

Simulation Software Vulnerability Assessment for Coalition Interoperability

John A. Hamilton, Jr., Ph.D.
Department of Computer Science & Software Engineering
Auburn University, AL 36849
(334) 844-6360
hamilton@eng.auburn.edu

Colonel Kevin J. Greaney, US Army
Director, Models and Simulations
Missile Defense Agency
(703) 697-4360
Kevin.Greaney@mda.osd.mil

Gordon Evans
Booz Allen Hamilton
Missile Defense Agency
(703) 697-4582
gordon.evans-contractor@bmdo.osd.mil

ABSTRACT: *The need for simulation software vulnerability assessment is being driven by three major trends: Increased use of modeling and simulation for training and operational planning; increased emphasis on coalition warfare and interoperability and finally increased awareness of the potential security risks inherent in sharing operationally useful software. This paper will describe in an unclassified manner the process developed by the Missile Defense Agency and Auburn University to evaluate potential vulnerabilities in shared simulation software.*

The Missile Defense Agency’s Models and Simulation Directorate (MDA/SES) developed a vulnerability assessment process in partnership with Auburn University’s Information Assurance Laboratory. We describe the process and the environment that framed our process. This research has demonstrated a viable, scalable means of assessing the vulnerability of complex simulation software. We believe this methodology is appropriate for use with other simulation programs. It is always difficult to prove a negative. We do not claim that our process can prove the absence of any vulnerabilities or find every vulnerability in every software implementation. However, this process can provide an important means of risk mitigation. We believe the process defined here can successfully identify vulnerabilities in simulation software.

1. Simulations are software

This may seem an obvious point, but many members of the DOD modeling and simulation community are not software engineers. So the first point to be made is that The security model for many missile defense simulations is similar to that used in the heyday of Army Nuclear Weapons training. When virtually all U.S. Army medium and heavy artillery batteries were nuclear capable, it was

the model’s software implementation must be analyzed as well as the model itself.

The security model for many missile defense simulations is similar to that used in the heyday of Army Nuclear Weapons training. When virtually all U.S. Army medium and heavy artillery batteries were nuclear capable, it was necessary to conduct training for units, particularly Reserve Component units, who did not have secure facilities to handle classified models. The result was a



Figure 1. US Army Nuclear Weapons Training Model circa 1980 (unclassified)

training system that performed unclassified results when given notional inputs and only provided classified results when given actual weapons performance data. Soldiers were thus able to train on how to do the targeting calculations without handling classified information as shown in figure 1.

Applying this model to missile defense simulations is more difficult because the calculations are much more complex and there are so many more parameters to deal with. Further, these simulations are implemented in software that increases the complexity. Therefore, the Missile Defense Agency’s Models and Simulation Directorate (MDA/SES) developed a vulnerability assessment process in partnership with Auburn University’s Information Assurance Laboratory. We next describe the process and the environment that framed our process.

2. Assessing the Threat

U.S. missile defense programs, training, tactics and procedures are a matter of intense interest to foreign intelligence agencies. An unclassified analysis of one missile defense simulation software site showed that nearly a third of the hits on that site could be traced back to the People’s Republic of China [1]. The largest number of recorded hits (the mode) came from Beijing, and this was more than twice the number from the site with the second largest frequency. This obviously does not include undetected intrusions.

Missile defense is a topic of keen interest to many foreign intelligence agencies. Intelligence agencies do not face the same kind of economic constraints, as do practitioners of economic espionage. For this reason, military-relevant

software may be attacked in ways that would not be feasible for an industrial reverse engineering application.

3. Defining a Process

The first step is recognizing that secrets in binary code are not secret. Just because it is hard to extract information from a binary file does not mean it is impossible to do so [2]. As shown in figure 2, our process analyzes inputs, outputs and the software binaries.

One very sensitive aspect of missile defense is weapons and systems performance. When analyzing the system inputs we look at how well we can simulate a system based on open-source data. Next, we search for buffer overflows. Given the popularity of the C programming language it is usually not hard to find a buffer overflow – either in the application or in the operating system it is running on. Whether a buffer overflow can be used to compromise sensitive information in the application remains to be seen. Theoretically, one could jump to a code segment written to start dumping out intermediate calculations. Operating systems are written in C and are also vulnerable to buffer overflows.

Buffer overflows can be used for more than just jumping a program to an unauthorized code segment. We found that entering control characters into an entry screen would bring up a debugger providing important clues to how the program was originally compiled. Knowing what compiler was used to compile the binary is useful for the next phase. What if the simulation developers used explicit bounds checking for every input? One thing to look for is to find out if any sensitive information can be gleaned from the bounds. An interceptor that has actual minimum and maximum ranges as bounds would be an example of a possible vulnerability.

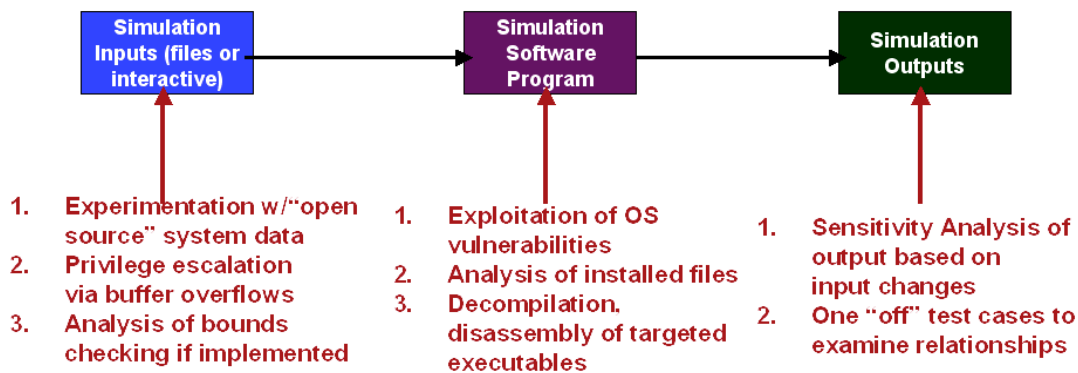


Figure 2. Process for Simulation Software Vulnerability Analysis

A good definition for reverse engineering may be found at: <http://www.program-transformation.org/twiki/bin/view/Transform/ReverseEngineering>. [3] Defines reverse engineering as “the process of analyzing a subject system with two goals in mind: (1) to identify the system's components and their interrelationships; and, (2) to create representations of the system in another form or at a higher level of abstraction.” In this process we look at both disassembly of code as well as decompilation.

Dissassembly is reconstructing assembly language code from a binary. Eric Imsand and Adam Sachitano have disassembled missile defense simulations using “dis” on Solaris and a shareware program called “Hackman” on Windows platforms. They report the following: The Hackman application ran easily. After launching the program, it was simply a matter of selecting the tool, either a hex editor or disassembler, and choosing the file to open. Hackman opened the file, and disassembled it without further user interaction. The entire disassembly process took approximately six hours running on a 400 MHz Intel Celeron processor with 128 MB of RAM [4]. Imsand and Sachitano produced about 1 gigabyte of assembly code and were later able to reassemble the binary and successfully run it.

With assembly code in hand it is possible to insert additional instructions to create a modified binary that dumps every variable value to an output device. It is also possible to search for string literals.

Decompilation is generally considered to be the generation of high-level source code from low-level input [5]. We have experienced little success in decompiling, primarily due to our reliance on freeware and shareware tools. It should be noted that commercial decompilers are available and that the state of the art in this area continues.

Weide, Heym and Hollingsworth discuss reverse engineering of large legacy software systems and conclude that the reverse engineering of such systems is ‘intractable’ in the sense that if one is given real [high-level] legacy code, the time required to show the validity of an explanation for why it exhibits a certain behavior is at least exponential compared to the size of the source code [6]. However, the same paper asserts a caveat, which is repeated here: “This does not mean that the task is impossible. It means that it is prohibitively costly for large systems [6].” We would add that what is prohibitively expensive in the commercial sector is not necessarily prohibitively expensive for a high priority intelligence effort.

It is not uncommon to provide the source code for the models that run on the simulation. A serious analyst will

want to know how a model is implemented in software before using it as a decision/study aid of any significance.

Finally, we look at the outputs. We attempt to determine the internals of the programs by analyzing the outputs. In general, we believe that missile defense simulations of any importance are too complicated to make this a useful strategy for reverse engineering the simulation. However, it is possible to gain some insights into specific aspects of the simulation by constantly running it and making minor changes to the input and tracking the changes. If the simulation is well documented – this strategy can be used in conjunction with an analysis of the documentation.

4. Refining and Applying the Process for Different Levels of Assurance

Given the costs associated with vulnerability analysis, we defined three rather arbitrary sets of tasks providing qualitatively better assurance. We developed three levels of assurance: High, Medium and Low. These categories reflect the level of effort required for the analysis. The requirements for each are enumerated below in Table 1:

High Assurance	Medium Assurance	Low Assurance
Line-by-Line verification of source code	Desk check of selected source code	Desk check of selected source code
Professional decompilation of executables	String search on disassembled code	String search on disassembled code
Complete review of published documentation	Targeted review of published documentation	Targeted review of published documentation
Open source review of weapons and systems data	Open source review of weapons and systems data	Analysis of degree of parameterization
Analysis of simulation runs to evaluate training, tactics and procedures	Analysis of simulation runs to evaluate training, tactics and procedures	
Analysis of degree of parameterization	Analysis of degree of parameterization	

Table 1. Assurance Levels for Simulation Software Vulnerability Analysis

It is important to recognize that anything sensitive in the source code is vulnerable. It is hard and time consuming and expensive to get at it – but it is naïve to think that a hostile intelligence agency would not make such an attempt. Next we address each item in Table 1.

4.1 Source Code A line-by-line verification of a simulation with a million+ lines of code is non-trivial. Worse, there is no guarantee that such a massive effort

will uncover all potential security issues. However, it is the best way to detect problems. Software engineering research has long held that the best way to find any problem in software is through desk checking. In most cases a line-by-line verification will not be warranted. If (and this is a big if) the simulation software is well structured, then it is reasonable to exclude large portions of the code and simply focus on the modules that deal with sensitive issues. It can be argued that a more focused review of “high-risk” code could potentially be more fruitful than plowing through a massive program in its entirety. Any source code provided, as part of the distribution must be reviewed. Source code analysis can give you a “worst case” vulnerability assessment.

4.2 Decompilation/Disassembly Decompilation and disassembly can be used to provide an “expected case” analysis. We pursue this to see what a potential adversary can learn from the binaries. For high assurance requirements, we recommend using professionals to decompile the binaries. Open market tools (available to a university anyway) are not yet to a point where useful results can be gained through reasonable efforts. Disassemblers are readily available and useful. It is reasonable to write scripts to do string searches on massive assembly code files and prudent to do that. In all cases, the binaries should be checked to make sure that all debugging information is stripped before the binaries are released.

4.3 Documentation Review Documentation of simulations must be included in the distribution [7]. Some simulations include more than 1,000 pages of documentation. Documentation is critical to the successful utilization of a simulation. As Sargent notes: “Documentation on model verification and validation is usually critical in convincing users of the ‘correctness’ of a model and its results, and should be included in the simulation model documentation [8].” The caveat to Sargent’s assertion is that the documentation must be reviewed to make sure that no sensitive information is inadvertently released. The physics of missile trajectories are not sensitive, probability of kill for a given system is very sensitive.

4.4 Open Source Review. There is a great deal of published information on missile and missile defense systems, particularly older systems such as SCUDs. One way to exercise a simulation is to create models from open source material and then experiment with it.

4.5 Analysis of Simulation Runs. Using open source inputs provides the means to develop simulation runs and analyze the outputs. The objective is to reduce the number of unknowns in the system. The more known information that can be input, the easier the analysis.

4.6 Analysis of Degree of Parameterization. Essentially we want to check and see how well the model lives up to the objective in Figure 1. If there are default values, then those values need to be checked to see if any of them are sensitive in nature. In general, the greater the degree of parameterization, the closer the simulation approximates the model in Figure 1.

5. Conclusions

In most cases we believe that a medium assurance assessment is sufficient. Before we share simulations (missile defense or others) with our coalition partners, it is essential to know what we are sharing. Simulations are increasingly being used for coalition training and operational planning. Simulations suffer from the same vulnerabilities as any other sensitive software.

There are ways to protect the secrecy of the code contained in the executable. Like all security issues though, the developers must weigh the cost and hassle of putting in devices to protect their code, against the ease of using and maintaining their software. We believe a client server model with strong encryption authentication will be used to transfer sensitive information and protect US-only information.

In the course of our evaluation of operational missile defense simulations, we have come away with several lessons learned:

- An executable file can be modified to run a background program without the user’s knowledge.
- Someone wishing to install a backdoor could disassemble the executable and update the assembly code to perform whatever task he/she wishes.
- If this is the desired effect, it is much easier just to add the background program to the source code and recompile, if the source code is available.
- Linux for the x86 platform offers useful tools to analyze Microsoft Windows programs.
- Remove as much debugging information as is possible/feasible from the executable code.
- Compilation is not encryption. A binary executable can be reverse engineered for information or altered to perform improperly.
- Don't put anything in the source code you wouldn't want extracted in some form.
- The string literals found in an executable file can often give information such as method names, compiler version, and output strings.
- Source code comments should not typically be included in the binary
- Certain tools can be developed to automate some of the customary checks in a vulnerability analysis.
- If information is sensitive force the user to input the parameters.

That being said there is a line between forcing a user to input information, and a system being actually usable. If a user must input all variables and equations, then the system is nothing more than Matlab with another GUI.

This research has demonstrated a viable, scalable means of assessing the vulnerability of complex simulation software. We believe this methodology is appropriate for use with other simulation programs. It is always difficult to prove a negative. We do not claim that our process can prove the absence of any vulnerabilities or find every vulnerability in every software implementation. However, this process can provide an important means of risk mitigation. We believe the process defined here can successfully identify vulnerabilities in simulation software.

6. References

- [1] [Mann, Steve, "Default Report Website Visitors," WebTrends, NetIQ Corporation, Internal Report, 17 May 2002, p 41.
- [2] Viega, John, McGraw, Gary, *Building Secure Software*, Addison-Wesley, Boston, Mass., 2002, p 109.
- [3] Van Deursen, Arie. 2003. Reverse Engineering. Website - <http://www.program-transformation.org/twiki/bin/view/Transform/ReverseEngineering>
- [4] Imsand, Eric, Sachitano, Adam, "Analyzing Security Vulnerabilities in National Missile Defense Simulation Software," unpublished, November 2002.
- [5] Breuer, Peter T., Bowen, Jonathan P. "Decompilation: The Enumeration of Types and Grammars," *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16 no. 5. 1994.
- [6] Weide, Bruce W., Heym, Wayne D., Hollingsworth, Joseph E., *Reverse Engineering of Legacy Code Exposed. Proceedings of the 17th International Conference on Software Engineering (Seattle, Washington)*. ACM, New York, NY, 1995, pp 327-331
- [7] Chatam, Wade, "A Vulnerability Analysis with an Emphasis on Using Documentation," unpublished, November 2002.
- [8] Sargent, R., "Verifying and Validating Simulation Models," *In Proceedings of the 28th Winter*

Simulation Conference. (Coronado, CA). ACM Press, New York, NY, 1995, pp 55-64.

Author Biographies

JOHN A. "DREW" HAMILTON, JR., PH.D., is an associate professor of computer science and software engineering at Auburn University and director of Auburn University's Information Assurance Laboratory. He has a B.A. in Journalism from Texas Tech University, an M.S. in Systems Management from the University of Southern California, an M.S. in Computer Science from Vanderbilt University and a Ph.D. in Computer Science from Texas A&M University. Prior to his retirement from the US Army, he served as the first Director of the Joint Forces Program Office and on the Staff and Faculty of the United States Military Academy as well as Chief of the Ada Joint Program Office. CRC Press publishes his book, *Distributed Simulation*, written with LTC D. A. Nash and Dr. U. W. Pooch.

COLONEL KEVIN J. GREANEY, U.S. ARMY, is the Director, Models and Simulations, Missile Defense Agency. Prior to his assignment, Colonel Greaney served as the Commander of the Communication-Electronics Command Software Engineering Center - Meade from September 1997 through September 2000.. He graduated from Northeastern University in 1974 with a B.A. and was selected as a Distinguished Military Graduated (DMG) prior to commissioning as a second lieutenant in the Army. He has also earned a M. S. degree from Shippensburg University, a M.A. from Webster University, and a Ph.D. in Software Engineering from the Naval Post Graduate School.

GORDON EVANS retired from the U.S. Army in 1992 as a Lieutenant Colonel. During his military service he served in multiple Field Artillery and Military Intelligence assignments. Overseas assignments have been in Germany, Korea and Viet Nam. Since his military retirement, Mr. Evans has worked as an on-site consultant to the Missile Defense Agency (MDA), first for Vanguard Research, Inc. and now Booz Allen Hamilton. His areas of concentrations during those years of service to MDA have been in Systems Engineering, Command and Control, Modeling and Simulations, International Programs, and Technology Transfers. He has been the lead MDA designer and investigator for its Modeling & Simulation Vulnerability Assessment program.