Session Resumption for the Secure Shell Protocol

Jürgen Schönwälder

http://www.eecs.jacobs-university.de/users/schoenw/

June 3, 2009
Outline of the Talk

1. Background and Motivation
2. Review of the Secure Shell Protocol
3. Session Resumption with Server Side State
4. Session Resumption with Client Side State
5. Performance Evaluation
6. Conclusions
IETF ISMS WG is extending SNMP so that SNMP can leverage secure transports such as SSH, TLS, DTLS, ...

Requires extensions of the RFC 3411 SNMP architecture
<table>
<thead>
<tr>
<th>Protocol</th>
<th>Time (meat)</th>
<th>Time (turtle)</th>
<th>Data</th>
<th>Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>v2c/UDP</td>
<td>1.03 ms</td>
<td>0.70 ms</td>
<td>232 bytes</td>
<td>2</td>
</tr>
<tr>
<td>v2c/TCP</td>
<td>1.13 ms</td>
<td>1.00 ms</td>
<td>824 bytes</td>
<td>10</td>
</tr>
<tr>
<td>v3/USM/UDP</td>
<td>1.97 ms</td>
<td>2.28 ms</td>
<td>668 bytes</td>
<td>4</td>
</tr>
<tr>
<td>v3/USM/TCP</td>
<td>2.03 ms</td>
<td>3.03 ms</td>
<td>1312 bytes</td>
<td>12</td>
</tr>
<tr>
<td>v2c/SSH</td>
<td>16.17 ms</td>
<td>91.62 ms</td>
<td>4388 bytes</td>
<td>32</td>
</tr>
<tr>
<td>v2c/TLS</td>
<td>18.00 ms</td>
<td></td>
<td>4109 bytes</td>
<td>16</td>
</tr>
</tbody>
</table>

- Overhead of SSH session establishment was measured using response time of an `snmpget` operation.
- **SNMPv2c/SSH** introduces significant overhead for session establishment.
- **SNMPv2c/TLS** uses less packets but exchanges similar amount of data.
- However, overhead can be amortized over long sessions...
More Recent Performance Results...

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Time (meat) [ms]</th>
<th>Time (turtle) [ms]</th>
<th>Data [bytes]</th>
<th>Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min avg max</td>
<td>min avg max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>v1/CSM/UDP/nn</td>
<td>0.24 0.25 0.29</td>
<td>0.85 0.95 1.43</td>
<td>292</td>
<td>2</td>
</tr>
<tr>
<td>v1/CSM/TCP/nn</td>
<td>0.39 0.40 0.43</td>
<td>1.27 1.38 1.72</td>
<td>1012</td>
<td>10</td>
</tr>
<tr>
<td>v2/CSM/UDP/nn</td>
<td>0.24 0.25 0.30</td>
<td>0.85 0.96 1.50</td>
<td>292</td>
<td>2</td>
</tr>
<tr>
<td>v2/CSM/TCP/nn</td>
<td>0.46 0.48 0.58</td>
<td>1.28 1.46 2.40</td>
<td>1012</td>
<td>10</td>
</tr>
<tr>
<td>v3/USM/UDP/nn</td>
<td>0.48 0.48 0.54</td>
<td>1.75 1.84 1.95</td>
<td>718</td>
<td>4</td>
</tr>
<tr>
<td>v3/USM/TCP/nn</td>
<td>0.63 0.64 0.69</td>
<td>2.22 2.46 9.59</td>
<td>1490</td>
<td>12</td>
</tr>
<tr>
<td>v3/USM/UDP/an</td>
<td>0.50 0.63 0.87</td>
<td>1.79 1.89 2.34</td>
<td>742</td>
<td>4</td>
</tr>
<tr>
<td>v3/USM/TCP/an</td>
<td>0.65 0.66 0.70</td>
<td>2.21 2.31 2.48</td>
<td>1514</td>
<td>12</td>
</tr>
<tr>
<td>v3/USM/UDP/ap</td>
<td>0.51 0.52 0.59</td>
<td>1.88 2.05 4.17</td>
<td>763</td>
<td>4</td>
</tr>
<tr>
<td>v3/USM/TCP/ap</td>
<td>0.66 0.68 0.71</td>
<td>2.31 2.42 2.60</td>
<td>1535</td>
<td>12</td>
</tr>
<tr>
<td>v3/TSM/SSH/ap</td>
<td>13.49 13.73 14.20</td>
<td>107.35 110.45 144.33</td>
<td>5310</td>
<td>31</td>
</tr>
<tr>
<td>v3/TSM/TLS/ap</td>
<td>11.01 11.15 12.57</td>
<td>67.44 68.70 86.59</td>
<td>4107</td>
<td>16</td>
</tr>
<tr>
<td>v3/TSM/DTLS/ap</td>
<td>10.89 11.05 12.00</td>
<td>67.68 69.96 155.10</td>
<td>3457</td>
<td>8</td>
</tr>
<tr>
<td>v3/TSM/TLSsr/ap</td>
<td>2.23 2.27 2.45</td>
<td>5.47 5.72 6.28</td>
<td>1457</td>
<td>15</td>
</tr>
</tbody>
</table>

- SSH (TLS/DTLS) transports behave like a DoS attack for short-lived SNMP sessions (e.g., shell scripts)
- TLS’s session resumption mechanism cures the problem
- How can we do session resumption with SSH?
## SSH Protocol Overview

### SSH Protocol Layers

1. **Transport Layer Protocol** provides server authentication, confidentiality, and integrity with perfect forward secrecy.

2. **User Authentication Protocol** authenticates the client-side user to the server.

3. **Connection Protocol** multiplexes the encrypted data stream into several logical channels.

⇒ SSH authentication is not symmetric!

⇒ The SSH protocol is designed for clarity, not necessarily for efficiency (shows its academic roots)
### Some SSH and OpenSSH Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSH Port Forwarding</strong></td>
<td>Allows users to tunnel unencrypted traffic through an encrypted SSH connection.</td>
</tr>
<tr>
<td><strong>OpenSSH SSH Agent</strong></td>
<td>Maintains client credentials during a login session so that credentials can be reused without further user interaction.</td>
</tr>
<tr>
<td><strong>OpenSSH Connection Sharing</strong></td>
<td>New SSH connections hook as a new channel into an existing SSH connection.</td>
</tr>
</tbody>
</table>
Details of an SNMP GET Operation over SSH
Server maintains session state for recently closed sessions

Client and server perform session resumption by using of a session resumption key exchange algorithm

SSH’s algorithm negotiation feature handles this nicely
Session Resumption with Server Side State

Algorithm (Server Side State)

C: Client sends the session identifier and a MAC computed over the session keys to the server in a SSH2_MSG_KEXSR_INIT message

S: Server looks up the cached session and verifies the MAC
   - If successful, it returns an SSH2_MSG_KEX_SR_OK message, followed by a standard SSH2_MSG_NEWKEYS exchange
   - On failure, SSH2_MSG_KEX_SR_ERROR is sent and key exchange proceeds with another key exchange algorithm, or fails

+ Simple design and easy to implement
- Server has to maintain session state (scalability)
Session Resumption with Client Side State

Algorithm (Client Side State)

**S:** After key (re)negotiation, the server sends an encrypted ticket in a SSH2_MSG_KEX_SR_TICKET message.

**C:** The client sends the encrypted ticket and a MAC computed over the session identifier to the server in a SSH2_MSG_KEXSR_INIT message.

**S:** The server decrypts the ticket and verifies the MAC.

- If successful, it returns an SSH2_MSG_KEX_SR_OK message, followed by a standard SSH2_MSG_NEWKEYS exchange.
- On failure, SSH2_MSG_KEX_SR_ERROR is sent and key exchange proceeds with another key exchange algorithm, or fails.

+ Server side state reduced to a key for encrypting tickets
TicketContent Data Structure

```c
struct TicketEnc {
    char* name;
    u_char* key;
    u_char* iv;
};

struct TicketMac {
    char* name;
    u_char* key;
};

struct TicketContent {
    u_char* session_id;
    u_int session_id_len;
    TicketEnc tenc_ctos;
    TicketEnc tenc_stoc;
    TicketMac tmac_ctos;
    TicketMac tmac_stoc;
    char* tcomp_ctos;
    char* tcomp_stoc;
    int hostkey_type;
    char* client_version_string;
    char* server_version_string;
};

- SSH allows to use different algorithms in each direction!
```
Ticket Data Structure

```c
struct Ticket {
    u_int seq_nr;
    u_char* id;
    u_char* enc_ticket;
    u_int enc_ticket_len;
    int64_t time_stamp;
};
```

- Contains the encrypted TicketContent data structure in `enc_ticket`
- The `id` uniquely identifies a ticket
- The `seq_nr` and `time_stamp` fields can be used to quickly discard outdated tickets
- Encryption key and its IV are generated at server start-up
**Performance Evaluation**

<table>
<thead>
<tr>
<th>Name</th>
<th>CPUs</th>
<th>RAM</th>
<th>Ethernet</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>meat</td>
<td>2 Xeon 3 GHz</td>
<td>2 GB</td>
<td>1 Gbps</td>
<td>2.6.16.29</td>
</tr>
<tr>
<td>veggie</td>
<td>2 Xeon 3 GHz</td>
<td>1 GB</td>
<td>1 Gbps</td>
<td>2.6.16.29</td>
</tr>
<tr>
<td>turtle</td>
<td>1 Ultra Sparc Ili</td>
<td>128 MB</td>
<td>100 Mbps</td>
<td>2.6.20</td>
</tr>
</tbody>
</table>

- SSH client: veggie / SSH server: meat and turtle
- Measuring overall execution time of “ssh $host exit”
- Used HMAC-MD5 hash function and AES-128 encryption
- Hosts and the network were idle during the experiments
- 1000 experiments, results sorted by the measured latency
- Absolute numbers irrelevant, look at relative numbers
Session Resumption Performance (key length 1024)

- With a key length of 1024 bits, the performance gain on an idle fast machine is observable but small.
- With the same key length, the performance gain on a small idle machine is significant (factor 4).
  - Session resumption is particularly useful for processing power constrained low-end consumer / enterprise products.
Session resumption performance is largely independent of the key length

With increasing key length, the performance gain increases also on fast idle machines

⇒ Even on a fast processors, the performance gain is significant if you need long keys to achieve strong security
Conclusions

**Contribution**
- Proposed a session resumption mechanism for SSH
- Implemented and evaluated using the OpenSSH package
- Makes SNMP over SSH viable for short-lived sessions

**Other usages**
- interactive command line completion
- system management scripts
- short lived sftp sessions
- ...
V. Marinov and J. Schönwälder.
Performance Analysis of SNMP over SSH.

J. Schönwälder and V. Marinov.
On the Impact of Security Protocols on the Performance of SNMP.
(under review), 2008.

J. Schönwälder, G. Chulkov, E. Asgarov, and M. Cretu.
Session Resumption for the Secure Shell Protocol.

H. Shacham, D. Boneh, and E. Rescorla.
Client-Side Caching for TLS.

X. Du, M. Shayman, and M. Rozenblit.
Implementation and Performance Analysis of SNMP on a TLS/TCP Base.