Lecture 23. Viruses and Secret Messages

• Remember `sum.toy`?

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0E</td>
<td></td>
<td>starting address</td>
</tr>
<tr>
<td>0E: B001</td>
<td>R0 &lt;- 01</td>
<td>R0 holds 1</td>
</tr>
<tr>
<td>0F: B10A</td>
<td>R1 &lt;- 0A</td>
<td>R1 is n</td>
</tr>
<tr>
<td>10: B201</td>
<td>R2 &lt;- 01</td>
<td>R2 is i</td>
</tr>
<tr>
<td>11: B300</td>
<td>R3 &lt;- 00</td>
<td>R3 is sum</td>
</tr>
<tr>
<td>12: 2110</td>
<td>R1 &lt;- R1 - R0</td>
<td>n--</td>
</tr>
<tr>
<td>13: 6118</td>
<td>jump to 18 if R1 &lt; 0</td>
<td>if (n &lt; 0) goto End</td>
</tr>
<tr>
<td>14: 1332</td>
<td>R3 &lt;- R3 + R2</td>
<td>sum += i</td>
</tr>
<tr>
<td>15: 1220</td>
<td>R2 &lt;- R2 + R0</td>
<td>i++</td>
</tr>
<tr>
<td>16: 2110</td>
<td>R1 &lt;- R1 - R0</td>
<td>n--</td>
</tr>
<tr>
<td>17: 5013</td>
<td>jump to 13</td>
<td>goto Top</td>
</tr>
<tr>
<td>18: 4302</td>
<td>print R3</td>
<td>print sum</td>
</tr>
<tr>
<td>19: 0000</td>
<td></td>
<td>halt</td>
</tr>
</tbody>
</table>

% /u/cs217/bin/toy /u/cs217/toy/sum.toy
0037

• Suppose an unknown source **modifies** `sum.toy` by appending the following code

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>87: 8088</td>
<td>R0 &lt;- 88</td>
<td>% /u/cs217/bin/toy /u/cs217/toy/sum.toy</td>
</tr>
<tr>
<td>88: B108</td>
<td>R1 &lt;- 08</td>
<td><strong>8888</strong></td>
</tr>
<tr>
<td>89: F201</td>
<td>R2 &lt;- R0&lt;&lt;R1</td>
<td>0037</td>
</tr>
<tr>
<td>8A: C002</td>
<td>R0 &lt;- R0^R2</td>
<td></td>
</tr>
<tr>
<td>8B: 4002</td>
<td>print R0</td>
<td></td>
</tr>
<tr>
<td>8C: 500E</td>
<td>jump to 0E</td>
<td><code>sum.toy</code> is infected with the ‘8888’ virus</td>
</tr>
<tr>
<td>87</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Infection Routes

- If a virus $V$ can find a **writable executable file** $P$, it may be able to embed itself in $P$
  
  $\text{infect}(P,V)$  A copy of $P$ with $V$ embedded so $V$ gets initial control
  
  $V$’s execution can be arbitrarily complex, perhaps involving self-modifying code to cover its tracks

- When $\text{infect}(P,V)$ runs, $V$ can do anything $P$ can do, perhaps without visible effects

  **Print ‘8888’**

  **Print**

  login:

  On some other computer and wait for a user id; then print

  **Password:**

  Snarf the password entered, spawn another process running `/bin/login`, and leave town with a fresh user id and password; user just sees

  login:

  **Scramble/delete your files**

  **Spawn a separate process running itself and find other executable files to infect**
Detecting Viruses

• Given a program $P$, how can you tell if it’s infected? You can’t
• Virus detection software looks for occurrences of specific viruses
e.g.,

  Is the instruction at location $87_{16} = 8088_{16}$? ‘Infected with the 8888 virus’

Oh oh… Viruses embed themselves in different ways and at different locations
Must update virus detection software on a regular basis (daily?)
Virus detection software does not solve the general problem ‘is $P$ infected?’

• Suppose you have two versions of supposedly the same program, $P_1$ and $P_2$
  Which one of $P_1$ or $P_2$ is infected?
  Do $P_1$ and $P_2$ produce the same output? (Even if one is infected)
  Both are unsolvable problems alà the Halting Problem

• Is there any hope?
  Intractable problems — those with only exponential-time algorithms — come to
  the rescue
Fingerprints

• Suppose that given a file $P$, $H(P)$ is a relatively small number that ‘characterizes’ $P$

  \[ H(/u/cs126/examples/compile.c) = 364BFFB1_{16} \]

  $H$ provides a fingerprint of /u/cs126/examples/compile.c

  Accept $P_2$, a copy of $P$, only if $H(P_2) = 364BFFB1_{16}$

• $H$ must be a one-way hash function with the following properties

  Given $P$, it must be easy to compute $H(P)$

  Given $H(P)$, it must be computationally infeasible to reconstruct $P$

  Given $P$ and a virus $V$, it must be computationally infeasible to arrange for $H(\text{infect}(P,V)) = H(P)$; that is, to find two bit strings with equal fingerprints

• Good one-way hash functions produce fingerprints with at least 128 bits

  MD5(compile.c) 979a7c5c ae9f12e2 702fc6ad 9ad4493a

  SHA(compile.c) 85025ddc bb5c8da7 44598fe0 d8b5e16d a75cb560
Fingerprints on the Internet

% ftp ftp.cs.princeton.edu
ftp> cd /pub/packages/cii
ftp> ls
README
cii10.tar.gz
cii10.tar.Z
cii10.zip
ftp> get README |more
...
The distribution directory contains the following files and directories. MD5 fingerprints for the files in this directory are listed below.
...
MD5 (cii10.tar.Z) = ba5b3c3b6c43061e4519c85f103be606
MD5 (cii10.tar.gz) = e3769ae785ec52427e1b807e02aae3e
MD5 (cii10.zip) = fa71f475c97a4bfae66767012367c77f
ftp> get cii10.zip
ftp> quit
% md5 cii10.zip
MD5 (cii10.zip) = fa71f475c97a4bfae66767012367c77f

• This isn’t foolproof — intruders can intercept Internet packets and substitute different fingerprints
Cryptography

- A cryptosystem keeps secret messages (and files) from prying eyes

`Please send money` 24 F8 A7 86 63 2E 28 0A

`Please send money`

68 25 B1 73 5F E0 70 99 E2

Key: 01 23 45 67 89 AB CD EF

- Modern cryptosystems exclusive-OR key with plaintext: \( C = P \oplus K \)

```c
void encrypt(char *buf, int len, char *key, int keylen) {
    int i = 0;
    for (i = 0; len-- > 0; i = (i + 1)%keylen)
        *buf++ ^= key[i];
}
```

Works for encryption and decryption: \( C \oplus K = (P \oplus K) \oplus K = P \oplus (K \oplus K) = P \oplus 0 = P \!

Watch out! Sending many 0s in plaintext gives attackers pure key: \( C = 0 \oplus K = K \)
Cryptography, cont’d

• Repeated use of a relatively short key isn’t secure; most systems use the key to generate a long stream of pseudo-key, which is XOR’d with the plaintext

• Assume the worst: Attackers know the algorithm, the length of the key, and have the ciphertext

• Security rests on the strength of the algorithm and the security of the key

• Best systems force attackers to use inefficient algorithms, which require trying try all $2^n$ n-bit keys; just use large $n$

• Designing secure cryptosystems sounds easy, but it’s not; don’t trust amateurs!

• Key distribution is just as hard as encryption: What’s the best way to exchange keys with your trusted correspondents and keep them secret? There isn’t one…

Public-Key Cryptosystems

- **Public-key** cryptosystems avoid the key distribution problem by using *two keys*
  
  Everyone knows your public key, $P$
  
  Only you know your secret key, $S$
  
  To send $M$: Send $P_{drh}(M)$ via any medium
  
  To read $M$: I read $S_{drh}(M)$

- List public keys in the phone book, or its equivalent

  ```
  % finger -l drh@cs.princeton.edu
  ...
  -----BEGIN PGP PUBLIC KEY BLOCK-----
  Version: 2.6.1
  mQBNAiluT8gAAAECAK8TOxmbQ6XhoJXrGptDKzhZkIqSRh3pMimt8nUh1nSfByec
  KittyH02STppLwncD47j8KK6Cm5hriyzusnX/hkABRG0JkRhdmlkIFIuIEhhbnNv
  biA8ZHJoQGNzLnByaW5jZXRvbi51ZHU+
  =JFCd
  -----END PGP PUBLIC KEY BLOCK-----
  ```

- For all public-key algorithms

  $$S(P(M)) = M \text{ for all } M$$
  
  All $S$, $P$ pairs must be distinct
  
  Deriving $S$ from $P$ must be as hard as reading $M$
  
  $P(M)$ and $S(M)$ must be efficient
RSA Public-Key Cryptosystem

• The RSA cryptosystem uses arithmetic on very large integers
  \[ P \text{ is } N, p \]
  \[ S \text{ is } N, s \]
  where \( N \approx 200 \) digits, \( p \) and \( s \approx 100 \) digits

• To choose \( N, p, s \)
  
  Pick 3 100-digit secret prime numbers, \( x, y, s \)
  The largest is \( s \)
  
  \[ N = x \times y \]
  
  \[ N = 47 \times 79 = 3713 \]

  Choose \( p \) so that \((p \times s) \mod ((x - 1)(y - 1)) = 1\)
  
  \[ p \times 97 \mod (46 \times 78) = 1 \]
  \[ 37 \times 97 \mod 3588 = 1 \]
  
  \[ 3589/3588 = 1 \text{ remainder } 1 \]

• Attackers see only \( N \) and \( p \)
  
  To find \( s \), attackers must factor \( N \) into its prime factors \( x \) and \( y \)
  
  It is believed, but not proven, to be infeasible to factor \( N \) if it’s sufficiently large
  
  Factoring 200-digit numbers probably takes \( \approx 10^9 \) years

• Are there enough primes for everyone? Yes: \( \approx 10^{150} \) primes with \( \leq 512 \) bits (\( \approx 155 \) decimal digits)
RSA Encryption

• To encrypt $M$, use $N$ and the public key, $p$

  Encode $M$ in numbers $< N$

  For each $M_i$, $C_i = M_i^p \mod N$ the remainder of $M_i^p$ when divided by $N$

For $N = 3713$, $p = 37$, $s = 97$

$M$ Please send money

Encode: 1612 0501 1905 0019 0514 0400 1315 1405 2500

Encrypt: 2080 0057 1857 3706 1584 0888 2067 0591 1277

$1612^{37} = 47,044,232,358,938,497,020,498,996,761,564,680,247,331,818,$

$462,325,046,870,527,453,082,869,350,611,474,961,064,423,374,$

$436,277,844,788,137,937,637,623,201,792$

$1612^{37} \mod 3713 = 2080$, etc.
RSA Decryption

• To *decrypt* \( M \), use \( N \) and the *private* key, \( s \)

  For each \( C_i \), \( M_i = C_i^s \mod N \)

  Decode numbers to reveal \( M \)

For \( N = 3713, \ p = 37, \ s = 97 \)

Please send money

\[
\begin{array}{cccccccccccc}
\times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \\
\end{array}
\]

\( C: \quad 2080 \ 0057 \ 1857 \ 3706 \ 1584 \ 0888 \ 2067 \ 0591 \ 1277 \)

Decrypt: \( 1612 \ 0501 \ 1905 \ 0019 \ 0514 \ 0400 \ 1315 \ 1405 \ 2500 \)

\( 57^{97} = 208,862,754,025,291,103,893,549,722,030,506,307,840,035,159, \)
\( 185,066,358,136,864,739,390,751,752,973,213,714,581,100,145, \)
\( 330,888,003,488,562,198,990,224,718,358,613,240,589,340,493,287, \)
\( 521,060,551,858,632,460,253,869,992,608,057 \)

\( 57^{97} \mod 3713 = 501 \)

Decode: \( 1612 \ 0501 \ 1905 \ 0019 \ 0514 \ 0400 \ 1315 \ 1405 \ 2500 \)

\( \text{PLEASE } \ _\text{SEND}_\ _\text{MO} \ _\text{NEY} \ _\)

• This example is from R. Sedgewick, *Algorithms in C*, Addison-Wesley, 1990

• For details on multiple-precision arithmetic, see D. R. Hanson, *C Interfaces and Implementations*, Addison-Wesley, 1997
PGP

- PGP — Pretty Good Privacy — is widely used public-key cryptosystem available for PCs, UNIX systems, etc.

you% cat / pgp -fea drh
Pretty Good Privacy (tm) 2.6.2 - Public-key encryption for the masses.
Can I have more time on the current
programming assignment?
--frazzled in Princeton
^D

-----BEGIN PGP MESSAGE-----

pgp -fda

Incorporating new mail into inbox...

  92+ 09/04 To:drh@fs.CS.Prin  <<-----BEGIN PGP MESSAGE-----

Can I have more time on the current
programming assignment?
--frazzled in Princeton