Erlang secure RPC and SSH module

Kenji Rikitake
Erlang Factory SF Bay Area 2010
25-MAR-2010
Who's talking?

• Name: Kenji Rikitake (rhee-key-tah-kay)
• '90-'92: VAX/VMS OS developer
• '92-'00: Corporate network admin
  – designed firewalls and sandbox systems
• '01-: Network security researcher
  – DNS UDP payload length (critical for DNSSEC)
  – IPv6 and NGN vulnerability issues
  – Studying Erlang for secure distributed systems
My Erlang activities

• Bitten by the Erlang bug in 2008
  – by Japanese version of Programming Erlang

• Patches accepted
  – TAI (leap second) (R13B, OTP-7609)
  – SSH aes128-cbc (R13B02, OTP-8110)
  – backporting FreeBSD patches (R13B04)
    • compiled works of Giacomo Olgeni, Paul Guyot and other FreeBSD Port contributors
  – FreeBSD Port support (lang/erlang)
Topics

• Security weakness in Erlang
• Why SSH for Erlang RPC?
• SSH protocol overview
• How Erlang supports SSH
• Prototype implementation and results
• Future plans and thoughts
Security weakness in Erlang (1)

• Clarification: Erlang/OTP actually has a lot of strength in secure programming
  – no pointer assignment
  – once-and-only-once variable assignment
  – message-passing based = minimized sharing
  – restrictive access for I/O devices
    • port, linked-in drivers, NIFs
  – OTP supports secure communication modules
    • crypto, public_key, ssh, ssl, etc.
Security weakness in Erlang (2)

• Problem 1: inter-node TCP links are not cryptographically protected by default
  – exception: inet_ssl_dist (not well-supported)
• Problem 2: weak inter-node authentication
  – only by pre-shared plaintext cookies
• Problem 3: epmd is totally unprotected
  – and is quite hard to implement a security policy on epmd either
  – many applications depend on epmd
Security weakness in Erlang (3)

• An important issue I will not address here
  – An Erlang node assumes all registered processes in the node and other nodes are equally trustable with each other
    • Making a sandbox environment within an Erlang VM might be extremely difficult, without resolving dependency between the library modules
    • Denial-of-Service (DoS) attacks to all the nodes in the RPC network are possible once the attacker gains control in an Erlang node
Erlang/OTP inter-node RPC

- Two kinds of links:
  - network link between hosts
  - inter-process links between the nodes and epmds

- Three types of inter-process links which have to be cryptographically protected:
  - between nodes (plain unencrypted TCP by default)
  - between nodes and epmds (usually within a host)
  - between epmds (plain unencrypted TCP only)
A traditional workaround for securing Erlang/OTP RPC

- Isolating Erlang nodes inside the perimeter is the most popular practice for protecting them
- A gateway (GW) process in an Erlang-running host is needed to communicate outside the perimeter

Possibly hostile networks and machines (e.g., global Internet)

Host A

- node
- epmd

GW proc

Host B

- node
- epmd

network link
Another way of securing Erlang inter-node links

- If each node is connected only through a hostile environment where attackers try to eavesdrop the communication between the nodes, all communication links between the nodes must be encrypted and authenticated.

- Protocol candidates for securing the links should be on the application level, such as:
  - SSL/TLS
  - SSH (Secure Shell)

- IPsec does not fit well for this purpose (only host-level policy).

Each link must be cryptographically protected.
Why SSH for Erlang RPC?

• For stronger auth/encryption channels
  – SSH is easier for sysadmin than SSL
    • SSH key management is a part of daily job
  – Erlang/OTP already has full SSH capability
    • including SFTP client/server in Erlang
    • OTP ssh_channel behaviour provided

• SSH safely coexists with the current RPC
  – Remote execution over SSH will not break existing modules
Related works

• Jungerl SSH
  – I assume it's the ancestor of OTP ssh module
  – no longer maintained since 2006
  – not working on current Erlang R13B04

• RPC ideas
  – BERT-RPC: generic RPC through Erlang
    • http://www.bert-rpc.org/
  – SDIST by Dave "dizzyd" Smith (of Basho)
    • multi-level authentication and security models
## SSH protocol overview (as in RFC4251)

<table>
<thead>
<tr>
<th>User shells and programs</th>
<th>Ports forwarded via TCP tunnels</th>
</tr>
</thead>
<tbody>
<tr>
<td>(called via shell, exec, subsystems)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSH client and server programs (including Erlang/OTP ssh and related modules)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SSH Authentication Protocol</th>
<th>SSH Connection Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC4252</td>
<td>RFC4254</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSH Transport Protocol (RFC4253)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>TCP (reliable, duplex, transparent, byte-oriented)</th>
</tr>
</thead>
</table>
SSH Communication Protocol (RFC4254) (1)

• Handling multiple streams of:
  – pseudo ttys (termcap, window size, signals)
  – TCP tunnels (X11/port forwarding)

• Application over SSH works as:
  – shell: interactive shell
  – exec: one-time remote execution (SCP)
  – subsystem: user-named services (SFTP)
  – Erlang ssh module supports all of these
SSH Communication Protocol (RFC4254) (2)

• Maintaining send/receive window
  – Keeping buffer windows for each direction
    • after sending message, window size decreases
    • after receiving acks, window size increases
    • This will prevent flooding without acks
      – when no ack comes the transfer will automatically stop
  – Window size is adjustable per request
    • tunable on purpose
      – interactive .vs. file transfer
SSH Transport Protocol (RFC4253)

- Server authentication (Diffie-Hellman)
- Protocol negotiation
  - Transport details
    - shared-key encryption and compression algorithms
    - HMAC for message integrity check
  - Server public-key encryption
- Binary packet format passed on to TCP
- Service requests
  - User authentication / Channel connection
SSH Authentication Protocol (RFC4252)

- User authentication
  - after the SSH transport is established
  - available authentication methods
    - public-key: pre-distributed private and public keys
    - password: conventional password of the host
    - host-based: trusting the host auth (rlogin/rsh)

- Banner message handling
What Erlang/OTP provides

- R13B04 ssh-1.1.8 application provides:
  - password and public-key user authentication
    • CAUTION: no password encryption for private keys
  - interactive SSH shell running on a BEAM
  - one-time SSH execution on a BEAM
    • passing a string to the shell as a command
  - frameworks for SSH subsystems
    • example of SFTP client/server code
  - ssh_channel behaviour of OTP programming
interaction between user code and ssh_channel behaviour

External SSH programs

OTP
ssh_channel behaviour and the ssh library functions

SSH RPC client/server code

request/data received from the other end

replies/data sent to the other end

sshrpc subsystem modules including callback routines

shut down request for the connection

terminate and disconnect
Prototype code (already implemented)

- Remote execution of functions
  - Module:Function(Arguments)-style execution
- Non-blocking call handling
  - Synchronous call
  - Exchanged data may be more than a single SSH packet
    - subsystem-level buffering required
Implementation details (1): packet format

It's basically an Erlang External Format Term embedded with the 4-byte content length header; minimal for larger message exchange over SSH binary packets

<< Length:32/unsigned-big-integer,
  % 4-byte binary (?UINT32())
  Content/binary
  % of the Length bytes
>>
Implementation details (2): message structure

• Each message is an Erlang tuple
  – marshalled with term_to_binary and demarshalled with binary_to_term

• Two types of messages
  – \{mfa, M, F, A\}: command of an M:F(A)
  – \{answer, Term\}: reply as an Erlang Term
Implementation details (3): ssh module modification

- Added aes128-cbc encryption (R13B02)
  - RFC4253 Section 6.3 recommends this
    - OTP SSH only had 3des-cbc (a required algorithm)
    - crypto:aes_cbc_ivec/1 added
  - ssh_transport:unpack/3 bugfix needed
    - of handling zero-length packets
  - Other algorithms can be added as well
    - blowfish-cbc: already in R13B04 crypto module
Implementation status as of 21-MAR-2010

- Basic server code complete
  - simply passing \{M, F, A\} to erlang:apply/3
  - multi-packet SSH message can be handled
- Basic client code complete
  - Non-blocking OTP code complete
- See my GitHub repository for the details
  - http://github.com/jj1bdx/sshrpc/
Performance evaluation

• ~500 sequential calls/second
  – System specification:
    • FreeBSD 7.2-RELEASE i386
    • Client: Core2Duo 2.2GHz memory: 2Gbytes
    • Server: Atom 1.6GHz memory: 1Gbytes
    • IPv4, 100BASE-TX
  – executed lists:seq(1,100) for 10000 times
  – CPU usage of server: 1~15%
Future plans and thoughts (1)

- ssh module needs more fixes and features
  - priority/choice of shared-key cryptography
    - current: hard-coded as [aes128-cbc, 3des-cbc]
    - more algorithms can/should be included
      - blowfish, aes192/aes256, etc.
  - Public key management
  - On R13B04 IPv6 client connection fails
    - the server/daemon code works OK
  - More comprehensive testing needed
Future plans and thoughts (2)

• RPC functions not yet implemented:
  – Asynchronous call handling
    • per-transaction ID needed
  – Spawning a remote process
  – Sending a message to a running process
  – Limiting the modules/functions to be called
• Many subsystems can be run concurrently
  – secure monitoring, control, logging, etc.
Acknowledgments (1)

- Dave "dizzyd" Smith for his SDIST paper
- Francesco Cesarini and Ulf Wiger for giving me a time slot of this presentation
- People on erlang-questions mailing list for their constructive criticisms:
  - Including Jason Vantuyl, Kenneth Lundin, Witold Babyluk, and Richard Andrews
Acknowledgments (2)

• This project is supported by:
  – Network Security Incident Response Group, National Institute of information and Communications Technology (NICT), Japan

• Tokyo Erlang Workshop activists
  – Including @cooldaemon, @voluntas, @takemaru_jp, @kuenishi, @higepon (all Twitter IDs)
Thanks

- Questions?