Session Resumption for the Secure Shell Protocol

Jürgen Schönwälder

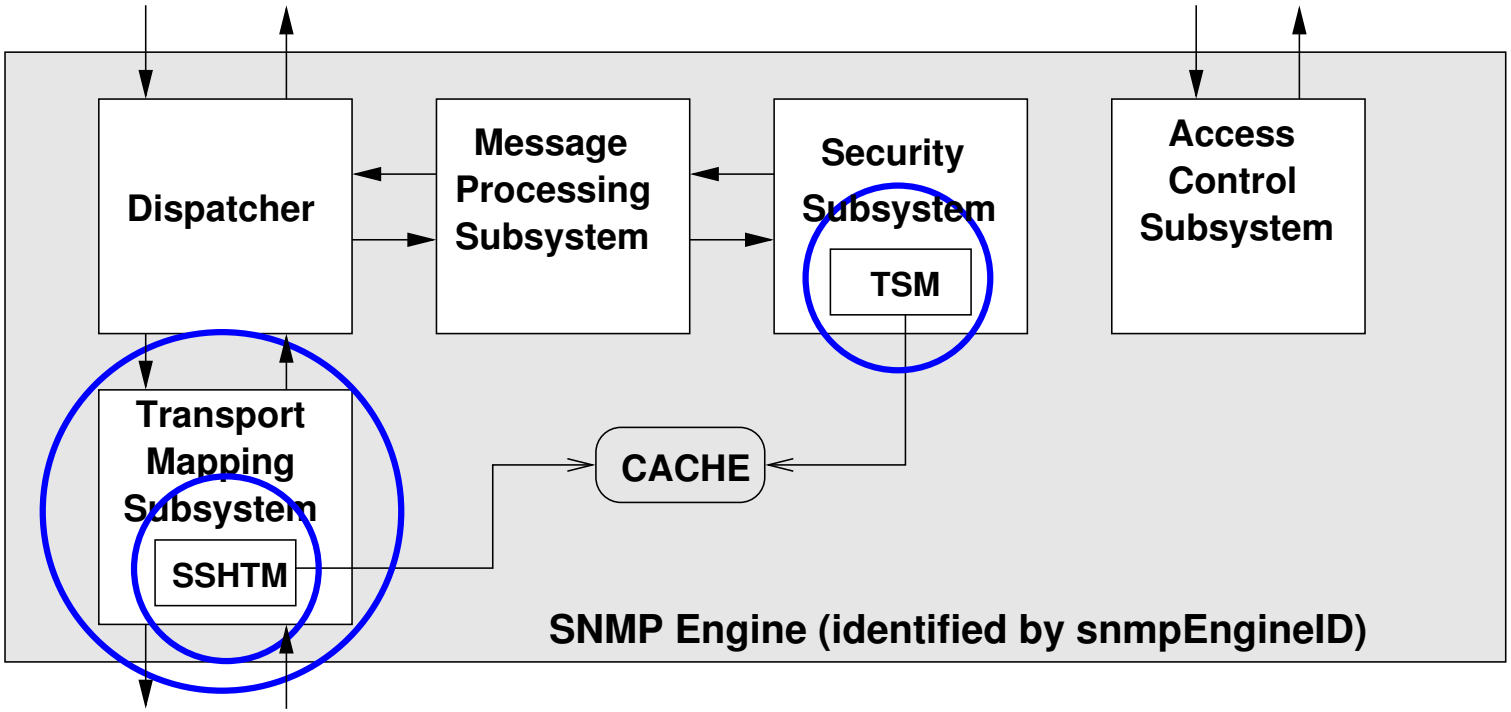
JACOBS UNIVERSITY

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

June 3, 2009

Outline of the Talk

Background: ISMS Work in the IETF



- 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

Early Performance Results (DSOM 2006)

Protocol	**Time** (`meat`)	**Time** (`turtle`)	**Data**	**Packets**
v2c/UDP	1.03 ms	0.70 ms	232 bytes	2
v2c/TCP	1.13 ms	1.00 ms	824 bytes	10
v3/USM/UDP	1.97 ms	2.28 ms	668 bytes	4
v3/USM/TCP	2.03 ms	3.03 ms	1312 bytes	12
v2c/SSH	**16.17 ms**	**91.62 ms**	**4388 bytes**	**32**
v2c/TLS	**18.00 ms**		**4109 bytes**	**16**

- 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...

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

- 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. The **Transport Layer Protocol** provides server authentication, confidentiality, and integrity with perfect forward secrecy
2. The **User Authentication Protocol** authenticates the client-side user to the server
3. The **Connection Protocol** multiplexes the encrypted data stream into several logical channels

$\Rightarrow$ SSH authentication is not symmetric!

$\Rightarrow$ The SSH protocol is designed for clarity, not necessarily for efficiency (shows its academic roots)

Some SSH and OpenSSH Features

SSH Port Forwarding

Allows users to tunnel unencrypted traffic through an encrypted SSH connection.

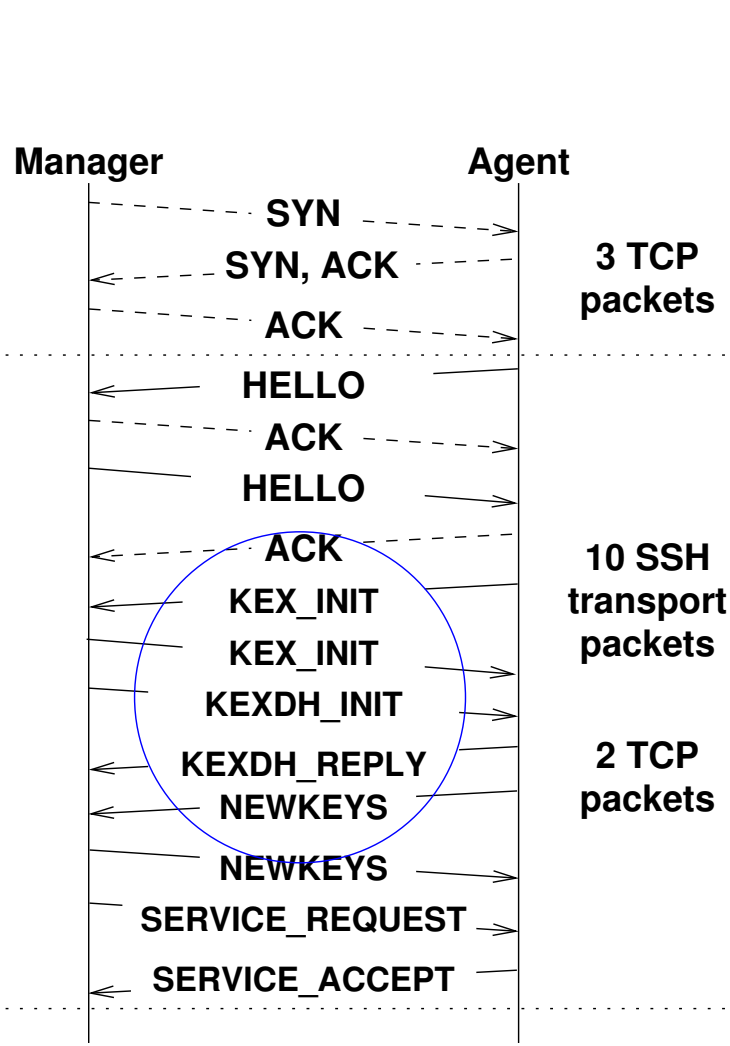
OpenSSH SSH Agent

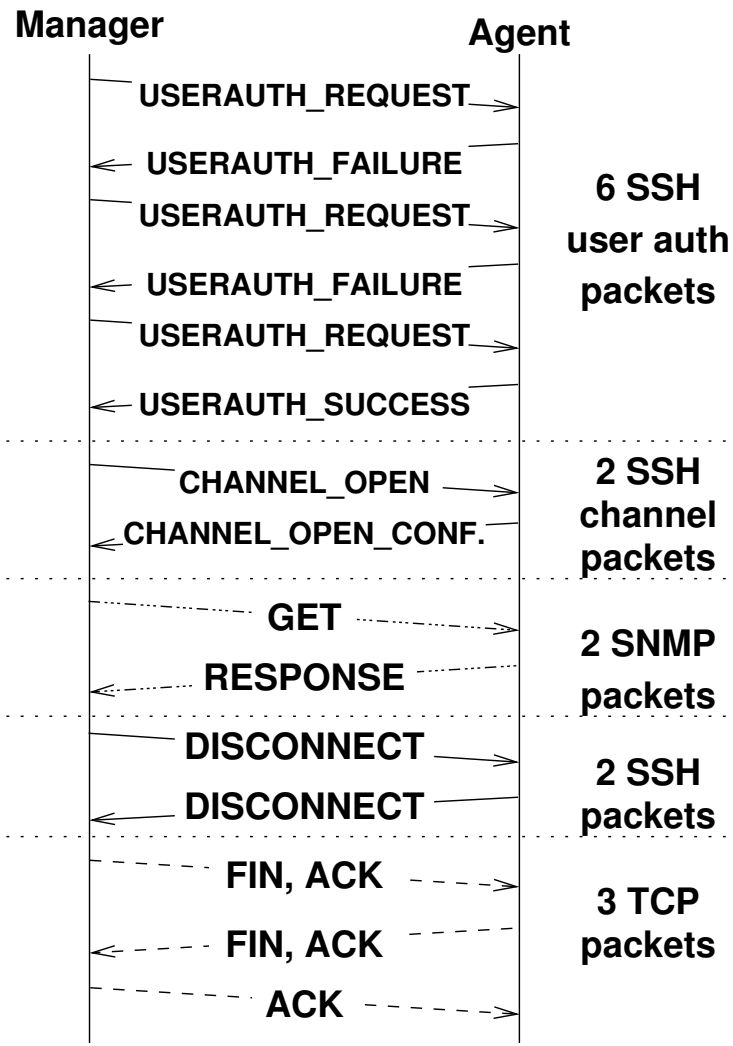
Maintains client credentials during a login session so that credentials can be reused without further user interaction

OpenSSH Connection Sharing

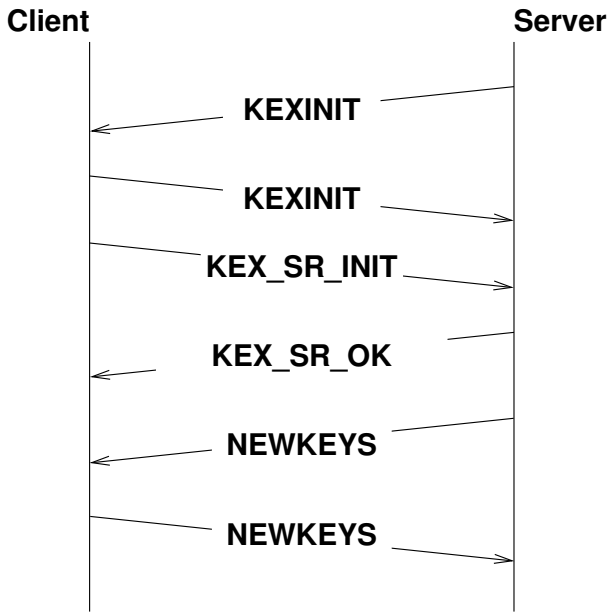
New SSH connections hook as a new channel into an existing SSH connection.

Details of an SNMP GET Operation over SSH





Session Resumption Key Exchange



- 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

```
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

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

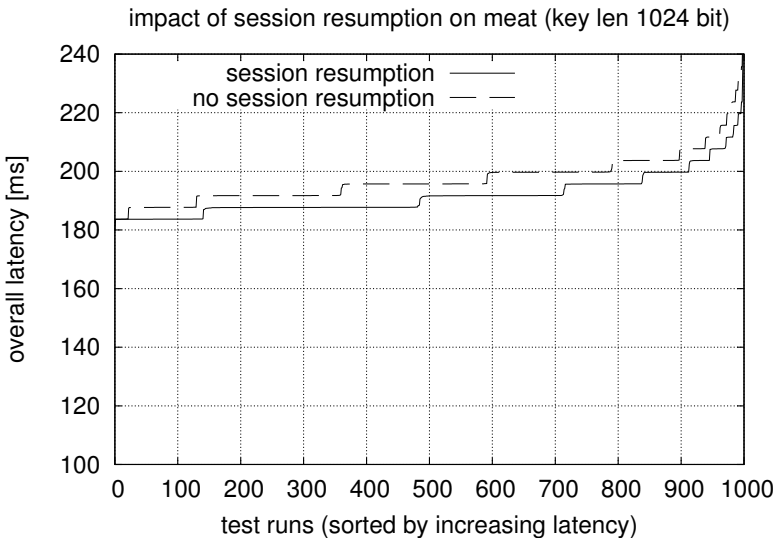
- Contains the encrypted `TicketContent` data structure in `enc_ticket`
- The `id` uniquely identifiers a ticket
- The `seq_nr` and `time_stamp` fields can be used to quickly discard outdated tickets
- Encryption key and its IV are generated at server start-up

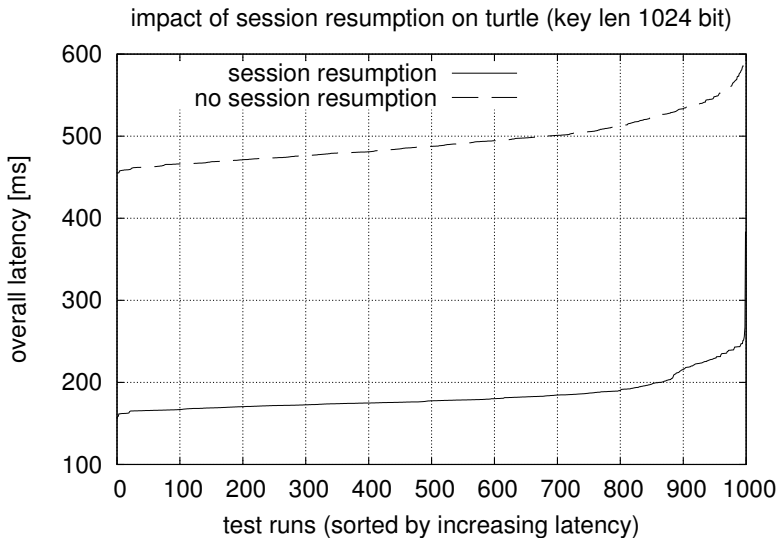
Performance Evaluation

Name	CPUs	RAM	Ethernet	Kernel
`meat`	2 Xeon 3 GHz	2 GB	1 Gbps	2.6.16.29
`veggie`	2 Xeon 3 GHz	1 GB	1 Gbps	2.6.16.29
`turtle`	1 Ultra Sparc IIi	128 MB	100 Mbps	2.6.20

- 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)

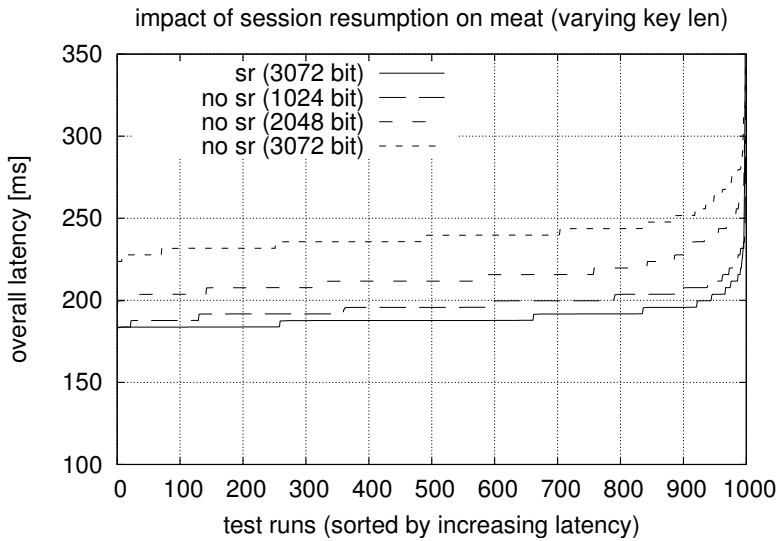




- With a key length of 1024 bits, the performance gain on an idle fast machine is observable but small
- With the same key length, the performance gain on a small idle machine is significant (factor 4)

$\Rightarrow$ Session resumption is particularly useful for processing power constrained low-end consumer /enterprise products

Impact of the Key Length on the Performance



- Session resumption performance is largely independent of the key length
- With increasing key length, the performance gain increases also on fast idle machines

$\Rightarrow$ 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
- ...

References

V. Marinov and J. Schönwälder.

Performance Analysis of SNMP over SSH.
In *Proc. 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2006)*, number 4269 in LNCS, pages 25–36, Dublin, October 2006. Springer.

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.
In *Proc. 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*, May 2009.

H. Shacham, D. Boneh, and E. Rescorla.

Client-Side Caching for TLS.
ACM Transactions on Information and System Security, 7(4):553–575, November 2004.

X. Du, M. Shayman, and M. Rozenblit.

Implementation and Performance Analysis of SNMP on a TLS/TCP Base.
In *Proc. 7th IFIP/IEEE International Symposium on Integrated Network Management*, pages 453–466, Seattle, May 2001.