How to Hack

Michael Rutter
mjr19@cam.ac.uk

Michaelmas 2003

“No person shall by any wilful, deliberate, reckless or unlawful act . . . jeopardize the integrity of data networks, computing equipment, systems programs, or other stored information.”

IT Syndicate Rules

Why Hack a User's Account?

Useful first step towards root.

Useful platform from which to hack other machines.

Blackmail and/or revenge.

Email forgery for the thick-witted.

Why Hack Root's Account

Trivial access to:

- all user accounts on system
- all data on system (including all keypresses)
- any other machine which trusts hacked machine

and the ability to put arbitrary packets on the local network.

When Root Falls

Once the root account is compromised, the attacker can modify the operating system's kernel in any way he chooses. Thus the compromise can be hidden.

All accesses to disk must go via the kernel, so the kernel can lie about the contents of the disk and thus hide any files, or changes to files, that it wishes.

All enquiries about which processes are running are handled by the kernel, which, therefore, can choose to hide any it wishes to hide.

All enquiries about existing network connections. . .

One spots root compromises either by the mistakes made, or by booting from a 'clean' disk (e.g. a CD).

How to Hack Root

Confuse something which has root privileges to give your process its privileges.

1. Daemons *(internal or external)*
2. Suid programs *(internal)*
3. Files read by root processes *(internal)*

Clearly this is easier if one already has a user account on the system.

Unlocked Unattended Terminals

- Email out the password file
- Edit the `.rhosts` file
- Edit the `.login` file

Clearly such things never happen in TCM(?), but only when one is somewhere foreign. However, good habits are worth cultivating.

/etc/passwd

UNIX stores passwords in a world-readable file after encypting them with a trap-door cypher.

```
> perl -e 'print(crypt("secret","xx")."\n");'
xxWAum7tHdIUw
```

To avoid a simple reverse-lookup table, there are 4096 possible, closely related, algorithms, which are specified by the second string above, which is called the *salt*, and which prefixes the encrypted password.

```
>  perl -e 'print(crypt("secret","aB")."\n");'
aBB1W4NnHXoyY
```

On login, the proffered password is encrypted in the same manner, and the encrypted strings compared.

Cracking passwords

Reversing the algorithm is not thought to be possible, and closely related passwords have very different encrypted forms:

```
secret -> xxWAum7tHdIUw
Secret -> xxS6FgH..85yA
```

although only the first eight characters are significant:

```
>  perl -e 'print(crypt("question","hp")."\n");'
hpwZiCLmlKGFA
>  perl -e 'print(crypt("questionable","hp")."\n");'
hpwZiCLmlKGFA
```

The encryption step was designed to take one second. However, three decades of improvements to algorithms and CPUs have reduced this to around 10μs.

Brute Force

There are about 95 possibilities for each character in a password, and eight characters, so $95^8 = 6e15$ possibilities. One can take an encrypted string, such as `tq6bWD1EuJX9M`, and simply try guessing one's way through the possibilities. This will take a thousand years before one has a 50% probability of sucess.

One can try relying on the fact that humans are stupid, and all their passwords are either eight lower-case letters, or seven with a trailing digit. Then there are just $26^8 + 10 \times 26^7 = 3e11$ possibilites, and the job is done in a couple of months.

Really stupid humans use dictionary words. There are around 20,000 of these, so it takes well under a second to crack such a password.

Utter idiots use their own names.

External Brute Force

Even without a copy of the password file, one can try repeatedly logging on with different passwords. With automation, one can probably achieve ten to a hundred attempts per second, particularly if a range of services (ssh, ftp, xdm) are offered.

Such an attack run over a weekend (when any logs won't be spotted) can run for about 200,000 seconds: plenty of time to try every dictionary word with simple capitalisation variants, simple letter for digit replacements (e.g. '5' for 's', '1' for 'l', etc.), and a trailing digit too.

UiUw2,tL

Some humans claim that choosing and changing passwords is hard. Assuming that their neurons are not completely pickled in alcohol, they are wrong. A human is capable of learning a random string typed once a day after a small number of weeks.

There is nothing wrong with writing passwords down. There is plenty wrong with writing down machine name, user id and password together.

And passwords need changing frequently only if one is careless (or carefree) with them. If one keeps them in a safely paranoid fashion, then changing once every couple of years is fine. If one starts faxing them, using them on untrusted machines and telling them to other people (?!), then changing them every couple of weeks would seem rather infrequent.

‘Improvements’

Unfortunately improvements to this encryption system have not met with much cross-vendor support. The most promising is probably the MD5 passwords introduced (mostly) by Linux, which permits passwords longer than 8 characters.

The least promising was an algorithm which permitted 16 character passwords, and encoded each half separately using the standard crypt algorithm. Thus each half could be broken independently, and even if one were to use all sixteen characters it would merely take twice as long to crack as an eight character password. Thus an exponentially hard problem was cunningly reduced to a linear one.

The Wacky World of Windows

Early versions of Windows used plain-text passwords for accessing network ‘shares’, just like the rest of the world at the time.

Later (c. Windows 98 & Windows NT 4) encryption became the default, with 14 character passwords supported, and a hash stored in a similar fashion to UNIX. Unfortunately, there were a few problems:

- no salt was used
- the password was not case-sensitive
- the 14 characters were encrypted as two independent 7 character parts

Those who believed that Bill’s 14 characters were better than UNIX’s eight were fools.

Current versions of Windows (NT4 SP4 onwards) use an algorithm called ‘NTLMv2’. This is sane, if one can persuade it never to fall back to the old ‘LanMan’ algorithm above!

Network Insecurity

The basic network protocols, such as TCP/IP and ethernet, were not designed to be used in untrusted environments. It is very easy for one machine to pretend to be another, by many, many different means (faking DNS responses, faking ARP replies, advertising oneself as a router, etc.). Such tricks are called *spoofing*.

It is also rather easy to view all traffic to or from another machine on the same network. This trick is called *sniffing*.

Anyone who connects a laptop to TCM's network, or who can gain root (or administrator) access to a computer on TCM's network, can do any of the above.

Protocols

The old, and common, services built on TCP/IP do nothing to address this issue. Telnet, ftp, imap, pop, smtp, X11, xdmcp, rsh, rlogin, rexec, http and many more all come from an era which did not worry about untrusted networks.

Anyone using those services to transmit even vaguely sensitive data (and certainly passwords) across anything but a wholly trusted network is using them in a manner for which they were not designed. And if one is using *anything* in a manner for which it was not designed one must know precisely what one is doing.

telnet, imap, pop, xdmcp, rexec: don't.
ftp, http: anonymous only.
rsh, rlogin: passwordless only, between machines which already trust each other.
smtp: postcardware
X11: see later

The `r` protocols

The `r` protocols form a family. The `rsh`, `rcp` and `xon` commands are all wrappers for `rsh`, and the `rlogin` command is similar except that it is the only one capable of prompting for a password. As it doesn't encrypt, don't give it one.

They allow passwordless login from user1 at machine1 to user2 at machine2 if either:

machine1 is in machine2's `/etc/hosts.equiv` file, and the two userids are identical

or

the text "`machine1 user1`" is in a file called `.rhosts` in users2's home directory on machine2.

Very vulnerable to spoofing, and to a quick:

```
> echo '+ +' >> ~/.rhosts
```

on an unattended terminal. ('+' means anyone/anywhere.)

Encryption in two parts

Encrypted protocols need to prevent both sniffing and spoofing. Not only must they protect data from third parties, but they must enable a remote server to identify itself conclusively before it needs to receive one's password. The latter is as important as the former.

Fortunately two common protocols exist which achieve both parts: SSL, on top of which one can add http or imap, and ssh, which replaces telnet, rsh, rlogin and ftp completely, and provides a way of dealing with X11.

Public / Private key cyphers

One common class of encryption algorithms is *asymmetric*: different keys are required for encryption and decryption. The public encryption key is given out freely, the private decryption key is guarded closely.

If one has a server's public key, and wishes to check whether the server to which one is connecting is the real thing, one merely needs to encrypt a message with its public key, and challenge it to decrypt it. Only one machine in the world will be able to do so.

There is the problem of the initial collection of the server's public key, but, once solved, one can verify all future connections to that machine.

Symmetric cyphers

In a symmetric cypher the encryption and decryption keys are identical. This is useless for issuing cryptographic challenges, but symmetric algorithms can be quicker to compute than the asymmetric version.

Hence one strategy is to use an asymmetric algorithm just for verifying identies and exchanging a symmetric key to be used for encrypting the bulk of the data transmitted. Thus ssh.

X11 is one of the great strengths of the UNIX world. It permits the remote display of applications between multiuser systems in a manner which is far ahead of anything which certain other OSes can offer.

It is also a great weakness, being initially an internal research project within MIT which leaked out. Security was not a feature of the initial design, and has been retrofitted in a slightly awkward fashion.

All X programs work out where they should be displaying by looking at the value of the variable `$DISPLAY`.

Why X security matters

If your X server is willing to accept a connection to it, that connection could:

1. pop up a window on your screen
2. lock your screen (`xlock`)
3. take a copy of your screen (`xwd`)
4. listen to all keystrokes
5. fake keystrokes into certain applications

The first two are inconvenient. The rest are progressively more serious.

X11’s original ‘security’ scheme was to accept all connections.

xhost

The first improvement was host-based authentication, using the `xhost` command. All connections are accepted from certain named hosts. This is only potentially useful for single-user machines, as anyone typing '`xhost + cus`' is potentially giving a few thousand people access to his terminal. It also works only if spoofing is not an issue.

The Good

```
> xhost
access control enabled, only authorized clients can connect
```

The Bad

```
> xhost
access control enabled, only authorized clients can connect
INET:tcmnb1.phy.cam.ac.uk
```

The Ugly (i.e. the utterly illegal and irresponsible)

```
> xhost
access control disabled, clients can connect from any host
```

The man pages

`man xhost`

‘this provides a rudimentary form of privacy control and security. It is only sufficient for a workstation (single user) environment,’

`man Xsecurity`

‘Host Access
. . . This system can work reasonably well in an environment where everyone trusts everyone, or when only a single person can log in to a given machine, and is easy to use when the list of hosts used is small. This system does not work well when multiple people can log in to a single machine and mutual trust does not exist.’

`man X`

‘When you add entries to the host list (with `xhost`), the server no longer performs any authorization on connections from those machines. Be careful with this.’

Magic Cookies

The next improvement was to use one-time passwords called *MIT Magic Cookies*. Each time one logs in at an xdm prompt, a 128-bit pseudorandom number is generated. Any program trying to initiate a connection to the X server must offer this ‘magic cookie’.

The cookie is stored in a file in one’s home directory called `.Xauthority`, readable by its owner only. The program `xauth` is good at manipulating this file.

Any program running as the same user on a machine which shares that home directory can thus read the cookie and connect. Anyone else cannot.

Problems with Cookies

The cookie is transmitted unencrypted every time a new connection is made to the X server, and hence is readily sniffed. Transfering cookies to remote machines is tedious. Many (old) versions of xdm fail to make the cookie as random as they might. One infamous example only ever generated four of the 2^{128} possible combinations, and thus was trivial to break in just four guesses…

However, if you log out each evening, no-one can be doing nasty things to your X session over night, and, when you log in the next morning, you get a nice shiny new cookie.

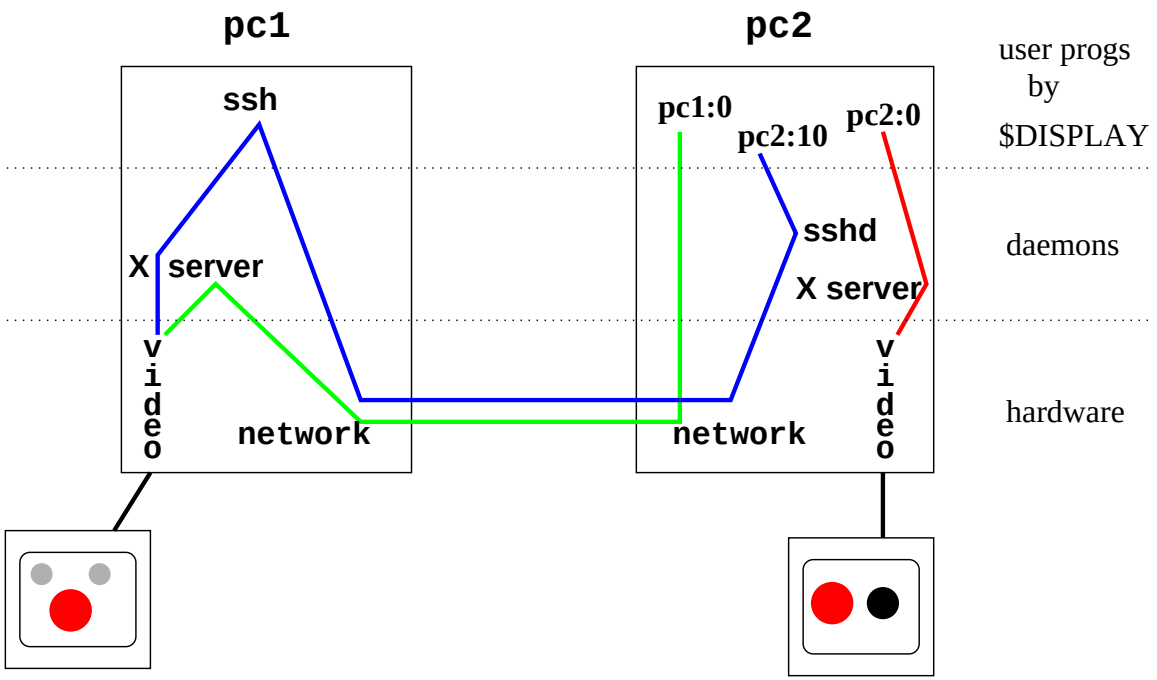
Ssh to the rescue

Most people nowadays know nothing about cookies, because between two UNIX machines one just uses ssh, and X magically works with no need to worry about `$DISPLAY` or `xauth`. The details are as follows.

Someone sitting in front of `pc1` makes an ssh connection to `pc2` (and has X forwarding enabled). The ssh client on `pc1` is clearly capable of connecting to the X server on `pc1`. The ssh daemon on `pc2` runs an alternative X server on `pc2` on the user's behalf. To distinguish it from the real X server on `pc2`, it uses a higher port number, and thus not `:0` and the end of the corresponding `$DISPLAY` variable, but rather `:10` (usually).

Processes running on `pc2` can now try connecting directly to the X servers on either machine, or connecting to this dummy X server, which forwards all traffic over the ssh link and represents it to the X server on `pc1`. All three X servers will have different magic cookies, with ssh correctly changing the cookie for traffic involving its 'X server.'

Ssh in pictures

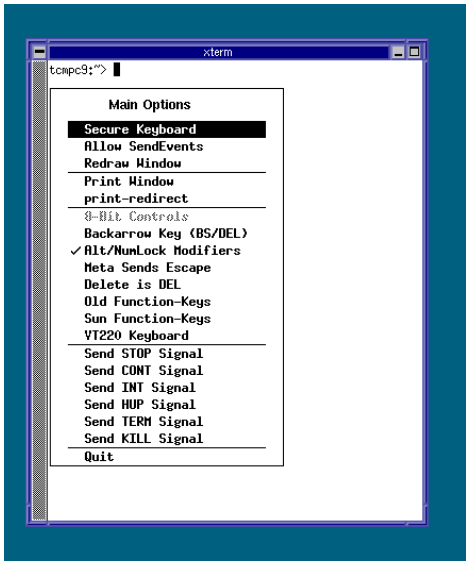


Ssh tricks

```
tcm34:~> logout
Waiting for forwarded connections to terminate...
The following connections are open:
  X11 connection from tcm34.phy.cam.ac.uk port 1985
~?
Supported escape sequences:
~.  - terminate connection
~^Z - suspend ssh
~#  - list forwarded connections
~&  - background ssh (when waiting for connections to terminate)
~?  - this message
~~  - send the escape character by typing it twice
(Note that escapes are only recognized immediately after newline.)
~.
Connection to tcm34 closed.
tcma:~>
```

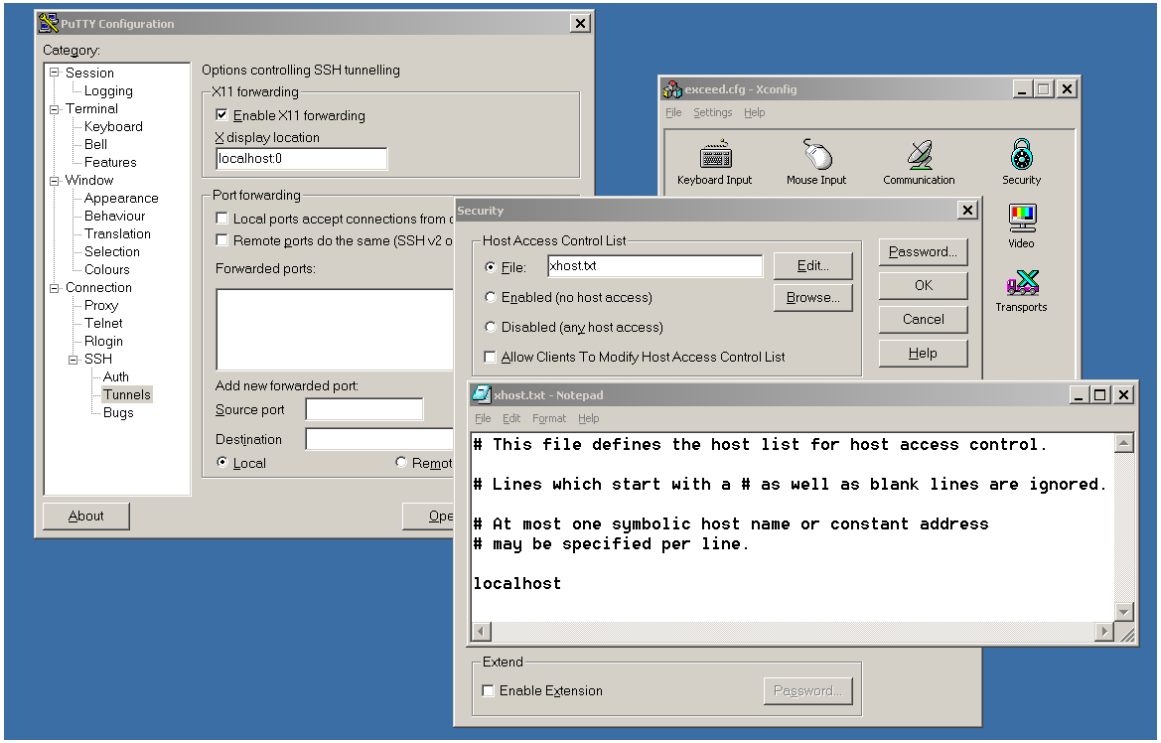
Xterm tricks

Xterm has three menus, accessed by pressing {Ctrl} and each of the three mouse buttons. One option is ‘secure keyboard’, which does an ‘exclusive grab’ on the keyboard, guaranteeing that no other application is being giving keypresses by the Xserver: useful when about to type a password. If successful, the text and background colours are reversed. If not, a warning beep is produced. Undoing the ‘grab’ is a requirement for changing focus.



X/ssh and Windows

Windows users have life easy, for Windows is not a real multi-user operating system, so
`xhost + localhost`
is quite safe, unlike on multiuser UNIX machines.



Quotas and X11

When you log in at an `xdm` prompt, it needs to write Xserver's cookie to your `.Xauthority` file. If you are over quota, it cannot.

If it were sensible, it might attempt to write it to `/tmp` and point applications there, but it doesn't. It just returns you to the login screen as all your X programs, including your window manager, fail to start (because the X server rejects them for having no valid cookie).

Similarly whenever you ssh into a machine, only ssh goes one stage better: if you are over quota, it is capable of deleting all of your existing cookies too.

Log out!

X programs need to authenticate themselves to the server, either via xhost or, preferably, with a magic cookie, when they start. Once started, they (usually) never need to reauthenticate themselves. This can be useful: one can type

```
xhost + machine
```

start an X application from it, and immediately type

```
xhost - machine
```

(the above is not a recommendation...)

It can also be bad. Once some kind person has started running a keystroke sniffer attached to your display, no matter what you do with `xhost` and cookies, there is no way of getting rid of him until you log out and thus reset the X server, at which point the X server will close all open connections.

So log out each evening, not each month.

WWW confusion

```
<html>
Click here to go to
<a href=www.hackers.org>microsoft.com</a>
</html>
```

Are things you click on always what they seem?

Would you fall for the ‘paste the following URL into your browser’ trick:
`http://www.barclays.com@131.111.8.46/verify`

Where does that page come from?

```
<head>
<meta http-equiv="Refresh" content="0; url=http://www.microsoft.com">
</head>

<script language="JavaScript" type="text/javascript">

<!--
window.open('http://www.hackers.org/ms.html','x','menubar=no');
// -->

</script>
```

SSL to the rescue

Again, encryption, this time in the form of SSL (https) solves these problems by allowing a server to identify itself as well as by encrypting traffic between client and server.

The identification is done by a cryptographic signing procedure. The remote site offers a certificate containing the server's public key, the name of the server, an expiry date, and a signature. The signature consist of a hash (publically known algorithm) of the certificate, which is then fed through the decryption stage of a public-private key encryption procedure. Any WWW browser which uses SSL comes with a set of public signing keys, and using the corresponding key it can decrypt the encypted hash, and calculate the hash from the signed message. The two should agree.

Any SSL certificate will be valid for a restricted time and domain, and is (theoretically) revokable. So anyone could construct a certificate claiming to be valid for `www.amazon.com`, and anyone could sign it with their own key, but most browsers accept only a small number of signing keys (Thawte and Verisign are the best known) without producing warning messages.

Email madness

```
Date: Sat, 15 Nov 2003 22:41:09 +0100 (CET)
From: MS Security Support <ovblafs@support.msdn.com>
To: MS Corporation User <tneiuntg-igmnagql@support.msdn.com>
Subject: Internet Critical Patch
Parts/attachments:
   1.1.1 Shown   ~32 lines  Text
   2             109 KB     Application

Microsoft User

this is the latest version of security update, the "November 2003,
Cumulative Patch" update which fixes all known security vulnerabilities
affecting MS Internet Explorer, MS Outlook and MS Outlook Express as
well as three newly discovered vulnerabilities. Install now to continue
keeping your computer secure from these vulnerabilities, the most serious
of which could allow an attacker to run executable on your system. This
update includes the functionality of all previously released patches.

Microsoft Product Support Services and Knowledge Base articles can be found
on  the Microsoft Technical Support web site, http://support.microsoft.com/
```

Running your own UNIX system

A big topic, too big for a single lecture:

> You should assume that *every* Unix box is shipped insecure.
>
> Spend at least a fortnight getting familiar with your system. Understand what the files and commands *really* do. This will take a huge chunk out of your research time, but that's too bad; it's the price you have to pay for the convenience of a system on the Internet.
>
> Expect to spend two to three hours *every* week looking after your machine.

That advice was written in 2000, when hackers were less active than they are now.

Honey-pots

An unpatched Windows2000 system will survive about 15 minutes on the Internet before it is hacked.

An unpatched RedHat 7.3 Linux machine about a day.

An unpatched Digital UNIX machine about a year.

Do check the relevant sites for security announcements. Frequently. (That's more than once a week.)

(Automatic updates go some considerable way to helping. If enabled. And working. And if they cover Office.)

Sensible precautions

When on holiday / at a conference, if all you want to do is send and receive email, use a system which does not give shell access and therefore will be of no use to hackers: e.g. hermes not CUS or TCM.

Why not change your password before going away, and then, if you must, change it back to what it was on your return? Otherwise, simply change it on your return.