Trust and Open Source

Based on: "Reflections on Trusting Trust" by Ken Thompson

Alon Altman

alon@haifux.org

Haifa Linux Club

Trust

Trust

From the dictionary:

> **trust n.,** Firm reliance on the integrity, ability, or character of a person or thing.

So, who do *you* trust?

Case Study: Joe Newbie

- Joe Newbie buys a PC from Dell, preloaded with Windows XP home.
- He connects to his ISP (AOL), downloads and installs Kazaa.
- The next day, he goes to the bank and gets a password for internet access.
- He launches IE and visits the bank's site, noticing the key symbol and proceeds to enter his password, and do his business.

Joe Newbie - Trust

Who does Joe Newbie (implictly) trust?

- Dell, suppliers of his PC.
- Microsoft, suppliers of the OS and browser.
- AOL, his ISP.
- Sherman Networks, suppliers of Kazaa.
- Suppliers of spyware bundled with Kazaa.
- The bank.
- Verisign, who signed the bank's certificate.
- Any other person who had physical access to the machine.
- Anyone else?

Case Study: Bob Hacker

- Bob built his own PC from parts mail-ordered direct from suppliers around the world.
- Bob's ISP is a local Linux-supportive ISP which sponsors installation parties.
- Bob used a Knoppix CD to download Debian CD images from the web, carefully checking the MD5 sum of each CD against the published sum from `debian.org`.
- He then burned these Debian CDs, and proceeded to install a clean system.
- After the install he connected to the Internet, downloaded a new version of Mozilla from `mozilla.org`, and proceeded to connect to `amazon.com` and make an order.

Bob Hacker - Trust

Who does Bob Hacker (implictly) trust?

- Knoppix, suppliers of the boot CD.
- Whoever gave him the Knoppix CD.
- The ISP.
- Debian, suppliers of the distro, and sysadmins of debian.org.
- The Open-Source code review process.
- Mozilla maintainers and sysadmins.
- Amazon.
- Verisign, who signed the Amazon's certificate.
- Anyone else?

Is this really such a big deal?

- Do open source servers really get hacked?
  - `debian.org` was recently broken into.
  - Just this week, Gnome 2.6 release was delayed due to a break in to `www.gnome.org`.
- Do people really insert exploits in open source?
  - The infamous TCPWrappers exploit of ’99 opened a root shell in what was supposed to be security product.
  - In 2002, trojan versions of the popular sniffer tools `tcpdump` and `libpcap` were placed on www.tcpdump.org, after it was hacked.

Case Study: Paranoid Mike

- Mike works for the NSA, he made two Linux from Scratch systems from a certified-good NSALinux PC at work.
- Mike personally checked every line of code of every script, file and program installed on his LFS system before compiling and installing.
- He has then connected one machine to high-speed internet, allowing only incoming SSH connections, writing the host key fingerprint on paper to verify its identity.
- He has then connected from the second machine to the first.

Reflections on Trusting Trust

Stage 1 — A Quine

Here is a simple self-generating program:

```
char s[] = {'0','}',';','\n','\n','/',
            '*', ...(deleted)... , 0};

/* The string s is a representation
 * of the body of this program from
 * '0' to the end.
 */
main() {
  int i;
  printf("char s[] = {");
  for (i=0; s[i]; i++)
    printf("%d,",s[i]);
  printf("%s",s);
}
```

Stage 1 — Comments

- The program in the previous slide, produces itself when run (or actually, produces a self-reproducing program).
- It can be easily written by another program.
- This program can contain an arbitrary amount of excess baggage. In this example, even the comment is reproduced.

Stage 2

Consider the code used to parse special characters such as `\n` in the C compiler. The code might look something like this:

```
...
c = next();
if (c!='\\') return (c);
c = next(); /* escape sequence */
if (c=='\\') return ('\\');
if (c=='n')  return ('n');
...
```

This code 'knows' in a completely portable way what character represents a newline in any character set, thus recompling itself and perpetuating the knowledge.

Stage 2 — Problem

Suppose we want to add a vertical tab character `\v` to the C compiler:

```
...
c = next();
if (c!='\\') return (c);
c = next(); /* escape sequence */
if (c=='\\') return ('\\');
if (c=='n')  return ('\n');
if (c=='v')  return ('\v');
...
```

This code does not work, as the binary version of the C compiler does not yet know what `\v` means.

Stage 2 — Bootstrapping

To let the compiler know about `\v` we must tell it once that it is ASCII code 11:

```
...
c = next();
if (c!='\\') return (c);
c = next(); /* escape sequence */
if (c=='\\') return ('\\');
if (c=='n')  return ('\n');
if (c=='v')  return (11);
...
```

Now we can use our new compiler to compile the code from the previous slide, as the compiler has "learnt" this new definition.

Stage 3

- Now, again in the C compiler, consider a function `compile` used to compile the next chunk of source.
- Suppose we modify the compiler, to deliberately miscompile the source whenever a patteren is matched, thus creating a “Trojan horse”:

```
void compile(char *s) {
  if (match(s,"pattern")) {
    compile("bug");
    return;
  }
  ...
}
```

Stage 3 — The bug

- Suppose we code a bug that will deliberately miscompile OpenSSH (or actually, PAM) to accept a known password in addition to the real one.
- Then we use the above scheme to change the compiler to automatically insert the bug.
- This bug will easily be noticed if you release the code to the compiler.
- However, if you change only the binary version, a re-compile of the compiler (like Mike and the Debian maintainers do) will ensure the bug be gone.

Stage 3 — final version

To solve this problem, we change the `compile` function as follows:

```
void compile(char *s) {
  if (match(s,"pattern1")) {
    compile("bug1"); return;
  }
  if (match(s,"pattern2")) {
    compile("bug2"); return;
  }
  ...
}
```

Suppose `bug1` inserts the bug into OpenSSH, and `bug2` is a self-reproducing program inserting *both* bugs into the compiler.

Moral

- No matter how much you inspect the source you must trust someone.
- Even if you check the binary of the compiler, a bug may be hidden in your processor's microcode.
- In fact, trecherous hardware that will not give the user control over her own machine is in production as we speak.

Discussion