7jdg's blog

感谢CCAV,感谢MTV,感谢黄袍怪,感谢桂兰

John the Ripper中文说明

admin 发表于 . 工具箱子

==========
甚么是 John the Ripper?
==========
John the Ripper 是一个 UNIX 密码破解工具程式, 可以使用的作业系统环境有 UNIX (在 Linux x86, FreeBSD x86, Solaris 2.x SPARC, OSF/1 Alpha 都测试过了), DOS, WinNT/Win95. 当然, 在你使用 John 之前, 最好已经使用过其它的 UNIX 密码破解工具, 这样你才可以很容易的了解 John 的运作方式与操作方法!
==========
1.3 版的新增功能
==========
- MD5 编码的密码档案解码支援; (以前的解码都是只针对 DES)
- SPARC V8 组合语言版本;
- 修正了许多前版的 Bugs. (以前的版本有那么多的 Bug 吗??)
======
概观
======
作者把 John 尽量设计成强大, 而且快速的破解工具. 在同一个程式里面包含了几种的破解方式, 而且你可以完全的依照你的需求来自订 John 破解的方式(我觉得这点很强, 甚至可以用内建的 C Script 来自定不同个案的破解方式, 容後说明). 而且 John 也可以在不同的系统平台上使用, 让你可以在不同的电脑上破解密码, 在范例中的接续破解, 可以让你在破解中断之後, 在不同的作业平台上接下去破解.
John 的 crypt() 函式在快速作业的模式之下进行了最佳化, 这可以让 John 在破解的时候跑得比其它的破解工具快, 这个函式同时套用了组合语言及可转平台式 C 语言两个语言所写出来的程式码.
John 支援了以下几种的破解方式:
- 有规则及不规则的字典档破解模式;
- "Single Crack", 用最简单的资讯来进行破解的工作, 速度最快.
- 增强破解模式(我们称暴力法), 尝试所有可能的字元组合;
- 外部破解模式, 让你可以定义你的破解模式.
==========
如何安装
==========
John 所提供的是压缩过的程式, 你需要建一个目录, 然後把这些档案都拷贝到那个目录底下, 然後将你需要的档案解压缩(当然你必需要有解缩的程式).在你使用这个压缩档以前你可能须要先下达一个 'chmod +x john' 指令.
如果你要 Compile 原始程式的话, 只要进入你解压这些档案的目录, 然後在系统提示符号下输入 'make', 你就会在萤幕上看到一份系统支援的列表. 选择你所使用的系即可. 如果你的系统没有在该列表上, 就请你输入 'make generic'. 请你确定你使用的是 GCC 且 GNU make (也许你得输入包含路径的执行指令, 例如 '/bin/make',如果你

« 2012年06月 »

日	一	二	三	四	五	六
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

分类

网站安全 (10)
平凡生活 (16)
工具箱子 (20)
闲言碎语 (19)
无线安全 (2)
PHP相关 (2)
源码相关 (2)

的系统没有设定路径的话).

==========

如何使用

==========

作者已经尽量把 John 的操作方式设计得跟 Cracker Jack 相同, 所以如果你先前曾经使用过 Crack Jack, 应该很快的就能够使用 John. 总之, 有很多 Cracker Jack 的功能, 都可以在 John 上面延用, 操作方法也是相同的.

要使用 John the Ripper, 你必需要有一些密码档, 破解好的密码将会显示在萤幕上, 并且会储存在一个叫作 ~/john.pot 的档案中 ('~' 所代表的是 John 的相同目录, 也就是你放置 John 的目录).这个档案还有另一个功能, 就是让 John 不要重覆跑你已经破解过的帐号, 如果你使用 John 再跑一次曾经跑过的密码档, 相同的帐号不会再跑过一次, 如果你想要看你已经破解过的密码, 你可以使用 '-show' 这个功能.

在破解的时候, 你可以按下 Enter 来观看目前破解的状态,或是按下 Ctrl+C 来中断目前的破解工作, 这样程式会自动将目前破解到的位置, 储存在一个档案之中 (~/restore 为内定的档名), 另外, 如果你是按了两下 Ctrl+C 来中断的话, John 就会直接中断而不会将目前破解进度储存了. 这个破解进度档每十分钟也会自动的储存, 以必免你的机器在破解中当机而功亏一溃. (这是个不错的设计, 当然 Jack 里也有这项功能)

命令列的功能选项

你可以在执行 John 的命令列後加上下面这些选项 (所有的选项都不须区分大小写):

-pwfile:<档名>[,..] 指定密码档档名 (可以使用万用字元)

这个选项用来指定你所要破解的密码档 (通常在命令列上要使用的档案名称, 不能够使用以 '-'这个字作为开头的档案名称, 因为 '-'已经被用来当作命令列的辨识字元了).

-wordfile:<档名> -stdin 字典档破解模式, 由字典档读取来破解, 或是 stdin

这个选项用来开启 John 为字典档破解模式.

-rules 打开规则式字典档破解模式

开启规则式 (就像使用 Alec Muffett 的方式破解). 至於规则的定义就是使用放在 ~/john.ini 中 [List.Rules:Wordlist] 所定义的规则.

-incremental[:<mode>] 增强模式 [使用 john.ini 中定义的模式]

开启增强模式, 使用你在 ~/john.ini 中所定义的模式来破解 (在 [Incremental:<mode>] 段落中你可以自行定义, <mode> 就是在这里你所要指定的 <mode> 名称, [Incremental:All] 为内定). 这一段可以让你指定很多不同的方式, 我们会在後面再说明.

-single Single Crack模式

开启 "Single Crack" 模式, 使用你在 [List.Rules:Single] 中所定义的规则.

-external:<mode> 外部破解模式, 使用你在 john.ini 定义的 <mode>

开启外部破解模式, 将会自动启用你在 ~/john.ini 的 [List.External:<mode>] 段中所定义的自订破解功能.

-restore[:<档名>] 回复上一次的破解工作 [经由 <档名>]

继续上次中断的破解工作, 由特定的档案 (通常是 ~/resotre) 读取上一次破解的时候中断的位置, 然後接下去破解. 这可用在中断破解之後的接续破解.

-makechars:<档名> 制作字元表, 你所指定的档名若存在会被覆写

产生一个内含字元表的档案, 他会以~/john.pot (找到的密码) 为基础来产生. 这个产生出来的档案你可以用在增强破解模式上. 除非你指定了其它的密码档, 不然系统将会自动抓~/john.pot 内容来产生字元表. 你也可以同时使用 filter() 函式来进行过滤.

-show 显示已破解的密码

显示已经破解完成的密码. 你必需同时指令你要显示的密码档是哪一个才行.

-test 执行速度测试

这个功能可以用来测试你所使用的电脑的破解速度, 它会显示一个速度的比较表来告诉你你的电脑在不同的环境 (要破解的帐号多寡) 所产生的不同效率, 让你可以作好最佳的调整.

在你不熟悉的密码档破解的时候:xform1() 跟 xform2() 是真实的编码函式, 它呼叫了每一个key/salt (一对一对的呼叫), 当你每个字呼叫 setkey(), 表示说 xform1() 或 xform2() (要看你是用哪一种破解模式才能决定是哪一个) 在足够的 salts 载入之後,

是唯一会被影响到破解速度的函式,总而言之, setkey() 可以决定你在使用这个程式的时候的速度, 当你使用的是以 MD5 编码的密码档, md5crypt() 就会取代所有的其它函式.

-users:<login|uid>[,..] 只破解这一个使用者(或群组)

让你能够过滤只破解某些特定的人, 你也可以用在 '-show' 这个功能. (通常你会只找 root, 但是我建议你能够找 uid =0 的人, 因为 uid=0 也就表示他具有 root 同等的权限)

-shells:[!]<shell>[,..] 只针对某些使用你指定的 shell(s) 的使用者破解

这个选项可以用在 破解/显示 使用的是你所指定的 shell 的帐号, 或者是不要显示/破解这些帐号(也就是说可以用来过滤) 在 Shell 名称前加上 '!' 就表示 Not 了. 你可以在的 Shell 名称之前指定绝对路径, 或者也可以不指定, 当然'-shells:csh'就会包含'/bin/csh'跟'/usr/bin/csh' ,如果你指定的是 '-shells:/bin/csh' 将只会包含 '/bin/csh' 这个 Shell 名称.

-salts:[!]<count> 只破解 salts 大於 <count> 的帐号

这个功能通常使用在让你得到最高的系统效能来破解密码上. 如同范例, 你可以只破解某些的 salts 使用'-salts:2'这个选项, 会使你的系统运作快一点, 然後再破解 rest 使用 '-salts:!2' 这个选项 (配合著使用). 总共需要的破解时间大约是相同的, 不过你得到已破解帐号的速度会快一些, 而且系统不需要休息.

-lamesalts 设定 salts 中密码所使用的 cleartext

当你不知道你在作甚么的时候, 不要使用这个选项.

-timeout:<time> 设定最长的破解时间 <time> 分钟

当你设定的时间到达时, John 将会自动中断目前的破解工作.

-list 列出每一个字

在标准输出设备上 (通常是萤幕) 列出已经破解出的每一个字(密码). 这个功能可以用在检查你的自订破解模式是否正确的完成.

-beep -quiet 当发现密码 是/否 要发出声响

你可以在 ~/john.ini 中指定你所要的预设值.

-noname -nohash 不要使用记忆体来储存 login name 跟其它的资料

在你没有足够的记忆体来跑 John 的时候, 你可能需要这个选项. '-noname' 不能使用在 "Single Crack" 模式之下. 因为 Login name 必须在此模式下使用!

-des -md5 强制使用 DES 或 MD5 模式

这个选项让你自行决定密码加密的方式(不用自动判断). 你要注意的是 John 并不能在同一个破解工作中同时跑两种不同的密码加密方式, 如果你的密码档是两种的, 你就要分开来跑!

附加的工具程式

你也许会使用到 John 所附的一些工具程式:

xtract [source] [> <target>]

由文字档中取出字典档, 让你可以把字典档使用在破解上. 重覆的字不会自动的移除, 你须要再使用 'sort -u' 来作输出 (这里的指令是用在 UNIX 上的).

unshadow <passwd> <shadow> [> <target>]

组合 passwd 跟 shadow 档案 (当你已经得到这两个档的时候) 让真正的密码档得以还原, 这样你就可以在 John 上面使用了. 你也许会需要这个功能, 如果你只有 shadow 过的密码档案, GECOS 资讯不能在 "Single Crack" 模式下使用, 而且也不能够使用 '-shells' 这个选项的功能.

==========

破解的模式

==========

这里的破解模式的叙述通常都很简短, 而且只函括了一些基本的东西, 你可以看一下 "使用者自订" 这一个小节, 来获得更多的资讯.

字典档模式

这是 John 所支援的破解模式中最简单的一种, 你须要的只是指定一个字典档 (文字档

案, 每行一个英文单字)及一个或一些密码档, 你可以使用规则化的方式(用来修正每个读入的单字) 来让这些规则自动的套用在每个读入的单字中.
字典档中的字不能够有所重覆, 因为 John 并不会删除重覆及将字典档排序, 所以如果有重覆的化会占用过多的记忆体, 最好能将一些常用的字典放在你字典档的开头, 当然你最好能够按字母的排列方式来排序你的字典档,(如果每一个字跟先前的那个字的差别小一点的话, John会跑得稍微快一点点, 这也是为甚么要排序的原因了, 这个状况在你破解的密码档小的时候特别明显). 译者按:没想到连这个都测试, 真是想得周到呀!!
另外, 你不须要担心每一个字元的长度超过了八个字元, John会自动比对这些字, 如果前八个字元一样, John会自动的处理这种情况, 而且只试同一个密码一次, 前提是这个字必须是接在前一个字後面, 这也就是为甚么要排序字典档的原因了. 你最好是不要擅作主张的把超过八字元的字切成八个字元, 因为在规则化的破解模式之下, 後面的字可能还会有其它的用处.
译者按: 因为 DES编码方法在以前的系统上有八字元的限制, 所以超过八字元的密码亦视同八字元来编码, 超过的部分则会被系统自动切除.
"Single Crack" 模式

这是你刚才使破解密码时需使用的, 它将会试著使用 login/GECOS资讯来当做密码, 这些资讯通常适用於该资讯来源的帐号(在帐号上是相同的 salt, 所以几乎不需要额外的破解时间 (很快的意思), "Single Crack" 婆式比字典档模式还要快很多, 它也让你可以使用许多种规则 (这些规则通常在使用此一模式时都会开启) 在合理的时间之内. 当然, 这个模式只会得到使用基本资料来当作密码的使用者资料.
值得注意的是, 当你用这个模式在同一个时间跑很多的密码档时, 通常会比你将这些档案分开来跑, 还要能够更快的得到已经破解的密码.
不要跑尽所有的规则也许是一个好方法, 但是节省时间, 然後用其它的破解模式来跑, 这样的话规则型态会以号码来排列破解的密码.
CJack 的使用者必需要注意到 John 的 "Single Crack" 模式是完全不一样(更好)的. 它不需要指定一个密码档, 因为密码档制造的规则型态与程式码已经内建在 John 里头了.
增强模式

这是功能最强大的破解模式, 它可是试所有可能的字元组合来当作密码, 但是这个模式是假设在破解中不会被中断, 所以当你在使用长字串组合时, 最好不要直中断执行 (事实上你还是可以在执行时中断, 如果你设定了密码字串长度的限制, 或者是让这个模式跑一些字元数少一点的字元集), 然後你将可以早一点中断它的执行.
这就是为甚么这个么式须要指定字元频率表(character frequency tables, 字元集)--在有限的时间内去得到所有可能的密码.
要使用这个破解模式, 你须要指定及定义破解模式的参数,(包含密码长度的限制还有字元集). 这些参数必须写入到~/john.ini 中的[Incremental:<mode>] 这一段内,<mode> 可以任意命名 (就是你必须要在执行 John 时在命令列指定的名称).你可以使用一个重新定义的增强模式, 或是定义一个自订的.
若你曾经定义过了一次, 日後当你要再使用同一种增强模式的时候只要指定增强模式要套用的选项, 相同的选项及密码档的套用就会自动的回复.
扩充模式

在使用John的时候你可以定义一些扩充的破解模式.只要在~/john.ini 的[List.External:<mode>] 一节中指定就可以了,<mode>就是你所指定的模式名称. 这一段中必须要包含一些 John 尝试要产生的字典的功能. 这些功能的撰写方式就是 C语言的子集,它会自动的在 John 执行前编译 (只有在你开启 John 时使用这个模式才会被编译).
==========
如何自订
==========
John the Ripper 的破解行为及方式可以编辑~/john.ini 来自订, 你可以在执行 John 的时候在命令列指定破解方式, 其它 John 的选项不会受到命令列的影响, 为增强模式

定义一些参数, 为 Single Crack 模式定义一些规则, 或者你也可以定义一个新的破解方式.
这个设定档 (~/john.ini) 是由许多小段组合而成的. 每一个小段的起始是由括弧括起来, 里面所写的是这一段的名称. 每一个段中内含了指定的一些变数来组成可变动的变数, 或由一些特别的项目来指定段的型态(像这样的段启始字元为 'list.'). 段跟可动变数的大小写对程式来讲是无所谓的. 如果在行前加上了 '#' 或是 ';' 字元, 就表示这一行将会被程式所略过不予执行, 你可以用来加入一些注解.
一般选项

预设一些命令列选项可以放在 [Defaults] 这一段中. 你可以定义下面这些值:
Wordfile 设定你的字典档档名, 这会自动虚拟成你正使用的破解模式是字典档模式, 你不需要再加上 '-wordfile' 这个选项.
Timeout 设定中断的时间, 单位为分钟.如果使用这个选项的话, 所有的破解模式都会接受时间到就自动停止 (建议不要指定这项, 在命令列指定就好了).
Beep 当系统找到密码时会发出 '哔'声, 或是在一些要问你 (Yes/No) 的时候也会令你的电脑发出声响来提醒你. 另一组相反的选项为 '-quiet', 它不会令你的电脑发出声响, 所以最好先指定 Beep, 当你需要让电脑安静时在命令列使用 -quiet 即可.
一些其它的选项可以先定义在 [Options] 这一段中:
Realtime 设定已经经过的时间为 D:HH:MM:SS 来取代原先用秒数来计算的方式 (跟 CJack 相同).
Percent 设定显示百分比指示器.
增强破解模式的参数

要定义一个增强模式的参数, 你需要先建立一个叫作 [Incremental:<mode>] 的段, 这里的 <mode> 你可以自订这一段的名称. 在 John 里面有一些已经设定好的预设增强模式的参数定义, 你可以用这些定义来当作你自订参数的范本. 下面这些是支援的参数:
CharCount 让你限制不同字元使用时的字数限制, 让 John启始时可以早一点试跑长字串的密码,也可以用来设定字元集的特别长度当使用一个外部的字元集档案其字元数量小於你在 CharCount 所设的字元数时.内定值(当此值未定义时的预设值) 所有定义的字元都会被使用.
MinLen 最小的密码字串长度, 字元数 (1 为内定值).
MaxLen 最大的密码字串长度, 字元数 (8 为内定值).
Wordlike 设定为 'Y' 可以开启一个简单的字典过滤器 (一排字有多於一个的母音, 或多於两个个没有母音的一排字, 都会被过滤掉).
File 外部字元集档名 (档案是由你所设定的路径读入, 内定为 ~ 目录) 设定这个参数会取消你在设定档中内定的字元集.
CharsetNM (N 与 M 为数字格式, 1 <= N <= 8, 1 <= M <= N)为一个密码档定义一个字元集长度为 N, 字元指标为 M.字元的顺序是很重要的, 比较频繁的字元将会较先被取代, 字元集不需要是相同的大小.
字典档的规则

定义给字典档及 "single crack" 模式的节段是放在一个叫做 [List.Rules:Wordlist] 跟 [List.Rules:Single] 之中. 作者使用一种叫作扩充破解 (by Alec Muffett) 的语法, 或许有许多人对於这种破解模式已经很熟悉了. 作者加入了更多重要的的规则来使它更为优秀, 让它能够使用同一个原始来源产生更多更复杂的方式.
当在定义规则的时候, 在每一行中只需要安排一种规则 (之前可能需要下达一些指令). 每一种规则是由一个或多个指令组成. 下面这些指令是 John 所支援的 (大部分的说明是由 Crack 程式的 dicts.rules 所转录下来的, 但是程式码是作者自己重写的, 而且比 Crack 还要快):
一般常用指令:
: no-op - 不要在输入的字之後作任何动作
<n 拒绝输入的字长度是 < n 的字元, n = 0-9

>n 拒绝输入的字长度是 > n 的字元, n = 0-9
^x 将 'x' 由每个输入的字移出
$y 在每个输入的字後加上 'y' 这个字元
l 强制小写字
u 强制大写字
c 强制第一个字大写(其馀小写)
r 把字元排列次序巔倒: "Fred" -> "derF"
d 重覆每个字: "Fred" -> "FredFred"
f 镜射形态字: "Fred" -> "FredderF"
p 尝试每个字元的大小写变化 (abc, Abc, aBc, abC....)
onx 如果字元不满几个字, 就把它加到几个字, n (由 0 开始) 每个未足字元都以 'x' 字取代.
inx 在指定字元数插入 'x' 的字元, 字元数 n (由 0 开始) 後面的字将会依插入的字元多寡
而向右边作位移
nb: 如果指定的 > 字串总长度(input), 字元 'x' 将会用增加的方式加上
xnm 由字串中抽离某些字, 字元数 n (由 0 开始) 而且会抽离 m 个字元.
会使用在字元层级的指令:
sxy 取代 (交换), 将字串中所有的字元 'x' 取代为 'y'
s?cy 用 'y' 来取代所有字元为 'c' 者
@x 由字串中清除所有字元为 'x' 者
@?c 由字串中清除所有字元为 'c' 者
!y 拒绝执行字串中包含有 'y' 者
!?c 拒绝执行字串中包含在 class 'c' 中有定义的
/x 拒绝所有字串中未包含字元 'x' 者
/?c 拒绝除了字中包含 class 'c' 以外的所有字
=nx 拒绝除了 class 等於 'x' 以外的所有
=n?c 拒绝除了 class 'c' 以外的所有字
nb: 所有的字串都由位置(Position) 0 开始计算
用在上面所叙述的字元 class 的指令:
?? 相同於 '?'
?v 相同於 母音: "aeiouAEIOU"
?c 相同於 子音: "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"
?w 相同於 空白符号: " \t"
?p 相同於 标点符号: ".,:;'\"?!`"
?s 相同於 一般符号: "$%^&*()-_+=|\\<>[]{}#@/~"
?l 相同於 小写字母 ('a' to 'z')
?u 相同於 大写字母 ('A' to 'Z')
?d 相同於 数字 ('0' to '9')
?a 相同於 字母 ('a' to 'z' and 'A' to 'Z')
?x 相同於 字母及符号 ('a' to 'z', 'A' to 'Z' and '0' to '9')
用来表示相反 class 的字元就用大写的字母来表示,
例如: 用 ?d 来表示 '数字(DIGITS)', ?D 就表示 '非数字(NON-DIGITS)' 依此类推.

上面所叙述的只令是跟 Crack v4.1 相同的, 下面的则是 John 中所新增的 (作者说明
这些指令也许不是很有用处, 且大部分的东西在 Crack v4.1 也可以作得出来):
{ 字串左移: "jsmith" -> "smithj", etc
} 字串右移: "smithj" -> "jsmith", etc
Dn 删除位置 n 的字元 (由 0 开始) 及把该字元後的字左移
P "crack" -> "cracked", etc (过去式, 只针对小写)
G "crack" -> "cracking", etc (现在进行式, 只针对小写)
~i 由键盘方式转换大小写(加 Shift 键): "Crack96" -> "cRACK(^", etc
~I 转换大小写: "Crack96" -> "cRACK96", etc
~v 将母音转小写: "Crack96" -> "CRaCK96", etc
~> 由键盘方式将所有字元右移: "Crack96" -> "Vtsvl07", etc

~< 由键盘方式将所有字元左移: "Crack96" -> "Xeaxj85", etc
特别针对 "single crack" 模式的指令有双字串支援, 控制这些指令要套用指令时需要用到下列的命令:
1 只有第一个字
2 只有第二个字
+ 包含两个字 (必需只用在 '1' 或 '2' 之後, 也就是 1+2 或 2+1)
如果你在规则设定中使用了上述的指令, 将只会执行双字串(全名, 由GECOS资料得来), 而放弃只有单一字串的字.
'+' 会假定在规则结尾使用这些命令的时候, 除非你用手动指定. 例如, '1l2u'会转换第一个字为小写, 第二个字为大写, 而且会把两个字相连. 在使用 '+'时可能要应用一些其它的指令: 在你分别应用一些指令来执行的时候 '1l2u+r' 会用相反顺序来相连这两个字, .
[Crack v5.0 在作者更新 John 的时候还没有发行, 所以作者使用一些自己的方法来扩充规则语法. 事实上, 新版 Crack v5.0 的规则看起来像是多馀的, 或是跟作者已加在 John 里的功能是相同的([== D0,] == rD0r, C == c~I, t == ~I, (x == =0x,)x == r=0xr, 'n == x0n). 不一样的规则为 %nx, or %n?c (拒绝执行,除非该字串最少包含了有 n 个 'x' 字元, 或是 n 个 class 'c' 里定义的字元).无论如何, 作者已经把所有的功能都加在 John 里面了, 基於相容性的理由. 请确定字首 '[' 及 ']' 加上了 '\'如果你有使用到的话, 因为他们已经用来控制其它的功能了.]
如果一个规则 (命令列的一行) 没有改变字, 那一个字将会被排除而不执行, 除非整个规则包含了冒号 ':', 假设你加入了冒号 ':' 在你的规则定义中.
前置处理是用在组合类似的规则进入同一个来源. 如同范例, 如果你希望让 John 尝试小写密码并加上数字, 你可以为每个数字定义一个规则, 总共有十个. 现在想像一下加入两位数字(号码)--这样设定档会便得很大且会很难看.
用前置处理, 你可以很简单的就作到这些事情: 很简单的写一个来源行, 包含这些规则的共用部分, 在括号中写出你要放进不同规则中的字元 (你必需要使用正规的表示法). 然後前置处理器就会在 John 启始的时候产生你所要的规则, 像上面的范例一样, 来源行会是 'l$[0-9]' (小写字串, 并且加上数字) 及 'l$[0-9]$[0-9]' (小写字串并加上两位数字). 这些来源行会分别的加到 10 及 100 规则. 总之, 前置处理的命令处理顺序是由又至左, 字元的处理顺序则是由左至右, 在加入两位数字这样的例子中, 会得到正常的顺序. 注意我在这些范例中只用到字元范围的形态 (由 A到Z 等的为范围形态), 但是你可以用字元列表来组合他们, 像'[aeiou]'会使用母音, 还有'[aeiou0-9]' 会使用母音跟数字.
有一些控制字元在规则里 ('[' 括号用来启始一个字元列表, '-' 表示一个包含在内的范围, 这一类的). 如果你希望把某一行放在设定档里面, 但是又不想使用他们的话,你可以在前面加上 '\' 符号, 那会使 John忽略这一行不去执行. 还有, 如果你需要开始一个前置处理表列在开始的一行中, 你将会用到 ':' 符号, 不然的话 John会认为它是一个新节段的开始.

定义扩充模式

要定义一种扩充模式, 你需要建立一个节段叫作 [List.External:<mode>], <mode> 就是你自己对这个模式定义的名称. 这一个节段中必需要包含一些使用 C语言子集所写的函数, 当你在命令列使用这个模式的时候, John会编译及使用这些程式, 编译程式会产生虚拟机械码, 比任何直译器或转换程机器可执行码来得好用 (现在只完成了 x86硬体可使用的部分).
下面这些功能是一般会在 John 中使用到的:
init() 在启始时呼叫, 会初始全域变数
filter() 在试每一个字串时呼叫, 可以将一些字串过滤掉
generate() 产生字串时呼叫, 当没有使用其它的破解模式时
restore() 回复一个中断的工作时呼叫
这些函数的型别都是 'void', 不需要引数(参数), 使用到了全域变数 'word' (已定义为 'int word[16]'), 除了 init() 之外, 它在 'word'初始之前已经被呼叫了.'word'

变数包含了要试的字串(ASCII),filter()可以改变它, 或是0以外的, 可以用'word[0]'
来保留它. generate() 不能假设任何特殊值的 'word'当它被呼叫的时候, 但是可以将
它放在下一个将要试的字串, 或是0以外 'word[0]' 当破解工作结束的时候 (这将会引
起 John 结束执行). restore() 必需设定全域变数来继续执行由支援的 'word' 字串.
你可以单独使用任意一个扩充模式, 或跟其它某些模式一起使用, 在这个情况之下只有
init() 以及 filter() 会被使用到 (而且只有 filter() 是必需要的). 使用扩充模式
的过滤器跟其它的破解模式及 '-makechars'这个命令是相容的.
我们建议你不要使用filter(), 最少在使用扩充模式於你自己的 generate() 时不要过
滤太多的字, 最好的方法是改一下 generate() 使它不要产生会被过滤掉的字串.
就像作者在上面所提到过的, 编译器支援了 C 语言的子集. John 是一个 cracker, 不
是一个编译器, 所以我不认为它需要其它的东西. 这里有一个列表, 列举出 John 跟 C
编译器功能上的差别:
- 只支援标准的功能, 你不能自行定义一个自己的;
- 只支援 'while' 的回圈;
- 只支援 'int' 与 'void' 资料型别;
- 只支援单一十进位的阵列;
- 不支援结构化
- 不支援指标 (没错, 你有更多的阵列了呀);
- 大概还有些别的吧...
一些支援的功能与 C 语言不同的地方:
- 阵列名称单独参考到它的第一个基本元件而非这个阵列的位址(以 0 作启始);
- '++' 跟 '--'运算元在表示式计算的时候执行, 而不是 pre/post-calculated; 这会
在 C 的很多例子中传回相同的结果 (像在 C 语言跟 John 的编译器中 'i = j++;'
都等於 'i = j; j = j + 1;'但是假如这些变数被应用不只一次的话, 传回的结果会
跟 C不同 (如 'i = j++ - j++;' 在John中就等於 'i = j - (j + 1); j = j + 2;'
但是在 C 语言里却变成是 'i = j - j; j = j + 2;');
- 我希望没有别的地方是不同的了...
无论如何, 强力的 C 语言语法结构 (所有整数运算元),'if'/'else' 跟 'while' 仍然
可以使用. 这对所有的小程式来讲应该是足够的了. 你可以看看在设定档支援的范例档
中的扩充模式范例.
==========
使用范例
==========
这些范例可以让你 解使用 John 可以帮你作哪些事, 也许不够明白的来表示该如何来
使用, 我只能试著来回答一些问题:
命令列
----
1. 假设你刚得到一个密码档,'passwd.1', 且你想要试著破解它, 你可以使用 Singel
Crack" 模式:
john -single passwd.1
或者,你也可以使用简写(John在很多选项都有提供简写的方式,让你很快的能够完成输入)
john -si passwd.1
如果你有很多的档案要破解, 最好的方式就是一次读进来:
john -single passwd.1 passwd.2
或者你也可以这样:
john -single passwd.*
2. 现在, 当你已经破解了一些密码, 这些已破的密码将会存在~/john.pot 这个档案里
你可以浏览一下你所破解的密码:
john -show passwd.1
如果这个列表超出了萤幕(解出了很多密码??), 你可以使用下面这个输出方式:
john -show passwd.1 | more
现在, 你可能会得到一些错误讯息告诉你有许多的帐号的 shell已经被取消掉了, 你可
以让 John 修改这些字串 (假设 shell 名称为 '/etc/expired'):

john -show -shells:!/etc/expired passwd.1
或是简写, 但是会跟 '/any/path/expired'有相同的效果:
john -show -shells:!expired passwd.1
或者, 你也想要修改其它的 shell 字串, 如 '/etc/newuser':
john -show -shells:!expired,!newuser passwd.1
检查看看有没有 root (uid 0) 帐号已经破解成功了:
john -show -users:0 passwd.1
或者, 检查所有密码档中, 已破解的 root (uid 0) 帐号:
john -show -users:0 passwd.*
只显示 root (login 'root') 帐号:
john -show -users:root passwd.1
3. 当你使用 "Single Crack"模式, 破解出来的帐号数目不是很多的时候, 你可以使用较具威力的破解模式, 例如字典档模式. 假设你的字典档名为 'words.lst':
john -w:words.lst passwd.1
或者, 把规则破解模式也打开 (会更慢, 但是更具威力):
john -w:words.lst -rules passwd.1
要只破解拥有完整 shell 使用权的帐号 (一般来说, '-shells' 跟 '-users' 这两个过滤就可以完成你想要作的工作了, 在其它的破解模式也是一样.
john -w:words.lst -rules -shells:sh,csh,tcsh,bash passwd.1
就跟其它的破解模式一样, 你可以更快的破解一些档案, 一次下达指令:
john -w:words.lst -rules passwd.*
可以只破解某些帐号. 像下面这个命令会试著破解具有 root (uid 0) 权限的帐号:
john -w:words.lst -rules -users:0 passwd.*
然而, 我不建议你只破解 root 的密码, 因为那通常会比用系统安全漏洞来获取 root 的权限花费更长的时间(通常不是在合理的时间内可以作到的), 如果你是用来试著破解你自己主机上的密码, 想要确定这些密码不会被破解的话, 最好是选一个好一点的root 密码, 然後只破解其它的.
有时把你的密码档分开两部分并且分别进行破解是有用的, 就像:
john -w:words.lst -rules -salts:2 passwd.*
john -w:words.lst -rules -salts:!2 passwd.*
这会使 John 在试两个或更多的帐号时动作快一点, 然後再试其它的总共需要的破解时间将会差不多, 但是你会更容易的得到一些破解的帐号, 而且可能也不需要其它的. 还有, 你可能想要用一个小一点的字典档来试所有的帐号, 只有用这个方法你可以试得快一点 (用 '-salts:2') 在一个大型的密码档上. 通常这是在使用 '-salts'大於 2 (有时甚至高於 1000 都还可以执行), 为你的个别状况进行调整吧.
注意你定义的字典档规则第一行中包含了 ':' (表示 '试所有包含在列表中的字'), 如果你已经执行了一个字典档破解模式而没有使用规则的话, 也确定你用相同的字典档加上规则来跑, 这一点要特别注意!!
4. John 里最强的破解模式是增强模式, 你可以试著用这个指令跑跑看:
john -i passwd.1
这个指令会使用内定的增强模式的参数, 定义在 ~/john.ini's [Incremental:All] 这一个节段中. 在设定档中支援了这些参数使用所有 95 个字元集, 而且试所有长度的密码, 从 1 到 8 个字元. 不要预期这个模式的破解会在合理的时间内结束 (除非所有的密码都设得很容易破, 而且很快的被破解掉了).
在很多的情况之下, 当你破解一些简单的密码时, 使用其它定义好的增强模式会比较快一些, 由限制字元集来著手. 下面的指令只会试 26 个字元组合排列方式, 由 1 到 8 个字元来算, 它将会尝试由 'a' 到 'zzzzzzzz'的所有字:
john -i:alpha passwd.1
相同的, 你可以配合著增强模式只破解 root 帐号 及使用一些其它 John的功能, 这个指令会试著破解所有的 root (uid 0) 帐号在所有的密码档中, 而且只有在这些产生的相同 salts, 所以你得到最少两倍的效率 -- 如果你有很多个密码档的话 (像 1000 个密码档, 命名为 '*.pwd'), 否则就是没有 root 在相同的 salts:
john -i -users:0 -salts:2 *.pwd

5. 如果你得到了一个密码档, 而且已经有很多个帐号已经破解了 (但你需要更多), 而且这个密码档的密码设定是相当罕见的,你可能会想要产生一个新的字元集档案, 以该密码档为基础的字元集:
john -makechars:custom.chr passwd.1
然後把这个新的档案用在增强模式中.
如果你由同一个国家得到很多个密码档的话, 也许可以把他们一起用来产生字元集档案这样你可以用它来帮你破解出更多的密码, 当然这个自元集日後也可以用在同一个国家所得到的密码档上:
john -makechars:custom.chr passwd.1 passwd.2
<把你的 custom 增强模式定义在 ~/john.ini 中>
john -i:custom passwd.3
上面这个范例中, 我门假设 'passwd.1' 跟 'passwd.2' 这两个密码档是来自同一个国家, 而且你已经拥有很多破解过的密码了, 而 'passwd.3' 也是从相同的国家来的, 你现在正打算要破解它.
当你在产生一个新的字元集档案的时候, 你可以使用一些已经定义好的, 或自行定义的过滤器, 来产生一些简单的字串:
john -makechars:my_alpha.chr -external:filter_alpha passwd.1
如果你的~/john.pot 设定档已经很肥大的话 (或是你没有的字元集档案), 也许你会想要使用它来产生新的字元集档案:
john -makechars:all.chr
john -makechars:alpha.chr -external:filter_alpha
john -makechars:digits.chr -external:filter_digits
在上面的范例中, John 会覆写已经存在的字元集档(如果它们原先已经在你的目录中的话), 写入的内容就是你在~/john.pot (John 使用整个档案, 如果你没有指定任何密码档的话), 为了你的方便使用, 注意字串过滤的使用也定义在 ~/john.ini 之中.
设定档
---
1. 假设你认为你要破解的某些密码档中有很多的使用者他们所设定的密码都是以帐号名称再加上 '?!' 的话. 你只要新增一个 "Single Crack" 模式的规则, 把这一行放在你的设定档中:
[List.Rules:Single]
$?$!
提示: 如果你要将 John 原先所设定的预设值保留下来的话, 你可以简单的修改这个节段的名称, 把它改成 John 没有使用的名称, 然後再建立一个跟旧节段一样名称的节段, 但请注意新节段必需要把 'list.' 这个关键字移除 (不使用), 这样在执行的时候才不会出现错误.
相同的指令也能够套用在字典档破解规则上.
2. 如果你产生了一个自订的字元集档案(如上所述) 你也需要使用增强模式的参数定义一个~/john.ini 的节段. 最简单的情况之下看起来会像下面这样 ('Custom'所指的可以是其它的档案, 用你所喜欢的名称来命名):
[Incremental:Custom]
File = custom.chr
这会让 John 只使用你用该密码所制作出来的字元集, 如果你想要用所有 95 个字元的话, 你也需要加入这一行:
CharCount = 95
加入这一行会告诉 John 扩充你的字元集档, 如果 95个字元(ASCII codes 32 to 126)中的某些字元没有出现在你的字元集档中, 字元加入的次序为: a-z, A-Z, 1-9, 0, 及其它.
你也可以使用 CharCount 来限制 John 使用不同的字元数:
CharCount = 25
如果你在产生字元集档时没有使用任何的过滤器, 设定较低的 CharCount会剃除一些罕见的字元, 能够让 John 更容易尝试较复杂, 较长的密码
要让 John 只尝试某些长度的密码, 可以加入下面这几行:

MinLen = 6
MaxLen = 8
把 'MinLen' 设定的长一点, 就像上面的范例一样, 在机器有限制使用者密码长度的时候是很合理的 (然而, 注意 root 通常可以为使用者设定任何长度的密码而不受此限). 相反的, 如果你觉得密码应该不会是很长的, 你可能会想要把 'MaxLen'设定得小一点. 当只使用字母字元(alphabetical)的时候, 也许开启简单的内见过滤器会很有用, 如果很多密码设定得很简单:
[Incremental:Wordlike]
CharCount = 26
MinLen = 3
Wordlike = Yeah
File = alpha.chr
3. 当使用 John 在安装小於 4Mb 的机器上时, 你可能须要使用小一点的字元集, 在使用 '-makechars' 来产生的字元集, 需要很大的记忆体来扩充 (要快一点执行的话) 在你同时读进很多个密码档的时候也需要这样子作, 在没有足够的记忆体时, 或让 John 在破解很少的 salts 时稍微跑得快一点. 这都有可能会发生, 因为 John在每个字读入像上一次读入的字, 或只有某些部分的字有改变时速度会快一些.当使用了大的扩充字元集档的时候, 字串通常每一次的测试都是比在~/john.ini 中的小字元集大很多, 字串的差别会相差很大. 而且,大的字元集在扩充的时候花花费较多的时间. 然而, 你需要注意的就是利益/时间是呈正比的, 要得到更多的密码就要花更多的时间,用最好的顺序来跑大的扩充字集也是很重要的. 所以聪明的人会使用小一点的字元集, 如果他们没有别的选择的话, 或是你原本就是想要尝试所有可能的组合. 我故意把比较小的字元集由~/john.ini中拿掉, 这样你才能够在 'File=' 这一行中写上你会用到的字元集
4. 在其它特殊的情况之下, 你也许会使用自订的较小字元集, 如果你知道使用者常把他们的密码加上'1', 你可以用这样的范例:
[Incremental:Suffix1]
MinLen = 6
MaxLen = 6
Charset61 = abcdefghijklmnopqrstuvwxyz
Charset62 = abcdefghijklmnopqrstuvwxyz
Charset63 = abcdefghijklmnopqrstuvwxyz
Charset64 = abcdefghijklmnopqrstuvwxyz
Charset65 = abcdefghijklmnopqrstuvwxyz
Charset66 = 1
5. 你也可以用写一个额外的字串过滤器来达到跟上例相同的结果:
[List.External:Filter1]
void filter() {
int i;
i = 0;
while (word[i] && word[i] >= 'a' && word[i] <= 'z') i++;
if (word[i] != '1' || word[i + 1]) word = 0;
}
这个过滤器只会把有相同字元且结尾是 '1'的滤除. 你可以把它用在其它不同的破解模式中, 但是都会变得很慢, 因为大部分的字都会被滤掉. 最好是使用它来产生字元集然後再来使用, (如果你已经有很多密码已经破解了, 会跳过过滤器).
如果你在某些情况下无法使用频率表(没有找到规则), 你可以在额外破解模式中用相同的程式:
[List.External:Suffix1]
int len, current[9];
void init() {
int i;
current[len = 6] = 0; current[i = len - 1] = '1';

```
while (i--) current[i] = 'a';
}
void generate() {
int i;
i = len + 1;
while (i--) word[i] = current[i];
i = len - 2;
while (++current[i] > 'z')
if (i) current[i--] = 'a'; else current = -1;
}
void restore() {
int i;
i = len + 1;
while (i--) current[i] = word[i];
}
```

========

F.A.Q.

========

Q: 为甚么要命名为 "John"?

A: 为甚么不?

Q: 为甚么命名中有 "the Ripper" 这个字?

A: 那是 Lost Soul 的主意. 问他吧!

Q: John the Ripper 有比 Cracker Jack 还要好用吗?

A: 我觉得比较好. John 支援了所有 Cracker Jack 所提供的功能, 而且也多了很多新的功能. 还有, John 在 Pentium 上跑得 Jack 还快, 甚至在有些 486 机器上也比 Jack 快很多.

Q: John the Ripper 比 Crack 还好用吗??

A: 看你自己. John 是有比较快, 而且有一些 Crack 没有的功能. 但是 Crack 也是一个不错的程式.

Q: 为甚么 John 不能在我的旧 386 上跑得比较快?

A: John 是在 486 以上的机器作最佳化的. 如果要为 386 的机器作最佳化要花很多的时间. 如果你只有 386 机器的话, 你最好能在 InterNet 上找一台快一点的机器来跑 John. (对啦! 就是不出 386 版本了! 当然, 386 面临淘汰了嘛!)

Q: John 有针对 Pentium 进行最加化的版本吗?

A: John 已经针对 Pentium 级的机器进行过最佳化了!

Q: 我要怎么样测试 John 的 crypt() 函式是否工作正常呢?

A: John 在每次执行的时後都会自己测试一次, 你不需自己测试.

Q: 我要怎么样使用 John 的 "single crack" 模式? 他好像不能使用字典档.

A: 没错, John 跟 Cracker Jack 的 "single crack" 模式有很大的不同. RTFM.

Q: 为甚么你不把 "single crack" 改良得跟 Jack 一样好?

A: Jack 的 "single crack" 模式不是最好的, 它比 John 来要差. 我很高兴有人会提出这个问题... 也许是因为 Jack 的模式看起来比较复杂而令你有这样的错觉吧.

Q: 可不可以把 Jack 的 JPP.EXE, 改成可以在 Windows 95 上面跑?

A: 你不需要它了. 但是你大概真的须要再读一下这篇文章中 "使用者字订"这个章节, 有关於字点档规则的部分. 而且, 我认真的建议你不要在 Windows 95 上作任何的工作.

Q: 我要怎么看已经破解的密码? 在 CJack 中有 JACKPOT.EXE 这个档呀.

A: 在命令列上使用 '-show' 这个指令.

Q: 为甚么 John 不读入我的密码档呢? 它只显示 'Loaded 0 passwords'.

A: 你的密码档大概是 shadow 过的吧. 你需要抓到密码档跟 Shadow档, 把它们组合在一起, 然後再来使用 John. 当然, 如果你的密码档格式没有被 John 支援的话也会出现相同的讯息.

Q: 我要如何解开 shadow?

A: 你大概是说不须要 root 权限就拿到 shadow 档吧. 嗯,这里有一些小诡计, 但最好你还是需要 root. 很抱歉, 我不是在提这些事, 或是告诉你该如何 "hack" root. 这不是这个 FAQ 的原意.

Q: 为甚么 John 在增强模式不显示进度指示呢?

A: 你真的想要一直看著 0%吗? 如果你再问一次这个问题, 你大概需要再读一次这份文件了(增强模式方面的说明).

Q: 为甚么 John 显示的是无意义的 c/s 数值而不是秀出每秒的 crypt() 进度?

A: John 显示出来的数值表示每秒组合 (login 及 password) 而不是每秒crypt(). 如果你只想要试试编码的速度的话, 使用'-test'这个选项. 注意破解中所显示的 c/s 数值并不是无意义的-- 它表示你在个别密码档的实际破解速度, 而且可能可以在你用 '-salts' 功能调整速度的时候.

Q: 使用增强破解模式时, 我查觉到 c/s 值小於其它的破解模式很多, 甚至跟 John 1.0 版比起来也少了很多, 这是怎么回事呢?

A:你可能只有执行 John 几分钟吧. 新的增强模式每次在 John 切换到不同的密码长度时, 使用需要扩充的较大字元集. 这个长度转换需要花费一点时间, 所以才造成 John 每一秒尝试的密码组合会更少. 很幸运的, 这是只有在 John破解了一段时间, 切换到一个新的密码长度, 重新开始执行破解工作时才会有这种情形, 我想你没有必要在这个密码档上使用. 总之, 如果你不喜欢这个新的方式, 你可以不要用这么大的字元集 (把'file='这一行由 ~/john.ini 中移除) .

Q: John 有支援平行处理吗?

A:我有一个像你所讲的, 可以在网路上分开(逐步)处理的 Cracker 计画 (很快就会完成). John 并不能支援真正的平行处理, 但你仍然能够以自订字元集的方式设定每台机器跑不同的字元集, 你也可以在增强破解模式加上字串过滤来达到相同的效果.

Q: John 内定的字元集是怎么来的(在 ~/john.ini 跟 *.chr 档的) 它的基础来源?

A: 我参考了一个由世界上的不同机器超过六万五千个真正密码的列表, 我要谢谢这些使用者帮忙设定他们的密码.

Q: 我要到哪里取得字典档?

A: 你可以在 ftp://sable.ox.ac.uk/pub/wordlists. 找到一些.

Q: 我要如何跟作者取得连络?

A: 你可以在这个文件最後的地方看到.

==========

感 谢

==========

在发展 John 的时候, 我使用了其它破解同好的方法及建议:

- Crack by Alec Muffett --

字典档规则语法;

- Cracker Jack by Jackal --

使用者介面;

- Star Cracker by The SOrCErEr --

证明了一个很大的字元表也有执行的价值.

crypt() 编码函式所使用的是 Alec Muffett 写的, 跟 Crack v4.1 是同一个. 只有一些设定初始化函式没有改写, 其它的部分已经由我自己的想法改写了新的 crypt() 来使用 (事实上, 目前的这个编码函式使用了许多不同的运算方式).

特别感谢 Roman Rusakov 提供了 x86 组合语言版本的 crypt() 编码函式, 这个函式已经放在目前的版本中, 他的最佳化方式是最棒的!

DOS 版本是用 DJGPP v2 by DJ Delorie 及 GCC 2.7.2 (http://www.delorie.com) 编译的, DPMI 伺服程式是使用 Charles W Sandmann (sandmann@clio.rice.edu; 1206 Braelinn, Sugar Land, TX 77479), 原始程式在 ftp://ftp.simtel.net/pub/simtelnet/gnu/djgpp/v2misc/csdpmi3s.zip.

« 上一篇 | 下一篇 »

访客评论

lug #1
2009, October 16, 2:32 AM

有点老了....很多东西都不一样了

发表评论

名字 (必填,如果已是注册用户请先登陆):

E-mail (必填,不会被显示在前台,仅为方便联系):

网址 (选填):

评论内容 (必填):

点击获得Trackback地址

验证码(*):

提交

Processed in 0.009068 second(s), 5 queries, Gzip enabled.