Лекция 20: Безопасность

Алексей Линёв
Александр Мощук
Кирилл Погорельский

somes slides are adapted from the OS course at the University of Washington

Содержание

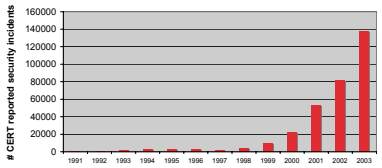
- Обзор "классических" вопросов безопасности
  - задача: обеспечить безопасное разделение ресурсов
  - основные принципы
  - Trusted Computing Base (TCB, базовое обеспечение безопасных вычислений)
- Современные проблемы безопасности
  - черви (worms)
  - армии взломанных систем (botnets)
  - программы-шпионы (spyware)

Безопасное разделение ресурсов

- Обеспечение защиты автономного компьютера с одним пользователем – несложная задача
  - Никто не имеет доступа к данным Вашего компьютера
  - Никто не может устанавливать дополнительное ПО
  - Никто не может атаковать Вас по сети
- Обеспечить защиту разделяемых ресурсов – действительно сложная задача
  - Нужно предотвращать доступ пользователей к "чужим" приватным данным
    - при этом предоставлять доступ к ним "правильным" пользователям
    - например, файлы с оценками, клавиатурный ввод
  - Нужно предотвращать использование пользователями слишком большого количества ресурсов
    - например, дискового пространства
  - Нужно предотвращать вмешательство пользователей в работу программ, запущенных другими пользователями
    - подмена содержимого экрана, замена кода программы вредоносным кодом, уничтожение процессов и т.д.

"Обеспечение безопасности – искусство, а не наука"

- Доказать безопасность системы очень сложно
- Безопасность основывается на наборе базовых принципов и "передовом опыте" (best practices)
  - опытным путем выявляются часто встречающиеся типы брешей в системе безопасности
  - безусловно, нужно стремиться к чему-то большему…



Почему так сложно обеспечить безопасность?

- Обеспечение безопасности снижает скорость работы системы
- Безопасность может помешать работе (firewall)
- Безопасность бесполезна в отсутствие попыток атаки
- Как правило, только правительство платит за обеспечение безопасности
  - а Internet делает всех нас потенциальными объектами атаки
- Все делают ошибки

Принцип наименьшего уровня привилегий (Principle of Least Privilege)

- Точно определяем, какие возможности необходимы программе для выполнения, и только их и предоставляем
  - начинаем с пустого перечня возможностей
    - запускаем программу и смотрим, когда она прервется
    - добавляем новые возможности по мере необходимости
- UNIX: существование "суперпользователя" (root) – отрицательный пример данного принципа
  - некоторым программам необходимо выполняться от имени суперпользователя, хотя им нужны минимальные возможности
    - например, ftp-демон требует выполнения от имени root'а
      - для привязки сокета и приема запросов на соединение через порт с номером <1024
      - для переключения на контекст другого пользователя после успешной идентификации
    - однако, если программа выполняется от имени root'а, она имеет доступ к любому объекту файловой системы

Взлом wu-ftpd

- wu-ftp пытается выполняться с минимальными привилегиями
  - но иногда повышает свой уровень привилегий:

```
seteuid(0);
// some privileged "critical section" runs here
seteuid(getuid());
```

  - однако, wu-ftpd не запрещает сигналы UNIX
    - таким образом, в процессе выполнения критической секции wu-ftpd можно прервать, и тогда он перейдет к выполнению обработчика сигнала
    - удаленный пользователь может инициировать запуск обработчика сигнала, прервав процесс скачивания!
      - правда, нужно поймать момент, когда wu-ftpd находится в критической секции
    - wu-ftpd не восстанавливает уровень привилегий по выходе из обработчика сигналов
      - результат: прерываем закачку – wu-ftpd работает от имени root'а
      - имеем полный доступ ко всей файловой системе

Принцип полной проверки возможностей (Principle of Complete Mediation)

- Проверяются **все** обращения ко **всем** объектам
  - в некоторых случаях можно уменьшить потери (например, используя кэширование)
    - но только в случае уверенности, что в системе не изменилось ничего существенного… а важных в данном отношении событий достаточно много!
- например, NFS и дескрипторы файлов
  - NFS – не слишком хороший пример полной проверки
  - протокол NFS:
    - клиент связывается с удаленным демоном "mountd" для получения файлового дескриптора для файловой системы, экспортированной на удаленном узле
      - это производится в момент монтирования удаленной ФС
      - mountd на удаленной машине проверяет права доступа в момент выполнения операции монтирования
    - файловый дескриптор – это "возможность": клиент предъявляет его в запросах на выполнение операций чтения/записи
      - проверка прав доступа не происходит после выполнения операции монтирования!
    - используем сетевой сниффер (sniffer) для получения файлового дескриптора
      - получаем доступ к экспортированной файловой системе не проходя проверку прав доступа

Принцип безопасных настроек по умолчанию (Principle of Fail-Safe Defaults)

- Начинаем с полного запрещения доступа, впоследствии позволяем делать только то, что разрешено явно
  - теперь ошибки настроек безопасности будут проявляться в виде "ошибочных отказов" (“false negatives”)
    - кому-то отказали в доступе, хотя должны были разрешить
  - противоположный подход ведет к возникновению "ошибочных разрешений доступа" (“false positives”)
    - кому-то разрешили доступ, хотя не должны были разрешать
    - "плохие парни" обычно не сообщают о возникновении подобных ситуаций
- Пример: Irix по умолчанию устанавливается с “xhost +”
  - что позволяет с любого компьютера открыть окно на Вашем экране и считать Ваш клавиатурный ввод!

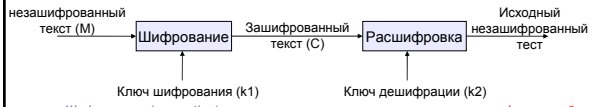
Безопасность за счет сокрытия информации (“Security through Obscurity”)

- Безопасность за счет сокрытия информации
  - «добиваемся безопасности» посредством скрытия деталей реализации системы
  - система должна быть безопасна даже в случае открытости ее реализации!
    - в действительности, открытость способствует повышению безопасности системы, поскольку сторонние разработчики могут исследовать реализацию и находить и исправлять дыры в системе безопасности
  - лучше полагаться на математические исследования и качественную архитектуру
- Контрпример: сотовые телефоны GSM
  - комитет GSM разработал собственный алгоритм шифрования и "закрыл" его
    - "клонирование невозможно"
  - "социальный взлом" и "reverse engineering" позволили разобраться в алгоритме
    - он оказался очень слабым
    - можно сыграть с микросхемой идентификации в "20 вопросов" и узнать секретный ключ в течение нескольких часов

Базовое обеспечение безопасных вычислений (Trusted Computing Base, TCB)

- Внимательно обдумайте, чему вы доверяете свои данные
  - если вы набираете пароль на клавиатуре, вы доверяете
    - производителю вашей клавиатуры
    - производителю вашего компьютера
    - вашей операционной системе
      - включая драйвер клавиатуры
    - библиотеке работы с паролями
    - приложению, проверяющему пароль
  - как насчет компилятора, который скомпилировал все вышеперечисленное программное обеспечение?
- TCB – совокупность программных и технических средств, создаваемая и поддерживаемая для обеспечения защиты средств вычислительной техники или автоматизированных систем от несанкционированного доступа к информации
  - Руководящий документ "Защита от несанкционированного доступа к информации - Термины и определения" Утверждено решением председателя Гостехкомиссии России от 30 марта 1992 г.
  - чем она меньше – тем лучше
    - общедоступные web-киоски НЕ должны быть в вашем TCB
    - должен ли быть там ваш web-броузер?

Отступление на тему шифрования



- Шифрование (encryption): на входе – ключ и данные, на выходе – зашифрованный текст (ciphertext): $E_{k1}(M) = C$
- Расшифровка (decryption): на входе – зашифрованный текст и ключ, на выходе – восстановленные данные: $D_{k2}(C) = M$
- Симметричные алгоритмы (алгоритмы с секретным ключом)
  - k1 = k2 (или можно получить k2, зная k1)
- Алгоритмы шифрования открытым ключом (Public-Key Algorithms)
  - ключ дешифрации (k2) не может быть вычислен по ключу шифрования (k1)
  - ключ шифрования можно сделать общедоступным!
    - ключ шифрования = "открытый ключ" ("public key")
    - ключ дешифрации = "секретный ключ" ("private key")
- Хеширование: на входе – произвольные данные, на выходе – значение хеш-функции фиксированного размера ("хеш", "fingerprint ")
  - Невозможно по значению хеша вычислить исходные данные, сложно (в идеале – невозможно) найти два набора данных с одним значением хеш-функции
  - H(Attack at Dawn) = 183870
  - H(attack at dawn) = 465348

Технические приемы

- Идентификация (Authentication) – установление личности пользователя или опознание программы ("Кто ты?")
- Авторизация (Authorization) – определение прав доступа пользователя или программы к различным объектам ("Что тебе позволено делать?")
  - при полной проверке возможностей проверяются все запрашиваемые действия со всеми объектами
- Аудит (Auditing) – регистрация действий пользователей и программ для последующего анализа ("Что происходит?")

Идентификация (Authentication)

- Как компьютер узнает, кто я?
  - имя пользователя + пароль
    - как хранить пароль?
    - как проверять пароль?
    - насколько надежен пароль?
  - открытые/секретные ключи (public/private keys)
  - одноразовые ключи (one-time keys)
  - биометрические данные
- Что компьютер делает с этой информацией?
  - назначает вам некоторый идентификатор
    - UNIX: 32-битное число, хранящееся в дескрипторе процесса
    - Windows NT: 27-байтное значение, хранящееся в *маркерах доступа* (*access token*) в структурах, используемых в ядре

Хранение паролей

- CTSS (1962): файл паролей {имя пользователя, идентификатор пользователя, пароль}

```
Bob, 14, “12.14.52”
David, 15, “allison”
Mary, 16, “!ofotc2n”
```

Если "нехороший человек" получит доступ к этому файлу, вас ожидают большие проблемы!

Хранение паролей (2)

- Unix (1974): шифрует пароли, используя в качестве ключа их самих

```
Bob: 14: S6Uu0cYDVdTAk
David: 15: J2ZI4ndBL6X.M
Mary: 16: VW2bqvTalBJKg
```

- Пароль пользователя "David" – "allison", он шифруется с ключом "allison" и результат сохраняется в файле
- Система может проверять правильность пароля
- Если кто-то получит доступ к файлу, то он не сможет непосредственно вычислить пароль, но сможет организовать "*атаку по словарю*" или "*атаку перебором*"

Хранение паролей (3)

- Unix (1979): дополненные (salted) пароли

```
Bob: 14: T7Vs1dZEWeRcL: 45
David: 15: K3AJ5ocCM4ZM$: 392
Mary: 16: WX3crwUbmCKLf: 152
```

Шифрование производится после добавления к паролю некоторого числа. Такой подход противодействует атакам по заранее сформированным словарям зашифрованных паролей

Подбор пароля

- Допустим, используются 26 букв в словах из 7 символов
  - 8 миллиардов паролей (33 бита)
  - Перебор со скоростью 100,000 паролей в секунду взломает ваш пароль через 22 часа
    - Лучше выбирать трудоемкие алгоритмы шифрования
- Но большинство людей не используют пароли в виде произвольных последовательностей букв!
  - имя мамы/папы/друга/собаки
- "Атака по словарю" традиционно организовывается очень просто, но часто дает хорошие результаты

Усложним задачу подбора пароля

- Будем использовать буквы и цифры, увеличим длину пароля
  - 95 различных символов, длина – 14 символов
  - 10^{27} паролей = 91 бит
  - Перебор со скоростью 100,000 паролей в секунду угадает пароль через 10^{14} лет
- Потребуем регулярное изменение пароля
  - и запретим передавать пароли друг другу для временного использования, записывать их куда-либо и т.д.

Есть ли польза от длинных паролей?

- Люди не могут запоминать пароли, состоящие из 14 случайных символов
- Люди записывают такие трудные пароли
- Люди выбалтывают свои пароли кому-попало
- Пароли могут присутствовать на жестком диске
- Если заставлять людей периодически менять пароль, они могут выбирать очень простые пароли
  - “feb04” “mar04” “apr04”
- Какие пароли вы сейчас используете и как вы их получили?

Часто используемые пароли

0, 000000, 00000000, 007, 1, 110, 111, 111111, 11111111, 12, 121212, 123, 123123, 1234, 12345, 123456, 1234567, 12345678, 123456789, 1234qwer, 123abc, 123asd, 123qwe, 2002, 2003, 2600, 54321, 654321, 88888888, a, aaa, abc, abc123, abcd, Admin, admin, admin123, administrator, alpha, asdf, computer, database, enable, foobar, god, godblessyou, home, ihavenopass, Internet, Login, login, love, mypass, mypass123, mypc, mypc123, null, oracle, owner, pass, passwd, Password, password, pat, patrick, pc, pw, pw123, pwd, qwer, root, secret, server, sex, super, sybase, temp, temp123, test, test123, win, xp, xxx, yxcv, zxcv

Пример атаки через алгоритм проверки паролей

- Проверка пароля в VMS
  - алгоритм проверки пароля выглядит следующим образом

```
for (I=0; I<password.length( ); I++) {
    if password[I] != supplied_password[I]
        return false;
}
return true;
```

  - видите ли вы проблему?
    - подсказка: подумайте о виртуальной памяти…
    - вторая подсказка: подумайте о страничных сбоях…
    - последняя подсказка: кто определяет, где в памяти размещается `supplied_password[]`?

"Вынюхивание" паролей Sniffing passwords

- Невероятно, но всего несколько лет назад общепринятой практикой являлась передача паролей по сети в незашифрованном виде!
  - включая беспроводные сети, где перехват чужих пакетов выполняется тривиально!

Авторизация (Authorization)

- Откуда система знает, что вам позволено делать?
  - идеологически, имеется матрица авторизации
    - объекты – сущности, к которым выполняется доступ
    - субъекты – сущности, которые выполняют доступ (пользователи или программы)

	Ivanov	Petrov	Sidorov
/etc	Read	Read	Read Write
/homes	Read Write	Read Write	Read Write
/usr	None	None	Read

Авторизация (2)

- На практике используются реализации через
  - Списки прав доступа (Access Control Lists, ACLs)
  - Перечни возможностей (Capabilities)
- (Вспоминаем обсуждение на лекции по файловым системам)
- Большинство систем используют оба подхода в различных сочетаниях