An Exploratory Study of Software Reverse Engineering in a Security Context

Christoph Treude, Fernando Figueira Filho, Margaret-Anne Storey
Dept. of Computer Science, University of Victoria
Victoria, BC, Canada
ctreude@uvic.ca, ffilho@uvic.ca, mstorey@uvic.ca

Martin Salois
Defence Research and Development Canada – Valcartier
Quebec, QC, Canada
martin.salois@drdc-rddc.gc.ca

***Abstract*—Illegal cyberspace activities are increasing rapidly and many software engineers are using reverse engineering methods to respond to attacks. The security-sensitive nature of these tasks, such as the understanding of malware or the decryption of encrypted content, brings unique challenges to reverse engineering: work has to be done offline, files can rarely be shared, time pressure is immense, and there is a lack of tool and process support for capturing and sharing the knowledge obtained while trying to understand plain assembly code. To help us gain an understanding of this reverse engineering work, we report on an exploratory study done in a security context at a research and development government organization to explore their work processes, tools, and artifacts. In this paper, we identify challenges, such as the management and navigation of a myriad of artifacts, and we conclude by offering suggestions for tool and process improvements.**

I. Introduction

In his 1987 article [1], Cohen coined the term "computer virus" to describe self-reproducing programs designed to infect other computer programs. At that time, computer viruses were created for experimentation purposes or merely for fun, therefore causing little to no damage to real world systems [2].

Today's landscape shows us a different scenario. Computers are widely used in criminal activities such as bank fraud, identity theft, and corporate theft. According to a recent Symantec report [3], 2010 saw an average of 260,000 identities exposed in data breaches caused by hacking, and 42% more vulnerabilities related to mobile platforms—up to 163 in 2010 from 115 in 2009.

Illegal activities in cyberspace affect national security and threaten citizen's rights and privacy, thus having significant political, economic, and social implications [4]. Sponsored by organized crime or entrepreneurial goals, hackers have been developing malicious software, or "malware". Forms of malware are not limited to viruses, but can also include worms, trojan horses, and spyware, all of which can install themselves on a computer without the computer owner's informed consent.

Organized cyber groups typically communicate using cryptographic protocols and store information using encrypted files or systems. As a countermeasure against cybercrime, government institutions and business organizations have been using reverse engineering methods to analyze malicious code and break into password protected file systems.

This paper reports the findings of a field study conducted with security engineers working in a research and development government organization. We consider their setting and work context to be unique for the following reasons. First, time sensitivity is an issue when dealing with malware. Software reverse engineers may have to quickly analyze and understand malicious code to provide a fast response on how to block and remove a particular piece of malware. Second, co-workers may have to be off the network as a way to mitigate the risk of infecting other computers and to protect sensitive information regarding system vulnerabilities. As such, coordinating work among team members is a challenging issue, even when co-located, since air gaps are required when exchanging information, which limits the use of modern collaborative tools. Third, documenting the reverse engineering process is a great challenge as engineers use different tools and typically create artifacts for their own use (e.g., data flow and sequence diagrams). As a result, transferring knowledge is a hard task, which may require a great deal of face-to-face communication. Overall, security reverse engineers have special needs in terms of time sensitivity, coordination, communication, and documentation.

To the best of our knowledge, there are no studies which have investigated the work practices of security reverse engineers. Nonetheless, previous work has proposed reverse engineering tools for computer security analysis. Cifuentes *et al.* [5] introduced a high-level debugging tool which aims to reduce the amount of time needed to solve security-related problems. However, malware runtime analysis using a typical debugger is often not possible right away, mainly because modern malware usually has plenty of anti-debugging tricks that must be removed first. Quist and Liebrock [6] presented a visualization tool for malware analysis but further work is needed to investigate how such tools would meet our participants' needs. Powerful tools for program comprehension are important, but visualization is just part of the security engineer's needs, as we will present in this paper.

II. Research Questions

To gain a comprehensive understanding of software reverse engineering in a government security context, our research questions focus on processes, tools, and artifacts:

1) What processes are part of reverse engineering in a security context?
2) What tools are being used?
3) What artifacts are being created and shared?

III. Methodology

The research was conducted as an exploratory qualitative study. We conducted seven semi-structured interviews with engineers at a research and development government organization. For the remainder of this paper, we use P1 to P7 to refer to the participants of our study. Their main task for this study was the understanding of targeted malware. This often goes beyond the work done by anti-virus companies, such as Symantec[1], as it involves identifying the perpetrators as much as possible in order to identify the type of attack as well as its political and social implications. We found two roles related to software reverse engineering: reverse engineers (P1, P2, P3, P6 and P7), and developers building specialized tools to support reverse engineers (P4 and P5). The latter was mainly tasked with creating plugins, particularly to automate discoveries of vulnerabilities such as buffer overflows. Each interview lasted approximately half an hour and was conducted in the participants' workplace. In the beginning of each session, we asked participants about their roles and main responsibilities. We then used a list of questions to explore our research problem[2]. Those questions were interleaved with emergent questions based on our interviewees' answers. Two researchers took notes in parallel during the interviews and notes were verified between the note takers. We then used an inductive approach to analyze the collected data. Two of the authors followed an open coding strategy for labeling and categorizing data. The later stages of analysis consisted of deriving emergent themes by drawing connections between codes. There was no predefined theory prior to the analysis. A draft version of this document was given to two members of the organization under study to verify our findings.

IV. Findings

In this section, we present our findings, subdivided for each research question posed in Section II.

A. Processes

Based on the interview data, we identified five processes that are part of reverse engineering in a security context.

1) Analyzing: Analyzing assembly code is at the heart of most reverse engineering projects. Typical projects include the detection of malware, such as trojan horses, or the decryption of encrypted file systems. Assembly code is more difficult to understand than source code written in high-level programming languages because the code is less structured, often lacks meaningful symbols or data definitions, and allows for tricks that can mislead reverse engineers in their analysis efforts. Following the flow of data is challenging: *"Understanding the data flow is a big part of understanding a program."*$_{P4}$

2) Documenting: Documenting reverse engineering has several purposes. Some documentation is done to provide cognitive support for the reverse engineers at the time of the analysis, some documentation is meant to capture the reverse engineers' own understanding of the code, and other documentation is meant to be shared either with team members or outside stakeholders. While it is already difficult to document source code written in high-level programming languages, it is even more difficult when dealing with assembly code. During the exploration of the assembly code, most reverse engineers document just enough information to be able to resume a task and do not document the paths that were explored without success.

3) Transferring Knowledge: Transferring knowledge is a challenge in reverse engineering. Documentation alone is often not enough to understand the work that has been completed by somebody else: *"[I would] look at a version with comments, but I'd still need to jump through to understand."*$_{P7}$ In the current setting, information is usually passed on verbally or via email and chat. These mechanisms do not scale beyond groups of about five reverse engineers. To solve some of these issues, the idea of a workflow would be useful: *"Right now it's being done like a craft, and we'd like to have some kind of assembly line"*$_{P4}$. However, workflows are not consistent for all cases, and most workflow support tools are too constraining. In addition, conventions around documentation and standards on how to share information could improve the reverse engineering process: *"Respecting conventions [would make it] easier to pass from one project to another."*$_{P2}$

4) Articulating Work: Articulating work consists of all the items needed to coordinate a particular task, including scheduling sub-tasks, recovering from errors, and assembling resources [7]. In reverse engineering, where tangible results are only produced when a path of exploration was successful, constantly re-doing work is a problem. Work was usually divided based on different pieces of hardware, different vulnerabilities, different functions, or different files. Relating information from the analysis of difference pieces of the problem was very difficult.

5) Reporting: When external stakeholders are involved, the final step in a project is reporting the results of the reverse engineering activities. In some cases, reporting includes a great deal of articulation work, especially when artifacts can be co-opted as reports: *"Instead of writing a report we shared a Word document."*$_{P6}$

B. Tools

Tools used by the participants in our study can be classified as disassemblers, Office and visualization tools, and communication and coordination tools.

1) Disassemblers: Most of the reverse engineering work is performed using IDA Pro[3]. The Interactive Disassembler Pro is a commercial product that performs automatic code analysis and offers interactive functionality to support the

[1] http://www.symantec.com/index.jsp

[2] For our initial questions, see http://tinyurl.com/WCREInterviewQuestions.

[3] http://www.hex-rays.com/idapro

understanding of disassembly. Reverse engineers typically start with an automatically generated disassembly listing, then rename and annotate sections in the listing until they understand the code. Debuggers are rarely used for malware in the early stages of analysis since portions of the code are often missing for execution or because of anti-debugging tricks used by the malware that need to be removed first. As one of our interviewees described it, the main analysis tool used by reverse engineers in the security context is *"brain power"*$_{P6}$.

2) Office and Visualization Tools: Most of the documentation is written using Microsoft Word, Excel, or OneNote. UML sequence diagrams are usually drawn to represent control flow understanding. However, the reverse engineers had *"trouble finding good tools that draw graphs and make it easy to navigate and export graphs"*$_{P1}$. Paper was also used, primarily for workflow support, small graphs, and articulation work.

3) Communication and Coordination Tools: For communication, only basic tools, such as e-mail and chat, were used. Our interviewees work in a co-located setting that allows face-to-face communication, but data sharing is complicated by the nature of the classified work. Interviewees coordinated work using tools such as wikis, bug trackers, and shared documents.

C. Artifacts

Artifacts created during the reverse engineering process in our setting consist of annotations, artifacts created for cognitive support, and reports.

1) Annotations: IDA Pro supports two notions of annotations: repeatable and non-repeatable. A repeatable annotation will appear attached to the current item as well as other items referencing it. Non-repeatable annotations only appear attached to the current item[4]. In addition, pre-comments and post-comments can be attached to lines and functions. All annotations also show up in the IDA Pro dependency graph.

The reverse engineers used annotations for several reasons: to keep track of variables, to rename functions, to document jumps, and to record where a particular piece of code was reading from or writing to. However, one of the challenges is that annotations are always incomplete: *"When you document stuff you tend to skip stuff that's obvious at the time."*$_{P6}$

2) Cognitive Support Artifacts: Depending on the use case, different documents are created by the reverse engineers to aid their cognition. These include: memory maps, Excel or Word tables showing register usage and boot processes, data flow diagrams, sequence diagrams, and scripts. A common scenario is when an engineer needs to keep track of different paths that are being explored in order to understand a particular piece of code. One of our interviewees used Microsoft OneNote to do that: *"I also used OneNote in other projects to keep track of paths that way. The last line in the OneNote document was the last path [that I had] explored."*$_{P6}$

3) Reports: Companies focused on malware, such as Symantec, frequently create reports that give an overview of how a particular piece of malware works. Such reports rarely include enough detail to understand the inner workings of the malicious program, mostly because security companies do not want to reveal their insights to malware writers. In contrast, reports produced in our study setting had more technical content, and often included assembly code for functions as well as detailed descriptions of all input and output parameters.

[4]http://www.hex-rays.com/idapro/idadoc/480.shtml

V. Discussion of Challenges

Table I summarizes our findings by showing the tools, artifacts, challenges, and needs for each of the processes we identified. Each work process described in the last section involved a different set of tools. These tools, in turn, were used to produce artifacts in distinct, non-interoperable formats. Therefore, moving from one process to another required a lot of manual work. For instance, reverse engineers typically used IDA Pro which provided them with a hyper-linked visual structure through which they could jump from one point of the assembly code to another. However, re-using this cognitive support across applications appeared to be a problem: *"IDA Pro views are not bad, but they are difficult to export... I would export them to something where we can play with level of details, merge bugs together, higher level view of things – have to do this manually right now."*$_{P1}$

By moving from the analysis to the documentation, engineers produce artifacts that would help them resume their own tasks, but also transfer their knowledge to other team members. For example, reverse engineers have tried using wiki-based systems for sharing mixed content (e.g., details on how particular hardware works, including pieces of code). However, wikis have shortcomings when navigating code and related artifacts: *"Wikis are very document like, not ideal for documenting code – some kind of graph tool would have been better."*$_{P1}$. Overall, even when knowledge sharing was encouraged, reverse engineers faced a lack of proper tools to pass information along to others: *"There's also stuff that we don't know how to document."*$_{P1}$. Navigation is particularly a challenge when dealing with different documents such as the cognitive support artifacts mentioned above. A map of all documents and their connections usually only exists in the reverse engineer's head.

To articulate their work and break problems into pieces, engineers often followed a divide-and-conquer strategy: *"We go after different pieces. The problem is how to share information then... different people have different processes."*$_{P2}$. This poses an interesting phenomenon: there is no general process in the work of security reverse engineers. The following factors would influence this phenomenon:

1) Task Complexity: Tasks, such as blocking malware and breaking into secure devices, often include unsolved problems, thus requiring the use of different approaches, tools, and skills.

2) Security: The security context further obstructs the reverse engineers' work. Classified information cannot be shared, and for classified tasks, the reverse engineers are only allowed to work on classified, often un-networked, equipment. Often, information can not be transported since it could belong to different projects, different classifications, or different

TABLE I
SUMMARY OF FINDINGS

Processes	Tools	Artifacts	Challenges	Needs
Analyzing	IDA Pro	annotations for code understanding	hard to export, complexity	different levels of detail, track interrupts and registers
Documenting	paper, Microsoft Office, graphing tools	memory maps, tables, scripts, data flow & sequence diagrams	done manually, addresses and offsets change	tagging of addresses, automatic updates for new versions
Transferring Knowledge	face-to-face communication, chat, e-mail, wiki	multiple files and resources	knowledge management, security, incomplete doc	standards for documentation, traceability across artifacts
Articulating Work	bug trackers, Microsoft Office, paper, wiki	shared documents	bringing pieces back together, re-doing work	workflow and task management tool, log of all actions
Reporting	Microsoft Office	technical reports	time pressure, lack of integration	integration with analysis

machines. Even for unclassified contexts, such as malware, the nature of the code prohibits sharing to prevent further infection. This also means that a lot of the work has to be completed offline and access to web resources is very limited. Most of the reverse engineers in our study worked by themselves, often for security reasons: *"I'm the only one allowed to look at it [...] You don't want others to be infected [with malware]"* $_{P2}$. The need to work individually also contributes to the increase in the effort needed to articulate work (e.g., summarizing work into a single report).

3) Time Constraints: The amount of time pressure depends on the scenario. Some projects have the goal of understanding everything about a particular piece of software and are usually completed without time pressure. In other scenarios, only a couple of weeks are allocated for a particular project in order to provide a fast response to a potentially harmful threat. In the latter case, the reverse engineers have to prioritize what they are working on. In the example of malware: *"[We have] four goals when dealing with malware: detect, block, remove, [and] understand everything. Usually [the process] stops after the third step."* $_{P7}$ The amount of documentation produced depends on the extent of the time pressure. Long-term projects without time pressure yield more documentation, whereas for short-term projects, there is often not enough time to document thoroughly: *"If you put too much documentation, you won't have enough time to finish."* $_{P2}$

4) Tool Constraints: A graph is often the best way to capture a certain aspect of a reverse engineering problem, but it is difficult to deal with different types of diagrams. One of our interviewees told us that he sometimes spends up to 100 hours creating a single diagram. Also, the graphs produced are usually not linked to the disassembly, thus losing traceability. There is a shortage of tools that span different aspects of reverse engineering such as hardware specifications and assembly code. The reverse engineering is also limited by memory since tools rarely scale beyond executables larger than a few megabytes.

While some of the analysis can be performed automatically, this is hindered by the fact that the disassembly produced by IDA Pro is not always perfect: *"Everything has to be perfect if you want to do automated analysis."* $_{P4}$ Creating plugins for IDA Pro is challenging when user interface functionality is required. The nature of assembly code yields additional requirements for tool support: addresses and offsets may change with every new version; malware code is often self-changing; and following one trace through the code means jumping to many different locations, both in the code and in external modules (e.g., DLLs). Annotations in the disassembly do not capture the order in which certain calls are being made, hence the use of sequence diagrams.

VI. IMPLICATIONS

In this section, we discuss tool and process implications based on our findings.

A. Process Implications

Software reverse engineers have to put a lot of effort into moving from one work process to another. For example, IDA Pro annotations are useful during the analysis, but using these annotations over time requires manually updating or tagging addresses to accommodate newer versions of malicious code. There is also a gap between annotations and high-level documentation such as memory maps and sequence diagrams. Transposing the barrier which separates each work process and dealing with a myriad of artifacts requires workflow support, both in processes and tools. However, since the nature of the tasks is already constrained, process or workflow tools should not add any additional constraints. In other words, process support for reverse engineers in a security context needs to be lightweight and flexible. First, it needs to support coordination by enabling the definition of sub-tasks while providing awareness on the progress of those tasks, and second, it should provide guidelines for high-level tasks that can be instantiated to meet the particular needs of a given context. To

bridge the gap between distinct work processes, some level of standardization across different types of documentation could make it easier to capture and share knowledge in consistent ways.

B. Tool Implications

To support software reverse engineering in a security context, a suite of tools will likely always be needed to deal with different devices and rapidly changing malware. However, there is a need for tooling to support traceability of artifacts created by different tools. Supporting more powerful navigation interfaces and the visualization of different levels of abstraction is essential. Different levels of detail should be supported in different parts of the tooling to distinguish between low-level documentation for cognitive support at the time of analysis and high-level documentation for reporting purposes, and to distinguish between different levels in the structure of a piece of code. One of our interviewees used OneNote for the latter: *"[I] used OneNote [...] and used tabbing to keep track of different levels in the structure."* $_{P6}$

In our study setting, no formal version control mechanisms were used but most interviewees made frequent backups. To deal with conflicts, one of our interviewees had implemented tool support for merging two IDA Pro files. Version control across different tools and file formats could help integrate these efforts. A log of everything that has been done to a particular piece of code would be helpful to make sure that work is not duplicated. As described in the previous section, workflow support is also needed. However, such tooling must be flexible enough to support different tasks with different requirements (e.g., different time constraints and individual vs. collaborative work practices). Newcomers and experienced software reverse engineers should be supported in their efforts to learn new tasks as software, malware, devices, and tools change rapidly. Documentation should not just communicate results, but also describe how these results were derived.

VII. Limitations

As with any chosen methodology, there are limitations with our research method. The first limitation lies in the small number of interviewees and the fact that all of them have gone through the same training process. However, gaining access to the unique setting of security reverse engineers is difficult due to time constraints and the security restrictions of their setting. As this is one of the first studies of reverse engineering in a security context, we believe that our findings provide initial insights on the impact of factors such as security limitations, time constraints, and insufficient tool support.

We were not allowed to use recording devices and some information was not available to us because of the security context of our study. For all of our interviewees, English is not their native language. The risk of not adequately capturing all answers by our participants was mitigated by the fact that two of the authors took notes independently and the notes were verified for accuracy by two of the participants. Both sets of notes were used in the analysis of the data.

VIII. Conclusion and Future Work

The work setting of reverse engineers tasked with security-related issues, such as the detection of malware or the decryption of encrypted file systems, is unique. Web resources are often unavailable because work has to be performed offline, files can rarely be shared to avoid infecting co-workers with malware or because information is classified, time pressure is immense, and tool support is limited.

To gain an understanding of the work done by security reverse engineers, and to inform industry and academia of their unique work practices, we conducted an exploratory study aimed at understanding their processes, tools, artifacts, challenges, and needs. We identified five processes: analyzing assembly code, documenting findings through different kinds of artifacts, transferring knowledge to other reverse engineers, articulating work, and reporting of findings to stakeholders. We found a lack of adequate tools to support their tasks that might also be prevalent in other reverse engineering settings. There is no general process that can capture all of the work done by security reverse engineers. Task complexity, security context, time pressure, and tool constraints make it impossible to follow a structured heavyweight process. Therefore, process and tool support has to be lightweight and flexible.

Reverse engineering in a security context is a fast-changing environment. New tools and approaches have to be learned on the spot as hackers and organized cyber groups create new security threats with implications for national security. Future work lies in addressing the challenges that we have identified with improved tools and processes, and in studying their usefulness in the unique work environment of security reverse engineers.

Acknowledgements

We wish to thank the participants in this study for conducting interviews with us, and we appreciate the feedback from Cassandra Petrachenko that helped improve this paper. This research is funded through NSERC grant DNDPJ 380607-09 and DRDC Valcartier.

References

[1] F. Cohen, "Computer viruses: Theory and experiments," *Computers & Security*, vol. 6, no. 1, pp. 22–35, 1987.

[2] T. F. Peterson, *A History of Hacks and Pranks at MIT*. The MIT Press, 2011.

[3] Symantec, "Internet security threat report (2010)," Available online: http://www.symantec.com/business/threatreport/topic.jsp?id=threatreport&aid=notable_statistics. Last access: 6/23/2011.

[4] K.-K. Choo, "Organised crime groups in cyberspace: a typology," *Trends in Organized Crime*, vol. 11, pp. 270–295, 2008.

[5] C. Cifuentes, T. Waddington, and M. Van Emmerik, "Computer security analysis through decompilation and high-level debugging," in *Proceedings of the 8th Working Conference on Reverse Engineering*, 2001, pp. 375–380.

[6] D. Quist and L. Liebrock, "Visualizing compiled executables for malware analysis," in *VizSec 2009: 6th International Workshop on Visualization for Cyber Security*. IEEE, 2009, pp. 27–32.

[7] E. M. Gerson and S. L. Star, "Analyzing due process in the workplace," *ACM Transactions on Information Systems*, vol. 4, pp. 257–270, 1986.