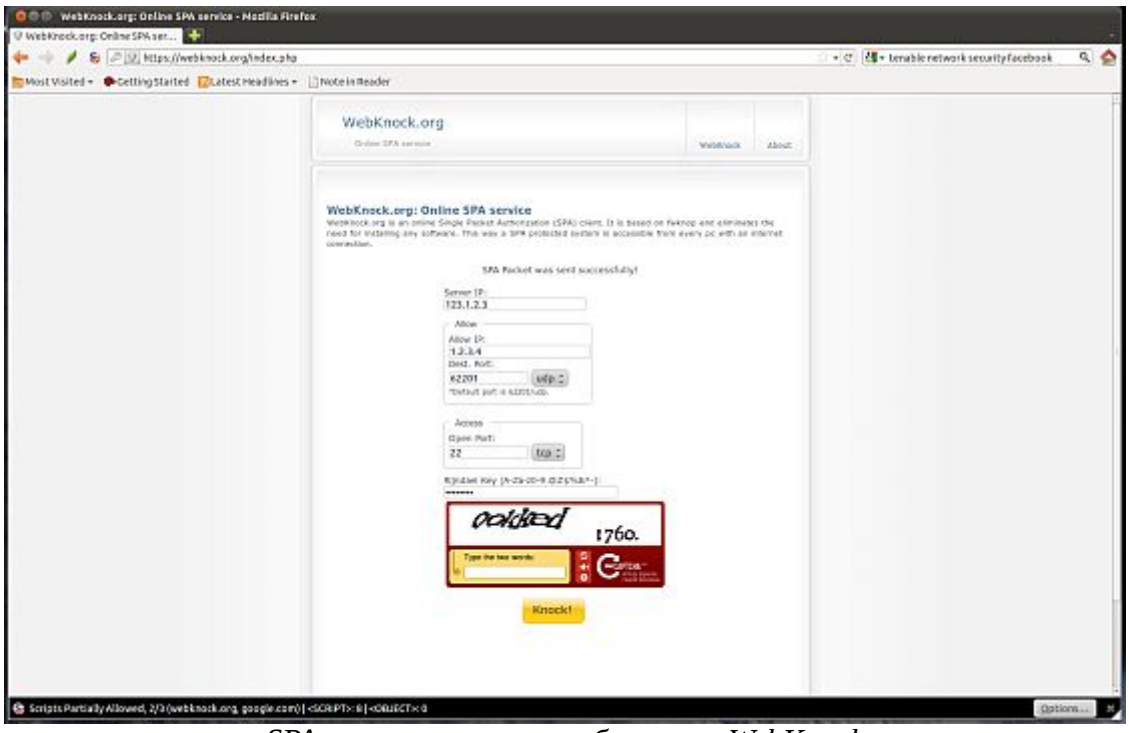
Поэтому для противодействия *replay-атакам* здесь применяется другая, более удобная для всех логика: сервер хранит хеши всех обработанных им пакетов. С учетом первого статического блока внутри зашифрованного пакета (случайная комбинация символов), а также динамического `timestamp`'а — клиент не может даже теоретически снова сгенерировать идентичный пакет, поэтому любой повторно принятый пакет *автоматически отвергается* со стороны сервера.



Графический клиент morpheus-fwknop

Третий, заключительный пункт касается немаловажного момента работы с клиентской частью `fwknop` — она будет работать везде, где доступен Perl (для Windows есть Strawberry Perl). Самые последние версии этой программы были полностью переписаны на Си (сохранена полная совместимость по протоколу), доступны её сборки под Linux, Mac OS X, *BSD. Есть графический Win32-клиент для Windows (поставляется с исходниками на Delphi), а также новый проект на .Net — morpheus-fwknop(смотрите рис. выше).

В дополнение имеется решение для Android и даже web-прокси WebKnock (смотрите рис. ниже), который позволяет послать свой SPA-пакет с любого стационарного или мобильного устройства подключенного к Интернету.



SPA-авторизация через браузер – WebKnock

При использовании `fwknop` в режиме обычного PK (non-SPA) подходит множество ранее упомянутых классических клиентов для knockd. Для тестирования правильности сопряжения и настроек клиента и сервера любезно поставляется специальный сервисный скрипт.

Port knocking на перекрестках истории

Исторически идея port knocking'а была впервые сформулирована в открытых источниках в начале 2001 года в почтовой рассылке *German Linux User Group*. Сам термин был придуман и озвучен специалистом по сетевой безопасности *Мартином Крживинским* (Martin Krzywinski) в 2003 году в журнале *SysAdmin Magazine*. Первым практически реализованным «кнокером» стал *Cerberus*, который кроме этого впервые использовал усложненную методику *одноразовых паролей* (One Time Password, OTP). В целом, из-за однотипности и простоты классического PK, всех его представителей принято обозначать как **TPK** (traditional port knocking), где наиболее типичный и яркий представитель — `knockd` .

Дальнейшее совершенствование методов сокрытия привело к созданию механизма авторизации по *единственному крипто-пакету* (Single Packet Authorization, SPA), который был впервые продемонстрирован в рабочем виде в 2005 году на конференции BlackHat. Самая удобная и популярная среди широких народных масс реализация SPA была представлена чуть позже — в 2008 году *Майклом Рашем* (Michael Rash), который создал `fwknop` (программа до сих пор активно развивается). Третье и последнее поколение «кнокеров» — это различные *гибридные техники* (hybrid port-knocking, HPK), сочетающие в себе сильные стороны от самых разных концепций и течений (иногда в силу экстравагантности своих подходов ставящие под сомнение даже свою принадлежность к PK).

Позади закрытых дверей: Port knocking. Часть 3

Третья, заключительная часть моей серии статей про portknocking (PK). Начало этой серии читайте здесь.

Сегодня мы завершим рассмотрение этого метода скрытия своих подключений и любого взаимодействия (например, авторизации) с целевым сервером, рассмотрев самый навороченный и современный вид PK — *гибридный*. Также мы подведём итог всему этому делу, укажем слабые и сильные места всех рассмотренных подходов.

Гибридный PK: сумма всех лучших технологий

Гибридными называют технологии PK, в рамках которых сочетаются сразу несколько разнородных концепций. Обобщая, можно выделить типичные составляющие гибридного PK (HPK): традиционный port-knocking, стеганография, криптография, реализация обоюдной авторизации (часто двухсторонняя SPA).

Не нужно думать, что подобные «слоеные бутерброды» изобретают лишь для увеличения формальной сложности защиты — скорее это попытка ювелирно подогнать друг под друга разные техники, чтобы каждая органично нивелировала своими преимуществами недостатки другой — и наоборот.

Существует несколько работ подробно описывающих HPK, из-за недостатка места я сосредоточусь лишь на двух, — типичных преимуществах этого подхода:

1. **Проблема идентификации пакета**: если с одной стороны, есть некая явная и однозначная сигнатура для каждого «*магического пакета*» (как в Cerberus), то такие пакеты будут рано или поздно обнаружены и выделены из общего TCP/IP-потока злоумышленником. С другой стороны, если опасаясь этого, такие пакеты никак не маркировать, заставляя сервер PK каждый входящий пакет «вслепую» распаковывать и верифицировать по некой контрольной сумме (как в `fwknop`), то это создает весьма ощутимую нагрузку на сервер. Это открывает возможность злоумышленнику для намеренной эксплуатации «*родовой особенности*» этой техники, что потенциально приведёт к возникновению сильнейшего DDoS-эффекта.

   HPK в этой дилемме занял выигрышную компромиссную позицию: он маркирует каждый свой «*магический пакет*», но при этом для сокрытия идентифицирующего признака применяются продвинутые методы *стеганографии*. Как пример последнего, номер запрашиваемого для открытия сервиса/порта + пакет авторизации здесь может быть передан даже не на транспортно-сетевом уровне TCP/IP, а на прикладном уровне (или их смешении), — будучи внедрены в обычное со всех точек зрения png-изображение.

2. **Проблема replay-атак**: в HPK всегда применяется двухсторонняя авторизация, таким образом, здесь PK осуществляется всегда в две стороны, что устраняет множество потенциальных проблем и неудобств, которые свойственны другим PK-техникам, пытающихся противодействовать replay-атакам с помощью различных полумер (необходимость *предварительной синхронизации времени* на клиенте и сервере, как в Cerberus; хранение базы из хэшей всех обработанных пакетов, как в `fwknop` и так далее). У такого подхода есть ещё несколько преимуществ, но для баланса отметим и главный недостаток — общий алгоритм взаимодействия при этом резко усложняется.

У разных реализаций HPK имеются множество нюансов, поэтому предлагаю очень кратко коснуться технических особенностей лишь одной, но весьма достойной — Tariq:

- Здесь очень ясный и чистый код на Python, в силу того, что логика сервера — это лишь надстройка над пакетом scapy. В результате все низкоуровневые детали его протокольно-транспортного устройства изолированы от самой системы.
- Tariq не прослушивает множество TCP/IP-портов, как это делают стандартные PK-сервера. Он использует возможности `scapy` для пассивного сниффинга всего входящего трафика, равно как и для конструирования обратного пакета-ответа.
- Для ещё большего упрощения логики устройства, Tariq использует сторонние библиотеки: шифрования (GnuPG), стеганографии (steganogra-py) и возможности упомянутого scapy для всего остального спектра сетевых операций.
- В данном случае для доставки скрытых зашифрованных пакетов используется приведенный выше пример — обычные графические изображения (прикладной уровень OSI), для работы с которыми дополнительно понадобится библиотека Python Imaging Library (PIL).
- Tariq поставляется сразу как совмещенный пакет — с сервером и клиентом, укомплектованный всеми необходимыми библиотеками, и может быть запущен в качестве клиента везде, где доступна среда исполнения Python (в том числе на мобильных платформах).

Несмотря на своё бесспорное техническое совершенство, HPK обычно алгоритмически очень сложен, что особенно ощущается при подборе клиента и иногда лишает мобильности и простоты его использования.

Watch_dns: пример высшей магии скрытой авторизации

Известный американский специалист по безопасности *Брайн Хатч* (автор популярной серии книг «Linux Exposed») описал и реализовал в 2009 году систему для прослушки DNS-запросов на предмет скрытых *knock-knock* запросов. В его оригинальной реализации простая Perl-программа под названием `watch_dns` просматривала весь UDP-трафик поступающий на локальный DNS-сервер, формируя список всех запрашиваемых хостов, а также IP-адреса источников запросов.

Второй Perl-скрипт в режиме реального времени сканировал это список и по специальному правилу выделял доменные имена, интерпретируя их как команды-запросы (для чего был разработан довольно простой, но в тоже время гибкий язык *HidenHand*).

Даже с точки зрения внешнего наблюдателя, который имел бы полный доступ к трафику обмена сообщениями между сервером и клиентом, не происходило бы ничего необычного — DNS-сервер, также как и все остальные сервисы работают штатным образом. Здесь, в отличие от обычного *port knocking*'а, нет вообще никаких бессмысленных, подозрительных или странных пакетов (например, ACK-запросов без ответа и т.п.). Но на самом деле *HiddenHand* позволяет не только открывать порты по запросу клиента, но также удаленно конструировать новые правила для работы системы безопасности, абсолютно незаметно, через имена подходящих по названию доменов-команд, динамически корректируя её логику работы и настройки.

Пока мы сосредоточились лишь на серверной стороне этой реализации, но стоит отметить также и удивительную простоту управления подобным механизмом со стороны клиента. В самом деле, здесь не нужно использовать какой-то сторонний «сакральный код» — для этого подходит множество подручных и обыденных программ, таких как `nslookup` или `host` , также можно воспользоваться веб-интерфейсами аналогичных сервисов. Но на самом деле все обстоит даже ещё проще. Брайан

рекомендует создание на *авторитетном DNS-сервере домена* (находящимся полностью под вашим контролем) настройку для произвольного субдомена, который будет использоваться именно для этой цели.

Например, для созданного домена третьего уровня `my.megashop.net` можно настроить *тригеры прослушки* и интерпретации ваших DNS-команд, при каждом обращении к нему. После этого можно использовать обычный браузер, для запроса к подобному домену, например запрос к ресурсу `static.my.megashop.net` может открывать порт с SSH, а обращение к изображениям хранимым на `img.my.megashop.net` будет снова закрывать его. На самом деле, подобные запросы могут исходить откуда угодно, не обязательно от вашего браузера, а, например, от стороннего сайта, который хостит страницу, элемент которой (.css-файл или *картинка-аватор*, как в примере Брайана) загружается с данного ресурса.

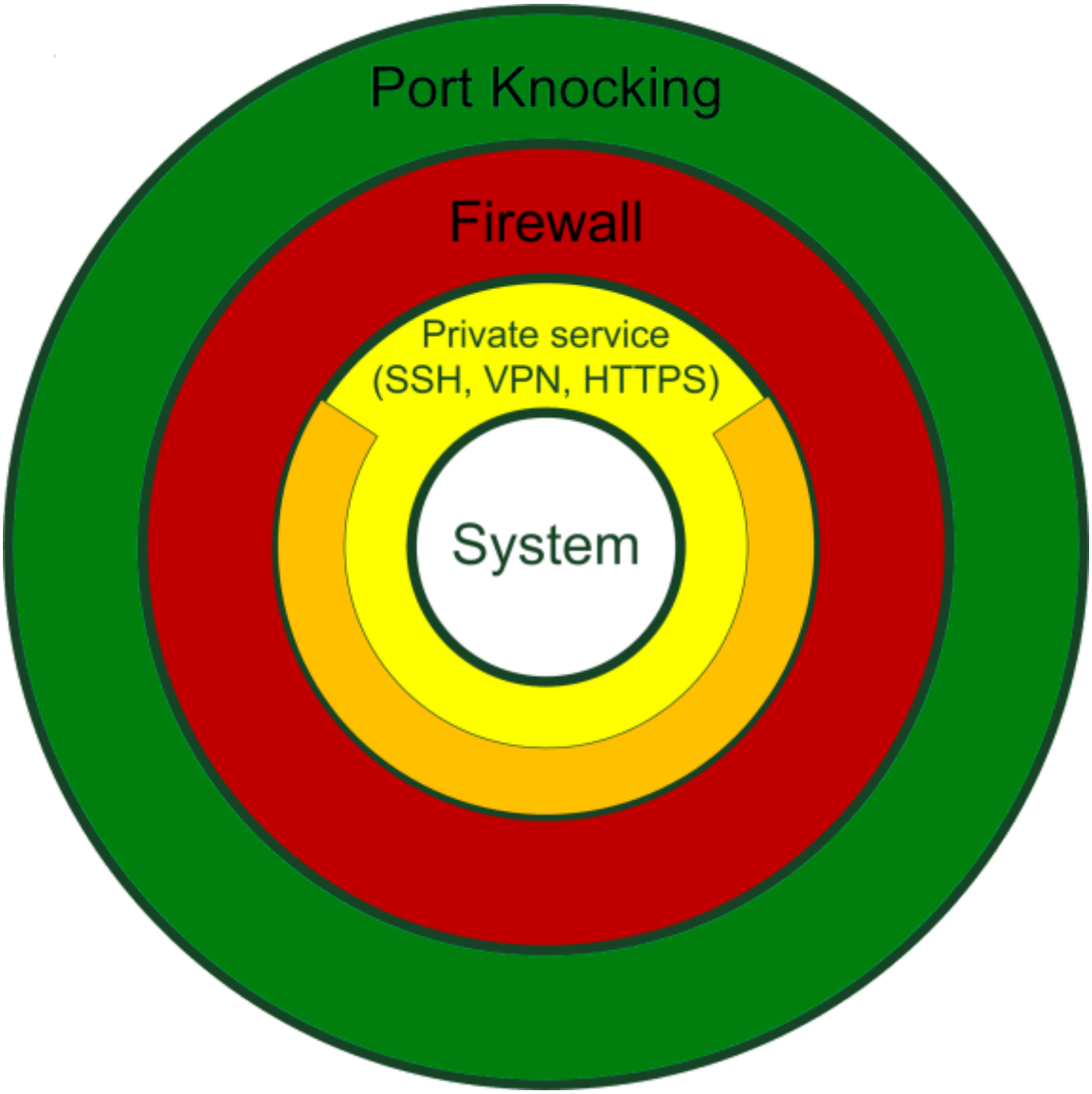
Брайан приводит рабочую демонстрацию этого, когда вход на его любимый технический форум (после авторизации на форуме происходит подгрузка аватора с «магического субдомена»), вызывает вышеописанный эффект «*веерно падающего домино*», когда в конечном результате SSH «вдруг» оживает и начинает принимать запросы. Фактически, на базе идей PK здесь реализована классическая двухфазовая авторизация (в данном случае SSH), за тем лишь отличием, что в отличии, например, от подобной двухступенчатой схемы у Google, здесь факт прохождения идентификации носит *крайне неявный характер* (успешный вход на публичный форум = первая фаза аутентификации при доступе к приватному SSH).

Хочу отдельно подчеркнуть, что этот дополнительный уровень сокрытия и безопасности очень прост в реализации. Брайн использовал три простых Perl-скрипта, на написание и отладку которых у него ушел один *викэнд*.

Словесы заключительные

В заключение ещё раз подчеркну: port knocking не претендует заменить традиционную аутентификацию, это лишь дополнительный слой безопасности, который эффективно нейтрализует огромное количество специализированных инструментов и методов, разработанных для автоматизации подбора паролей, изучения уязвимостей публичных сервисов и поиска ошибок настройки, а также последующего их взлома или незаконного использования.

Во многом PK основывается на известном принципе «security through obscurity» и позволяет вывести наиболее чувствительные сервисы из зоны прямой видимости (и досягаемости) злоумышленника.



Но, как и любая технология, PK не лишен недостатков. Традиционная схема (TPK) подвержена опасности нарушения последовательности доставки (или потери) пакетов из-за естественных сетевых коллизий, сложностям в некоторых случаях при подключении через NAT/PAT (в этом случае можно попробовать более всеядный вариант — uPortKnock), использовании в многопользовательской среде или на нагруженных системах.

Кроме того на неё легко осуществить своего рода DDoS-атаку: когда злоумышленник бомбардируя потоком случайных «стуков» защищенный сервер (и для создания этого «шума» хватит даже низкоскоростного подключения), блокирует всякую возможность авторизоваться у легального пользователя. Большинство этих и других проблем успешно решаются в более современных реализациях PK, в первую очередь в вышерассмотренных методиках на основе **SPA** и **HPK**.