Eliciting a Sensemaking Process from Verbal Protocols of Reverse Engineers

Adam R. Bryant[1,2] **(adam.bryant@wpafb.af.mil)**, **Robert F. Mills**[2] **(robert.mills@afit.edu)**,
Gilbert L. Peterson[2] **(gilbert.peterson@afit.edu)**, and **Michael R. Grimaila**[2] **(michael.grimaila@afit.edu)**
[1]Cognitive Models and Agents Branch, 711th Human Performance Wing
[2]Department of Electrical and Computer Engineering, Air Force Institute of Technology
Wright-Patterson AFB, OH 45433 USA

Abstract

A process of sensemaking in reverse engineering was elicited from verbal protocols of reverse engineers as they investigated the assembly code of executable programs. Four participants were observed during task performance and verbal protocols were collected and analyzed from two of the participants to determine their problem-solving states and characterize likely transitions between those states. From this analysis, a high-level process of sensemaking is described which represents hypothesis generation and information-seeking behaviors in reverse engineering within a framework of goal-directed planning. Future work in validation and application of the process is discussed.

Keywords: Sensemaking; Information seeking; Human computer interaction; verbal protocol studies.

Introduction

Sensemaking is a term used to describe a broad family of cognitive activities in which a person comes to develop a mental model to represent the elements of some situation of interest (Klein, Phillips, Rall, & Peluso, 2007; M. R. Endsley, 2000). Whereas situation awareness refers to the ability to attend to relevant information elements in a task as a situation unfolds (M. Endsley & Rodgers, 1994), sensemaking refers to the process that enables one to come to an understanding of the meaning and relevance of the elements that make up the situation (Klein, Moon, & Hoffman, 2006; M. R. Endsley, 2000). The sensemaking process has been described as an ongoing integration of knowledge from a mental model of a situation, available information about the context of a situation, and perceptual data from the environment (D. M. Russell, Stefik, Pirolli, & Card, 1993; M. R. Endsley, 2000).

Sensemaking is described as connecting inferences and observations, integrating knowledge and conjecture, finding explanations for ambiguous data, diagnosing ambiguous symptoms, and identifying problems (Klein et al., 2007). Sensemaking also refers to comprehension of the significance of ambiguous events and data in the environment (Weick, 1995). The many functions of sensemaking describe a class of distinct cognitive processes involving interactions between knowledge, information, and actions in an environment which may encompass a number of different inference and learning behaviors (Menzies, 1996; Josephson, Chandrasekaran, Smith, & Tanner, 1987). Sensemaking has been studied in naturalistic settings (Klein et al., 2007), in computer-human interaction (Pirolli & Card, 2005), in intelligence analysis (Zhang, Soergel, Klavans, & Oard, 2009), and in organizations (Weick, 1995), but no process-level description exists to characterize how people make sense of executable programs.

As part of a larger study to understand and model how people make sense of programs from executable representations, we wanted to understand the general interaction process that is involved as a person works with a debugger to make sense of a program. While sensemaking might be a general process of interacting with an environment to develop a mental model, we investigated sensemaking in a particular applied domain (reverse engineering assembly language) because it provides a restricted set of semantics to ground the investigation without being a "toy" problem. This was also essential because understanding sensemaking in that domain was the major focus of the larger study. So while the term 'sensemaking' has been used to mean many different things, we were interested in studying how people connect information from the task environment with background knowledge in order to develop and modify a mental model, all with a focus on applying this process to improve the information processing automation in software reverse engineering tools.

To understand this element of understanding executable programs, we undertook a study to collect verbal protocols from reverse engineers while they analyzed programs from assembly language representations. Four participants were observed performing a challenging reverse engineering task, and verbalizations from two of the participants were transcribed and coded. From the coded data, two participants' state transitions were extracted to determine a process involving the state transitions.

First, the methods used in the study are briefly described. Next, the methods used in the analysis of verbal protocols is presented. Following that, the process of sensemaking used in the reverse engineering tasks is described in the context of complex problem-solving.

Method

The participants were instructed to complete a problem called *Angler* that was downloaded from the *crackme.de* website (Schneider, T., 2011) in February 2011[1]. Angler is a type of crackme called a *keygen* (which stands for key generation), a type of program that typically presents text fields for the user to enter a name and a serial number and asks the participant to reverse engineer the algorithm which processes the user's data. In keygen tasks, people disassemble and reverse engineer the program to discover a valid key for a given user

[1]Shortly after obtaining the crackme, the crackme.de website was taken down from the Internet. The Angler program can also be obtained from the website of the program's author at *http://cyclops.ueuo.com*.

name or to write an algorithm that, when given a user name, produces a valid license key that unlocks the functionality of the program.

The Angler crackme program presents the participant with a semi-transparent visual window with two entries for text. It has three buttons, labeled "Check", "About", and "Exit." When a user submits a name and serial number combination, the application tests whether it is a valid combination and informs the user.

Selection of Participants

The researcher solicited reverse engineers to participate in a verbal protocol study through an e-mail invitation to the Air Force Institute of Technology and to a cross-organizational reverse engineering working group at Wright-Patterson Air Force Base. The solicitation requested that participants have knowledge of reverse engineering and experience using tools such as OllyDbg, WinDbg, Immunity Debugger, and the IDA Interactive Disassembler. The solicitation produced four reverse engineers from Wright-Patterson Air Force Base who participated without remuneration.

Data Collection

Video data of the participants' computer screen was captured and combined with audio recordings of the participants' verbalizations during the task. The participant computer ran a Windows XP operating system that was hosted within a VirtualBox virtual machine, and which was preloaded with the software tools and documentation identified from a previous subject matter expert study of the reverse engineering task. The experimenter viewed the participant's task environment remotely over a virtual network computing (VNC) connection using the TightVNC server and client software (TightVNC Software, 2011). The experimenter's computer was outfitted with a microphone which was wired into the participant's cubicle to enable the combined collection of audio and video data.

Each participant was seated at a small cubicle in an unoccupied, quiet room in front of the participant computer which contained a mouse, keyboard, monitor, and the microphone. The participants were instructed as to the different reverse engineering tools and documentation available, and were permitted as much time as was needed to become familiar with the task environment. Paper was also available so participants could make notes, as recommended by Wood (1997).

Each participant was given instructions on how to verbalize thoughts during task performance and was instructed not to try to explain the task or thought processes. The experimenter stressed this point and demonstrated examples of poor, acceptable, and high-quality concurrent verbalizations with a simulated coffee-making task to help the participants understand how they should verbalize during task performance. During the performance of each task, the experimenter was seated out of the participant's view in the opposite cubicle and reminded participants to verbalize when they fell silent for more than a few seconds using simple prompts such as "please remember to verbalize during the task" and "remember to talk aloud" as discussed in Trickett and Trafton (2007). Other than those reminders, the experimenter was silent throughout the task, and took notes about each participants' goals, strategies, concepts, and problems in the task.

Audio data of each participant's concurrent verbalizations and video data of the experimenter's computer monitor (showing all actions on the participant's computer monitor) were recorded using the CamStudio software. After the task, participants were asked to recall what they thought their strategies were, what parts of the task they thought were the most difficult, what would have made the task easier, and what they felt they needed to pay attention to.

Overview of the Reverse Engineering Task

Participants were to investigate, modify, and re-implement an algorithm that an executable Windows program called "Angler.exe" (Cyclops, 2011) uses to process a user-provided serial number. Participants worked from assembly language representations of the program provided through the IDA interactive disassembler and the OllyDbg debugger without access to source code or debugging symbols.

The code in Angler consists of 15 major subroutines, including the subroutine called `WinMain` which starts the windowing process and other subroutines to present dialog boxes. The program runs within a single thread of execution in memory and does not have hidden sections, encrypted code, or code obfuscations. Angler's file header contains pointers to four program sections that are mapped into memory at run time: the `.text`, `.rdata`, `.data`, and `.rsrc` sections, which are typical for portable executable programs compiled to run on Win32-based operating systems (Eilam, 2005).

Though there are many strategies to approaching the challenge, successful completion required that participants do the following:

1. Read and understand the goal of the task
2. Determine that a system function handles input
3. Isolate the input handling function
4. Determine the format for the serial number input
5. "Catch" the input as the program executes
6. Craft data so the program executes the success message
7. Translate the function into pseudocode
8. Write pseudocode for a key generator

An algorithm in the program composed of 27 basic blocks of assembly instructions processes the user's serial number. The algorithm takes the first four characters of the person's name and performs a cyclic redundancy check (CRC) to produce an even-numbered value from that character, finds four pairs of prime factors that sum to the even-numbered values, and assembles an eight-value number separated by dashes (Cyclops, 2011). Through a series of programmatic checks, the algorithm determines if the user's serial number matches the computed serial value. In these checks, the last instructions in each basic block of code check for a data value from

Verbalization Describes	Coded As
Desired future state	Make goal representation
Activities to accomplish goal	Plan approach
Status of an ongoing activity	Carry out plan
Noticing something	Sense information
Recognizing relevance	Interpret information
More abstract statement	Update knowledge
An assumption	Create hypothesis
Question about something	Create hypothesis

Table 1: Rules Used to Code Segments

the input and then transfer the program's execution based on the result of that check. If the reverse engineer does not understand the meanings of these behaviors, the program appears to be making a large number of arbitrary numerical checks in a long sequence of assembly language instructions.

Verbal Protocol Analysis

Participant B's video and audio data recordings were accidentally destroyed during a problem with saving the video file to disk and were not able to be recovered or transcribed. Additionally, Participant A lacked familiarity with the tools and the task and was not able to make sense of the program. Although observations of all of the participants provided valuable context and examples, only verbal data from Participants C and D were coded and analyzed to elicit the sensemaking process.

Concurrent verbal protocols were collected from Participants C and D following the method outlined in (Ericsson & Simon, 1980). We reviewed the video and verbal data from Participants C and D and transcribed it into a spreadsheet, broken into one verbal segment per row as discussed in Trickett and Trafton (2007). The participants' verbalizations were segmented during transcription in order to take advantage of other contextual clues from the audio and video. Verbalizations were segmented based on whether they represented a single idea, and when a segment contained a shift from one idea to another, the second idea was recorded in its own row as its own new segment. Where significant verbal breaks occurred, the subsequent verbalizations were recorded on a new row as a separate segment.

After all of the available data was transcribed, it was coded according to the following taxonomy of sensemaking in reverse engineering, established from a previous literature review of sensemaking in reverse engineering in Bryant et al. Bryant, Mills, Peterson, and Grimaila (2012): *Make the goal representation*, *Plan an approach*, *Carry out a plan*, *Sense information*, *Interpret information*, *Update knowledge*, and *Create a hypothesis*. This taxonomy of sensemaking steps is similar to the information-processing loop used in programming artificial agents to interface with an environment (S. Russell & Norvig, 2003), with the addition of a state to generate goals and state to generate hypotheses.

The data from the two participants were coded according to the coding rules in Table 1. To ensure that the coding scheme was appropriate for the data, interrater reliability was computed between two independent coders. One researcher coded all of the data (592 segments) and afterwards a second coder independently coded 29.2 percent of the segments (173 sequential segments) without having seen the original coder's data, and from a starting point randomly selected by the second coder. Cohen's Kappa statistic (Cohen et al., 1960) was computed to measure interrater reliability for the 29.2 percent of segments coded by both coders. Cohen's Kappa measures the agreement between coders on positive and negative instances while taking into account the likelihood of agreement based on chance. Cohen's Kappa is computed as:

$$\kappa = (P_o - P_c)/(1 - P_c)$$

P_o is the proportion of agreements between the coders and P_c is the proportion of agreement which would be predicted by chance. Generally, a Cohen's Kappa value of 0.0 to 0.4 indicates zero to very little agreement, 0.6 to 0.8 indicates significant agreement, and 0.8 and above represents near perfect agreement, though there is disagreement in the literature about specific ranges of values (Bakeman & Gottman, 1997; Trickett & Trafton, 2007). After both coders independently coded the data, the interrater reliability was calculated and the coders met to discuss disagreements. If codes had weak interrater reliability (0.4 or below), the categories were removed or changed and the data was recoded. The final interrater reliability for the dual-coded verbalizations was 0.82, which demonstrates significant to "near perfect" agreement in all categories (Table 2). Following standard practice, the remainder of the verbalizations were coded by the researcher (Trickett & Trafton, 2007).

Computing State Transitions

The state transitions from the two participants' data were computed to determine how the reverse engineers made sense of the programs. Transitions between the states indicated movement through the problem-solving process

State	Category	Cohen's Kappa
a	Make goal representation	0.93
b	Plan approach	0.82
c	Carry out plan	0.78
d	Sense information	0.72
e	Interpret information	0.75
f	Update knowledge	0.81
g	Create hypothesis	0.92
	Average agreement	0.82

Table 2: Interrater Reliability of Coding Scheme (173 segments)

As described in Bakeman and Gottman (1997), matrices of state transition probabilities were computed to determine the process used in the task. For m states S_j and S_k, and n seg-

ments i, the total transitions between each state S_j and state S_k are computed as:

$$Tr(S_j, S_k) = \sum_{i=1}^{n} (S_{i,j} \times S_{i+1,k}) : S \in (0,1) \quad (1)$$

The total transitions departing a state S_j are computed as:

$$Tr(S_j, out) = \sum_{k=1}^{m} Tr(S_j, S_k) \quad (2)$$

The total transitions entering a state S_k are computed as:

$$Tr(in, S_k) = \sum_{j=1}^{m} Tr(S_j, S_k) \quad (3)$$

The overall transition probabilities for state S_j to S_k are computed as:

$$P(Tr(S_j, S_k)) = \frac{1}{2}\left(\frac{Tr(S_j, S_k)}{Tr(S_j, out)} + \frac{Tr(S_j, S_k)}{Tr(in, S_k)}\right) \quad (4)$$

Using these equations to compute the state transitions, the state transition probabilities for the two participants are shown in Table 3 and Table 4.

	a	**b**	**c**	**d**	**e**	**f**	**g**
a	0.17	0.24	0.10	0.10	0.15	0.00	0.04
b	0.14	0.20	0.21	0.21	0.03	0.00	0.12
c	0.10	0.13	0.05	0.33	0.09	0.10	0.00
d	0.10	0.08	0.12	0.37	0.37	0.10	0.14
e	0.00	0.11	0.25	0.22	0.25	0.28	0.18
f	0.05	0.00	0.00	0.10	0.10	0.21	0.29
g	0.23	0.15	0.07	0.10	0.17	0.07	0.38

Table 3: Transition Probability Matrix (Participant C)

	a	**b**	**c**	**d**	**e**	**f**	**g**
a	0.21	0.26	0.22	0.15	0.14	0.00	0.04
b	0.18	0.19	0.26	0.18	0.09	0.00	0.11
c	0.08	0.12	0.07	0.22	0.11	0.10	0.13
d	0.08	0.21	0.09	0.45	0.28	0.18	0.18
e	0.18	0.06	0.08	0.22	0.24	0.20	0.03
f	0.13	0.03	0.07	0.12	0.10	0.31	0.16
g	0.25	0.07	0.04	0.15	0.03	0.08	0.10

Table 4: Transition Probability Matrix (Participant D)

The mean transition probabilities were computed as $1/N\sum_i^N P(Tr(S_j, S_k))$ for N participants. The mean transition probability was $\mu = 0.14$ and the standard deviation was $\sigma = 0.09$. The threshold for significance was set at $\mu + \sigma = 0.23$. The significant transitions at the threshold $P(Tr) \geq 0.23$ are shown in Figure 1.

Because only two participants' verbalizations were coded, inferences cannot be made about the how the processes from these two samples apply to the broader population of reverse engineers or the broader sensemaking process. Nevertheless, previous studies have used observations and verbal data from small samples during exploratory research as way to generate hypotheses which are to be verified with further investigation and more participants (Newell & Simon, 1972; Trickett & Trafton, 2007). The verbal protocols from the two participants and the observed problem-solving processes from all four participants are useful to provide a framework for understanding how reverse engineers make sense of executable programs.

Discussion

Figure 1 shows a process of how the sensemaking behaviors were used by reverse engineers attempting to solve the Angler task. When the participants were working on problems in the task, they continually moved through a loop of activities, which included the establishment of a goal representation, a plan to achieve the goal, carrying out the actions of the plan, sensing information from the task environment, interpreting the information, potentially updating knowledge if the information was relevant, and developing hypotheses based on the new knowledge. The sensemaking loop in Figure 1 shows this cycle as a Markov model populated with the probabilities that each state transition would occur.

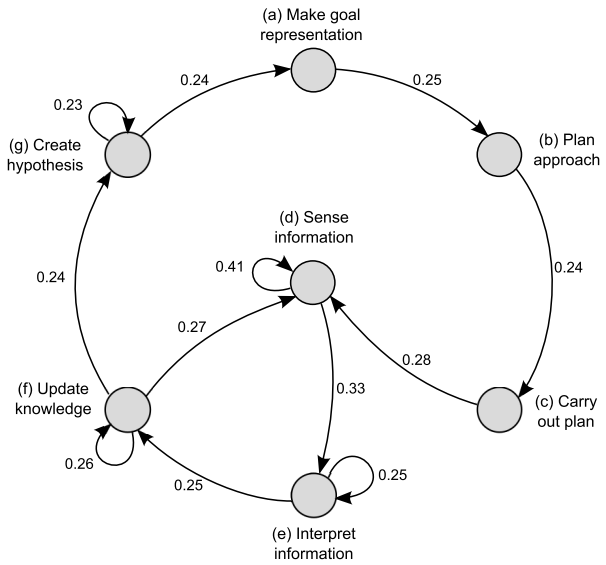

Figure 1: Sensemaking Processes from the Reverse Engineering Task

While progressing through this process, the participants gathered information to help them construct and refine their mental models of the Angler program. As the participants worked on the problem, they gained information about components such as functions, the program's execution paths, data in the program, and sequences of instructions. They were then able to relate these elements to items in the task environment.

Plans, Goals, and Actions

Once the participants had verbalized a goal, this often was immediately followed with the development of a partially-constructed plan of actions which would enable the attainment of the goal. If the goal was to gather information from the task environment, the plan involved actions which the person could use to help gather the required information. Likewise if the goal was to configure the program or the reverse engineering tools in a particular way, then the plan involved sequences of actions which led the person to be able to change elements in the situation.

In some cases, a participant's goal did not directly lend itself to a plan, so the person deliberated over different ideas to construct and evaluate an approach that would generate a usable plan. Sometimes the deliberation was verbalized, and others it was inferred by the presence of long pauses.

Participants appeared to determine the best actions for their situation by thinking through hypothesized behaviors and inferred future states of the program. When participants verbalized plans, their plans involved sets of actions in which some of the actions were sequenced. Sometimes the participants did not order their actions into a sequence until they were already taking some initial actions and detecting conflicts in how those actions would affect the state of the situation.

Other plans the participants expressed included searching information from the reverse engineering tool about text strings, debugging the program to see how the program's memory stack changes, labeling a function so it can be identified later, inserting a breakpoint after the initialization routine to "catch" the program, and tracing data as it flows through a function during simulated program execution.

Information Processing and Interpretation

Participants processed information by actively seeking it out and by passively noticing information during the performance of some other action. When participants passively sensed the information, they executed the Angler program to gather information about its behaviors or looked through the disassembled code to gather clues that might be useful.

When participants actively sought out information, they set a goal for the information they were interested in obtaining, made a plan to acquire that information, and followed the plan by carrying out actions in the task environment. In one example, a participant wrote down the addresses of system calls and used the search features of the debugger to find each of the calls.

Participants actively sought out information about the program's behavior by isolating phenomena. To do this, participants used the debugger to move the program's execution past a system call (which performs some action for the program), and then looked through the current register values and disassembled code in the debugger to determine what had changed in the state of the program.

In either active or passive information processing, the participants seemed to perform some initial processing to determine whether the information was relevant to one of their goals, and consequently ignored or attended to the information. If participants deemed the information as relevant, they interpreted it to connect it to concepts they already understood. If the information element and its connected concept provided the person with a new perspective or more insight that was relevant to one of their goals, the person verbalized a phrase indicating they had updated their knowledge. These phrases were typically summarized or distilled statements that captured the essense of how the different concepts were related.

During coding discussions, the second coder characterized this process as when the person "compiled" their information in an analogous manner to how a compiler converts a program's source code into executable code.

Mental Models and Hypotheses

Once participants had added new summarized knowledge to their mental model of a program, they often came up with a hypothesis directly afterward. Participants appeared to generate hypotheses in the task after deducing the logical conclusions of new knowledge they had acquired.

Hypotheses and assumptions were generated mainly after participants sensed and interpreted information and updated their knowledge with the implications of this information. The participants' hypotheses took the form of verifiable statements such as: "it looks like `GetDialogItem` creates a handle to some part of the dialog that's open." In this case, the participant started a subsequent loop through the sensemaking process with the goal of verifying whether or not the `GetDialogItem` function creates such a Window handle.

The hypotheses that resulted from this process were typically used to generate a new goal, such as to seek out information from the environment to confirm or refute a fact or to investigate another line of investigation about how the program works. However, when participants did not progress through the process to the development of a hypothesis, they were not able to generate information-seeking goals and got "stuck" in the task. In these cases, participants reverted to exploring instructions or behaviors of the code, since they did not have specific hypotheses about the Angler program to investigate.

Conclusions

A process of sensemaking in reverse engineering was described and characterized through obervations and analysis of verbal protocols from participants reverse engineering programs from assembly language representations. Verbal protocols were analyzed to extract state transition patterns from two reverse engineers' performance, and from that data a process of sensemaking was elicited and the steps of that process were described in a theory of how people make sense of executable programs. Participants were observed forming goals, creating plans to achieve their goals, and carrying out plans. Participants also sensed information from the task environment, interpreted the information, updated their mental

model with the information, and generated hypotheses from that integration which led to new goals.

This research represents a necessary step toward increasing the autonomy of reverse engineering tools, and can be used to determine a general theory of sensemaking that can improve the ways in which people interact with other complex systems. The work is limited in that it only involved observations of four participants and collection of verbal protocols from two participants, and its generality is potentially limited by the nature of the task; nevertheless the results provide insight which can be used to develop a more general computational theory of sensemaking through future empirical study.

Future work is needed to determine the generality of how people work through this process in similar information-processing tasks. Future research is also needed in determining how these processes can be realized as models within established computational cognitive architectures. Finally, work is needed in employing these findings to improve humans' interactions with complex systems such as reverse engineering tools.

Acknowledgments

The Sensors Directorate at Wright-Patterson Air Force Base supported this research. The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

References

Bakeman, R., & Gottman, J. M. (1997). *Observing interaction: An introduction to sequential analysis* (2nd ed.). Cambridge: Cambridge University Press.

Bryant, A., Mills, R., Peterson, G., & Grimaila, M. (2012). (in press) software reverse engineering as a sensemaking task. *Journal of Information Assurance and Security*.

CamStudio Developers. (2011). *Camstudio.* Available from `http://camstudio.org`

Cohen, J., et al. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, *20*(1), 37–46.

Cyclops. (2011). *Crackmes by cyclops.* Available from `http://cyclops.ueuo.com/crackme.html`

Eilam, E. (2005). *Reversing: Secrets of Reverse Engineering*. Wiley.

Endsley, M., & Rodgers, M. (1994). Situation awareness information requirements analysis for en route air traffic control [Conference proceedings (article)]. In *Proceedings of the Human Factors and Ergonomics Society 38th Annual Meeting* (Vol. 38, pp. 71–75).

Endsley, M. R. (2000). Situation models: An avenue to the modeling of mental models. In *Proceedings of the Human Factors and Ergonomics Society 44th Annual Meeting*.

Ericsson, K., & Simon, H. (1980). Verbal reports as data. *Psychological Review*, *87*(3), 215.

Hex-Rays. (2011). *The IDA Pro Disassembler and Debugger.* Available from `http://www.hex-rays.com/idapro/`

Immunity, Inc. (2011). *Immunity Debugger.* Available from `http://www.immunityinc.com/products-immdbg.shtml`

Josephson, J. R., Chandrasekaran, B., Smith, J. W., & Tanner, M. C. (1987). A mechanism for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man, and Cybernetics*, *17*(3), 445–454.

Klein, G., Moon, B., & Hoffman, R. (2006). Making sense of sensemaking 1: Alternative perspectives. *IEEE Intelligent Systems*, *21*(4), 70–73.

Klein, G., Phillips, J., Rall, E., & Peluso, D. (2007). A data-frame theory of sensemaking. In *Expertise Out of Context: Proceedings of the Sixth International Conference on Naturalistic Decision Making* (pp. 113–155).

Menzies, T. (1996, September). Applications of abduction: Knowledge-level modelling. *International Journal of Human-Computer Studies*, *45*(3), 305–335.

Microsoft, Inc. (2011). *WinDbg.* Available from `http://msdn.microsoft.com/en-us/library/ff561300(v=vs.85).aspx`

Newell, A., & Simon, H. (1972). *Human Problem Solving*. Prentice-Hall Englewood Cliffs, NJ.

OllyDbg. (2011). *OllyDbg.* Available from `http://www.ollydbg.de/`

Pirolli, P., & Card, S. (2005). The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of the International Conference on Intelligence Analysis* (pp. 2–4).

Russell, D. M., Stefik, M. J., Pirolli, P., & Card, S. K. (1993). The cost structure of sensemaking. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI93)* (pp. 269–276). New York, New York, USA: ACM Press.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education.

Schneider, T. (2011, jan). *crackme.de.* Available from `http://crackme.de`

TightVNC Software. (2011). *TightVNC.* Available from `http://www.tightvnc.com`

Trickett, S., & Trafton, J. (2007). A primer on verbal protocol analysis. In D. Schmorrow, J. Cohn, & D. Nicholson (Eds.), *The PSI Handbook of Virtual Environments for Training and Education: Developments for the Military and Beyond.* Westport, CT Praeger Security International.

Weick, K. (1995). *Sensemaking in Organizations*. Sage Publications, Inc.

Wood, L. E. (1997, March). Semi-structured interviewing for user-centered design. *Interactions*, *4*(2), 48–61.

Zhang, P., Soergel, D., Klavans, J. L., & Oard, D. W. (2009, June). Extending sense-making models with ideas from cognition and learning theories. *Proceedings of the American Society for Information Science and Technology*, *45*(1), 23–23.