COMP 3704 Computer Security

Christian Grothoff
christian@grothoff.org

http://grothoff.org/christian/
Dilbert

Tour of Accounting

Over here we have our random number generator.

Nine nine nine nine nine.

Are you sure that's random?

That's the problem with randomness: you can never be sure.

Copyright © 2001 United Feature Syndicate, Inc.
Motivation

• How do we seed the PRNG algorithm?
• What if our PRNG is too predictable?

⇒ Need “physical” sources of randomness!
Security Requirements

• Randomness
• Forward Security
• Backward Security
Basic Structure
Candidates from /proc

- Dirty pages
- Page faults
- Free pages
- System uptime
- Disk I/O (total number, merged, sectors, time spent)
- IO operations in progress
- Hardware interrupt counters
Candidates from `/proc/net/`

- Number of packets received
- Number of packets sent
- Transmission errors
- Network latency
Timing

- Date and time
- CPU cycle count:
  
  ```c
  __inline__ uint64_t rdtsc() {
    uint64_t x;
    __asm__ volatile ("rdtsc" : ":A" (x));
    return x;
  }
  ```
- CPU performance counters (cache misses)
- Timing of events (keystrokes, mouse)
Entropy, again!

- For each entropy source, we would like to know how much entropy it actually provides!
- Estimating entropy is costly/difficult!
- Key idea: use **compression**!
- Compression estimates (non)randomness in data
- Can even use lossy data compression for estimation
- Compression ratio of 1:10 is typical!
Cheaper Entropy Estimation

Let $t_n$ denote the timing of event number $n$. Define:

\[ \delta_n = t_n - t_{n-1} \]
\[ \delta^2_n = \delta_n - \delta_{n-1} \]
\[ \delta^3_n = \delta^2_n - \delta^2_{n-1} \]

Estimate the entropy added by event $t_n$ to be:

\[ \log_2 \left( \min(\{ |\delta_n|, |\delta^2_n|, |\delta^3_n| \}) \right) \] (1)
Main Design Choices

Choices for the RNG implementation:

- Non-blocking or blocking or both?
- Hashing method
- Entropy sources and entropy estimators
- Deterministic modifiers (counters)
- Pool size

End-user: Blocking, non-blocking or deterministic?
Usability Issues

A safety check to test whether the PRNG of OpenSSL was properly initialized was added to version 0.95. User responses were to seed it with:

- a constant
- output from \texttt{rand()}
- public key
- the executable
- \texttt{/etc/passwd}, \texttt{/var/syslog}
Usability Issues

Cryptographers: The device should refuse to work unless sufficient entropy is available.

Product developers: Cannot ship device that refuses to function.

- 0.01% of users have discipline to handle RNG failures
- 99.99% will see RNG failure as defective product

⇒ Actual security is a hard sell!
Questions
Problem

Design a protocol that does not use public key cryptography which allows Bob to prove his identity to Alice. Assume that Bob and Alice share a secret key $K$. Make sure that your protocol is safe against replay attacks; it does not have to be secure against a man-in-the-middle attack.
Homework Hint

- https://gnunet.org/svn/Extractor/src/plugins/hash/