«Все лгут». Погоня за истиной при поиске руткитов

Алиса Шевченко, eSage Lab
alisa@esagelab.ru

Дмитрий Олексюк, eSage Lab
dmitry@esagelab.ru

Вступление

Задача охотника за руткитами - будь то человек или программа - сводится к получению правдивой информации, на основании которой можно судить о состоянии системы. У руткита диаметрально противоположная задача: любыми способами скрыть истинное положение вещей. В силу такого конфликта интересов охотник за руткитами вынужден в своих поисках исходить из презумпции ложности любых полученных данных и непрерывно изыскивать такие источники информации, которые могут быть сочтены доверенными в потенциально модифицированной руткитом системе.

Проще говоря, выбор правильных источников информации - краеугольный камень задачи о поиске руткитов. Это также и динамический процесс, поскольку с течением времени и эволюции руткитостроения доверенные источники информации имеют свойство утрачивать свою правдивость.

В данной статье мы хотели бы осветить один достаточно простой, и, вместе с тем, более эффективный и безопасный, чем общепринятые, метод получения информации о системе. Вначале мы вкратце рассмотрим преимущества и ограничения существующих технологий, широко применяемых в антивирусах и антируткитах. После этого мы приведем детали предлагаемого метода, его плюсы и минусы, очевидные пути обхода и пример практической реализации.

Стандартные способы получения информации о системе

В современных антивирусах и антируткитах применяются различные подходы к выявлению аномалий, характерных для модифицированной руткитом системы: сопоставление полученных из разных источников списков объектов, сопоставление системных структур и кода с доверенной моделью и поиск отклонений в них.

Независимо от конкретного подхода к поиску аномалий, выбор механизмов сбора потенциально доверенной информации о системе - т.е. информации, потенциально не искаженной вмешательством руткита - ограничен.

Фактически, выбор сводится к двум опциям.

1. Восстановление предположительно модифицированных участков кода ядра (global unhooking) перед сбором информации о системе.
2. Сбор информации с более глубокого уровня архитектуры системы, чем наиболее глубокий предполагаемый уровень rootkit-модификаций.

Восстановление ядра

Восстановление участков кода ядра - это способ нейтрализовать эффекты руткита глобально, для системы в целом. Обычно в рамках этого подхода восстанавливают таблицу SDT, некоторое количество кода в начале системных функций, IRP-обработчики - и, в общем, все системные структуры, подозреваемые в модификации. (См. Рис. 1).

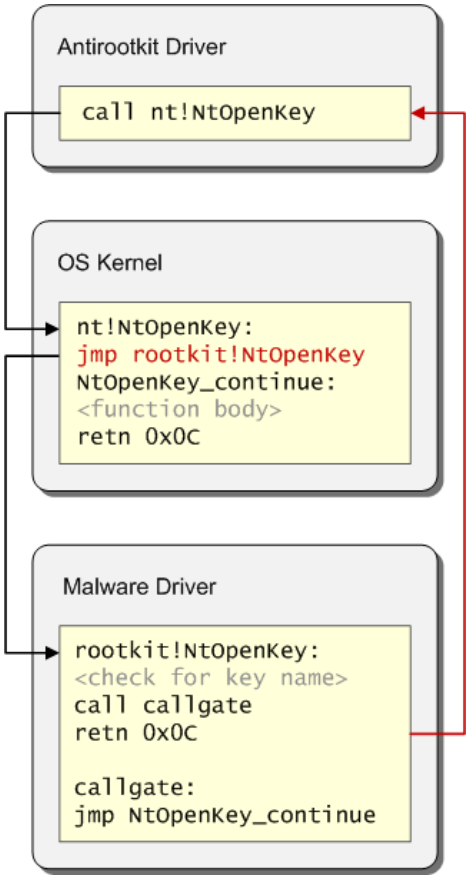

Рис. 1. Rootkit-модификация ядра

В процессе разработки алгоритмов восстановления структур ядра разработчик сталкивается с несколькими сложностями.

1. Поиск или вычисление «чистых» указателей на системные вызовы, IRP-обработчики и т.д., используемые впоследствии для восстановления цепочек системных вызовов. Также - поиск оригинальных системных файлов, из которых можно восстановить модифицированные участки ядра.
2. Различение безвредных модификаций и перехватов, установленных легитимным программным обеспечением (например, фаерволом или антивирусом).
3. Обеспечение безопасности алгоритма. Поскольку ядро системы непрерывно используется, запись в исполняемые регионы памяти ядра в определенных обстоятельствах может привести к BSOD.

Единственное преимущество глобального восстановления структур ядра заключается в том, что при удачном стечении обстоятельств удается полностью развеять «магию» руткита, возвращая всем приложениям в системе полный доступ к неискаженной системной информации.

Вместе с тем, техника глобального восстановления структур ядра обладает рядом существенных недостатков.

1. Небезопасность. Модификация структур в работающем ядре - рискованная операция при любых условиях.
2. Ненадежность. Руткит может в любой момент восстановить свои перехваты.
3. Бессистемность. Должны быть указаны конкретные структуры и участки кода, подлежащие восстановлению, что позволяет руткиту скрыться в непредусмотренной локации.
4. Трудоемкость. Получение оригинальных указателей на системные функции и различение вредоносных модификаций и перехватов от безопасных.

С учетом приведенных недостатков можно утверждать, что глобальное восстановление структур ядра - скорее, примитивная реакция на известные угрозы, чем универсальный способ получения неискаженной информации. Кроме того, вследствие небезопасности данного подхода, он не может использоваться (и обычно не используется) в защитных решениях, претендующих на высокую надежность.

Углубление в систему

Поскольку архитектура Windows NT имеет многоуровневую структуру, информация, проходящая по цепочке от ядра до API и прикладных программ, может быть перехвачена на различных участках этой цепочки. Таким образом, антируткит может попытаться обойти искаженные элементы цепочки, если будет запрашивать информацию с ее участков, исполнительно предшествующих предположительной модификации.

Этот подход намного безопаснее и универсальнее, чем предыдущий, но так же ограничен рядом недостатков.

1. Трудоемкость. Углубляясь в систему, разработчик вынужден самостоятельно обеспечивать абстракции и преобразования информации, которые в норме обеспечиваются высокоуровневыми механизмами.
2. Стратегическая неэффективность. Углубление в систему стимулирует «гонку вооружений», поскольку достаточно очевидно обходится ответным углублением противника.

Типичный пример гонки вооружений: руткит перехватывает системные вызовы с целью маскировки файлов - антируткит обращается к драйверу файловой системы - руткит перехватывает обработчики прерываний драйвера файловой системы - антируткит обращается к драйверу диска, и так далее.

В конце концов разработчик защиты вынужден эмулировать всю операционную систему, чтобы обойти руткит.

Альтернативное решение

Из всех недоверенных источников информации, собственные источники являются наименее недоверенными. Антируткит может выполнять системные вызовы самостоятельно, если он оснащен собственной, заведомо чистой копией ядра. Это «дешевый» способ получить чистую информацию в обход искажающих модификаций, не рискуя стабильностью системы.

Использование собственного ядра, правильно полученного и настроенного, позволяет надежно детектировать большинство современных руткитов ядра. В частности, облегченная реализация данной техники (описанная ниже в этой статье) позволяет обнаруживать объекты, скрытые посредством модификации таблицы SDT и кода ядра. Более сложная реализация позволит детектировать практически любой руткит ядра.

Детали техники

Создание работающей копии ядра относительно просто и может быть реализовано посредством следующего алгоритма. (См. Рис. 2).

1. Поиск необходимых исполняемых файлов. Для минимальной реализации работающей копии ядра достаточно двух файлов: основной файл ядра (в большинстве случаев - ntoskrnl.exe) и hal.dll. Поскольку пути к этим файлам могут быть подменены руткитом, их рекомендуется вычислять посредством анализа аппаратной конфигурации, описанного ниже.

2. Загрузка файлов в память. Чтение файлов рекомендуется осуществлять методом прямого доступа к диску, чтобы снизить вероятность подложных данных.
3. Релокация указателей и инициализация переменных.
   В норме, все переменные ядра в собственной копии должны быть инициализированы вручную. Но, поскольку это трудоемко и не является критичным для детектирования большинства руткитов, переменные могут быть скопированы из глобальной копии ядра, за исключением наиболее критичных, часто модифицируемых руткитом (таких как pIofCallDriver и pIofCompleteRequest).
4. Временная блокировка механизма системных нотификаторов.
   Легитимные callback механизмы не должны вмешиваться в работу антируткита. Отключить их можно патчингом функции ExReferenceCallBackBlock так, чтобы она во время работы антируткита безусловно возвращала нулевое значение.
5. Перенаправление собственных вызовов ядра на локальную копию.

Поскольку мы должны предполагать, что очевидные источники информации о пути к файлу ядра (такие как файл boot.ini или ключ реестра HKLM\System\CurrentControlSet\Control\SystemStartOptions "KERNEL") могут быть подставными, рекомендуется использовать вычислительный алгоритм получения имени файла ядра. Последнее определяется двумя системными параметрами: количеством процессоров и поддержкой PAE (Physical Address Extension). (См. Рис. 3).

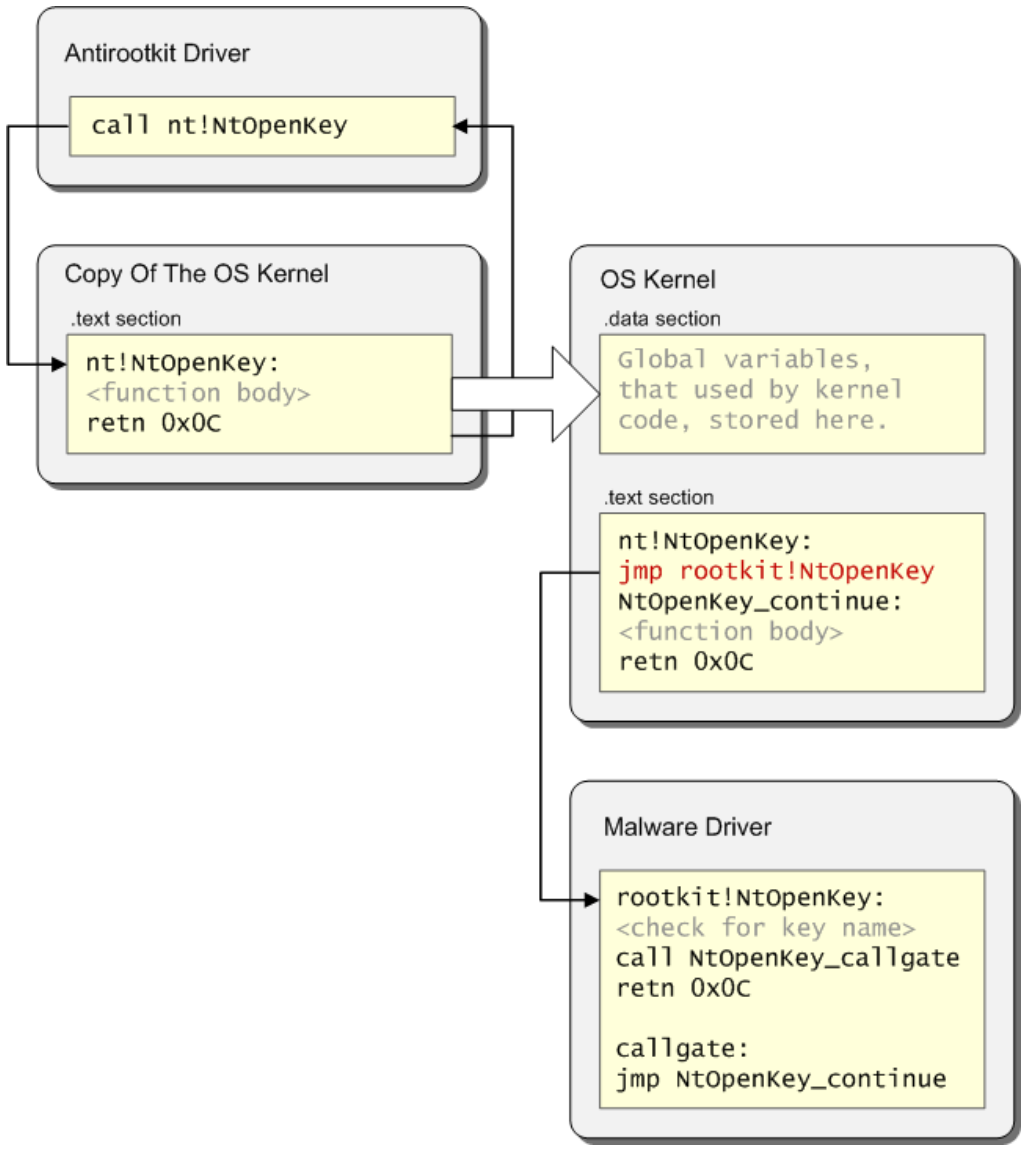

Рис. 2. Цепочка вызовов дубликата ядра

Имя файла ядра	Поддержка PAE	Мультипроцессорность
ntoskrnl.exe	-	-
ntkrnlpa.exe	+	-
ntkrnlmp.exe	-	+
ntkrpamp.exe	+	+

Рис. 3. Таблица для определения имени файла ядра

Обход

Пути обхода данной техники сводятся к фальсификации или блокированию тех немногочисленных внешних источников информации, на которые полагается антируткит. В частности, путей и содержимого файлов, используемых для построения собственной копии ядра. Рассмотрим некоторые примеры обходов с решениями.

1. Руткит может подставить антируткиту модифицированный файл ntoskrnl.exe при его чтении с диска. Решение: проверка подписи Microsoft.
2. Руткит может подставить ложный путь к файлу ядра. Решение: получение имени файла посредством анализа аппаратной конфигурации.
3. Руткит может заблокировать чтение файлов ядра. Маловероятно, так как это может нарушить функционирование легитимных приложений.

Преимущества

1. Безопасность. Грамотные манипуляции с собственной копией ядра абсолютно безопасны, в отличие от манипуляций с работающим кодом ядра.
2. Надежность. Руткит никогда не сможет установить перехваты в локальной копии ядра и, таким образом, исказить получаемую антируткитом информацию, поскольку ничего не знает о местонахождении копии.
3. Правдивость. Код ядра, прочитанный с диска вручную и инсталлированный с соответствующей предусмотрительностью, гарантированно целостен. Это значит, что любые данные, полученные посредством вызова функций из данной копии, гарантированно чисты - если только не модифицирован сам источник данных.

Заключение

Не претендуя на панацею, описанная в данной статье техника представляет собой дешевый и безопасный способ обнаружить деятельность большинства руткитов ядра посредством обнаружения скрытых ими объектов.

В качестве проверки практической ценности предложенной техники, мы разработали утилиту на ее основе: Trojan.Win32.TDSS remover. [1] Утилита «заточена» под детектирование и лечение руткита TDSS[2], хотя по своим возможностям представляет собой достаточно универсальный антируткит.

Несмотря на то, что качественная реализация описанной техники достаточно тривиальна и безопасна, нам не известны прецеденты ее применения в антивирусных программах и антируткитах.

Ссылки

[1] eSage Lab, Trojan.Win32.TDSS remover
http://www.esagelab.ru/resources.php?s=tdss_remover

[2] Алиса Шевченко, Анализ руткита TDSS
http://www.nobunkum.ru/issue001/tdss-analysis.html