Performance Impact of Connectivity Restrictions and Increased Vulnerability Presence on Automated Attack Graph Generation

James Cullum, Cynthia Irvine and Tim Levin
Naval Postgraduate School, Monterey, CA, USA
jcullum@nps.edu
irvine@nps.edu

Abstract: The current generation of network vulnerability detection software uses databases of known vulnerabilities and scans target networks for these weaknesses. The results can be voluminous and difficult to assess. Thus, the success of this technology has created a need for software to aid in network vulnerability analysis. Although research has shown the effectiveness of automated attack graph generation tools in displaying potential attack paths in a network, research involving the performance of these tools has been limited. The performance impact of connectivity restrictions and the number of vulnerabilities present on a network for these tools is not well understood. Using empirical testing, we have collected quantitative data using CAULDRON, an attack graph generation tool developed at George Mason University, on a collection of simulated networks defined to modulate connectivity at certain points in our networks and represent the number of vulnerabilities present per node. By defining our model to include sets of nodes, which allow connectivity from all nodes to all vulnerable nodes in the set; the number of nodes present in each set, the number of connections between sets; and the number of vulnerabilities per node as our variables, we are able to observe the performance impact on CAULDRON of both connectivity restrictions and the increased presence of vulnerabilities in our networks. The effect of these variables on processing time and memory usage is presented and can be used as a metric to assess the scalability of this tool within various customer environments.

Keywords: Attack graph, network, exploits, vulnerability analysis, performance.

1. Introduction

Automated attack graph generation tools use a database of known vulnerabilities, and input from network vulnerability scanners, to construct *attack graphs* comprising the potential paths through the network from an attacker to a target. The attack graphs are used to better understand the large volume of data produced by network vulnerability detection software. Research involving the performance of automated attack graph generation software has been limited. For example, the performance impact of connectivity restrictions and the number of vulnerabilities present on a network for these tools has not been well understood. We present the results of performance tests of the Combinatorial Analysis Utilizing Logical Dependencies Residing on Networks (CAULDRON) tool (Jajodia 2003), an attack graph generator developed at George Mason University. The tests are based on simulated networks in which the several performance-dependent factors are varied, including: the degree of internal connectivity between subnets (i.e., subsets of network nodes, or *node sets*); the number of vulnerabilities present per node; and the number of nodes in each subnet. The effect of these variables on processing time and memory usage is summarized, which we hope will provide a useful metric to assess the scalability of CAULDRON and similar tools within various customer environments, as well as provide useful feedback to its developers. Our analysis shows that CAULDRON's processing time and memory usage were directly responsive to several of the independent variables. Processing and memory use increased polynomially as the number of nodes in the network increased, as did processing with respect to the number of vulnerabilities per node. Processing time scaled linearly as the number of fully-connected nodes that were reachable from the attacking node increased. Other factors scaled linearly, or had minimal effect. This indicates that CAULDRON has the potential to scale well in large enterprise IT environments.

2. Background and related work

2.1 Cauldron

CAULDRON is an ongoing project developed and supported by George Mason University that uses network vulnerability scan reports to automatically generate attack graphs depicting all known combinations of vulnerabilities that could be systematically exploited by an attacker to reach an attack goal (Jajodia 2003). The version of CAULDRON examined in this paper and used during experimentation is version 2.6.

While there are currently many network vulnerability scanners that can detect the presence of vulnerabilities that exist on individual systems, in many of these tools vulnerabilities are considered in isolation (Ritchey

2002). By automating the task of analysis and by considering the vulnerabilities in combination rather than in isolation, CAULDRON's approach is to depict attack graphs showing how an attacker could systematically compromise a network in order to reach his attack goal (Noel 2002)(Jajodia 2003). CAULDRON also contains additional functionality which allows it to correlate Intrusion Detection System (IDS) events with attack graphs. These correlated results can be used to better distinguish attacks in progress from false alarms, and also provide context to attacks in order to determine where a response should be directed (Noel 2004), although this feature was not analyzed in the current work.

2.2 Cauldron abstract model

This section describes the abstract model underlying production of the attack graph (see Figure 1).

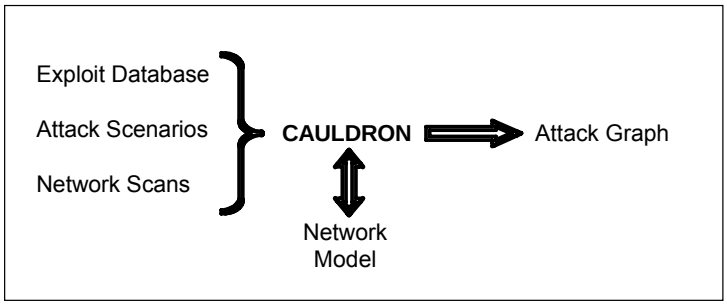

Figure 1: Cauldron engine inputs and output

2.2.1 Exploit Database

As depicted in Figure 1, a database of exploit models is input to CAULDRON. The exploit model is an abstract representation of the pre-conditions necessary for a single vulnerability to be exploited and the post-conditions that result after the exploit has been run. An exploit's pre-conditions define the required vulnerability, connection to that vulnerability, and user privileges that must be present prior to execution of an exploit (Ritchey 2002). Post-conditions reflect the state of the network after the exploit, in which the attacker may be able to obtain more information about the network, and/or obtain elevated user rights (Ritchey 2002). Thus, it is possible to represent how an attacker could chain the post-conditions of one exploit model to the pre-conditions of another exploit model until an attack goal is met. An initial database of exploit models is provided with the CAULDRON tool, which is updated regularly by researchers at GMU, and can also be customized by the end user to include additional knowledge.

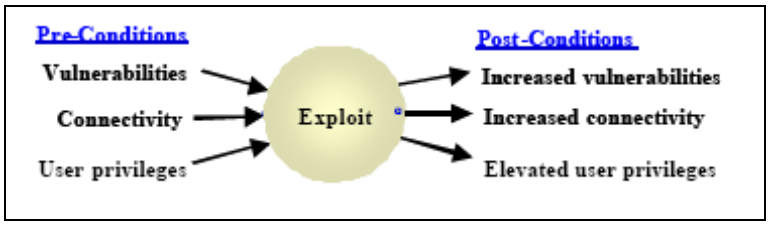

Figure 2: Exploit Model (Ritchey 2002)

2.2.2 Network scans

Another input to CAULDRON is network vulnerability scan data of the target network. If planned correctly, these scans can also show connectivity rules being enforced by filtering routers or firewalls (Ritchey 2002). By conducting a vulnerability scan through the firewall as well as behind the firewall, as depicted in Figure 3 and Figure 4, it is possible to model the attacker's knowledge of the network prior to the initial attack, as well as any knowledge gained after hosts residing behind the firewall have been compromised.

Vulnerability reports, which must be in XML format, are preprocessed by CAULDRON using CAULDRON's GUI to create a *network model* containing vulnerability and connectivity information.of nodes reporting vulnerabilities. Network hardening and other changes to the network model can be made by making modifications to the model's XML code as directed by the CAULDRON customization manual.

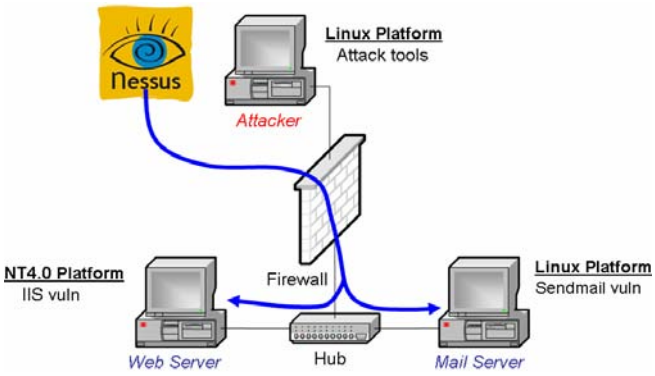

Figure 3: Attacker's initial knowledge of the network

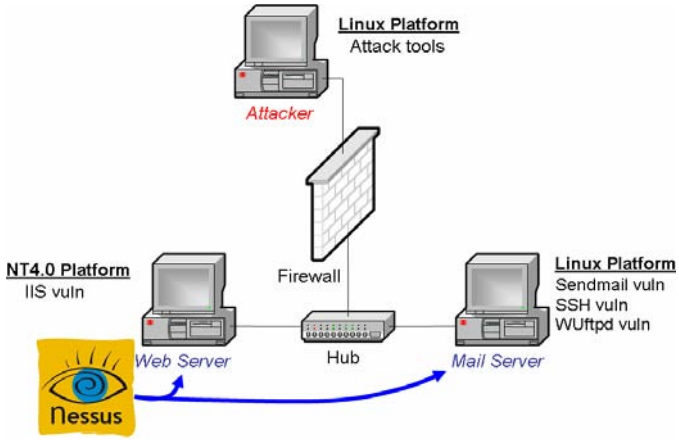

Figure 4: Attacker's knowledge after a host has been compromised

2.2.3 Attack scenario

The attack scenario is provided by the CAULDRON "user" and identifies the attacker and target nodes as well as the initial capabilities of the attacker and desired capabilities of the target. Capabilities of both the attacker and the target are represented in terms of "access level" and user privilege on a given node. The *access level* represents the ability to transfer files and execute code. *User privilege* represents whether or not the user has super user, or root level, access to his machine. The attack scenario is defined manually within the CAULDRON graphical user interface (GUI) using a predefined list of user privileges and access levels to and from nodes included in the network model, and can be edited using the GUI or by making changes directly to the XML code by following instructions provided in the customization manual.

2.2.4 Attack graph

Given the inputs described, CAULDRON generates and displays an attack graph. An attack graph depicts all paths between the attacker and the target (per the attach scenario) that are known to the exploit model. The vertices of the attack graph represent security conditions and exploits and are connected by edges, which represent dependencies (Jajodia 2003). CAULDRON can automatically aggregate certain attack information to produce a simplified attack graph, providing the user with a less complex visual representation of the attack graph. Figure 5 depicts an aggregated attack graph.

2.3 Previous Cauldron Performance Measurements

In research that led to the development of CAULDRON, an initial scalability experiment was conducted by GMU to test the scalability of combinatorial analysis using a model checker as the underlying analysis engine and a fully connected network with an increasing number of nodes on the network as a scaling parameter (Noel 2002). Although combinatorial analysis is still the underlying analysis approach that is used, the analysis engine is now CAULDRON rather than a model checker. Results from the original scalability experiment showed performance scaled in polynomial time, but that results could not be computed for networks containing more than 50 nodes using 512 MB of physical memory.

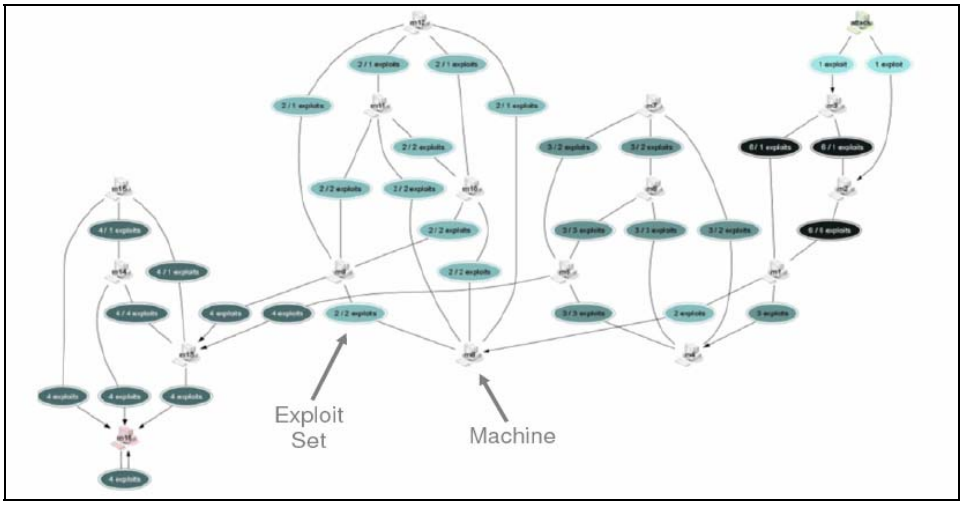

Figure 5: Aggregated attack graph (Jajodia 2004)

2.4 Other automated attack graph generation tools

In addition to ongoing work being done on CAULDRON, there are several other independent efforts underway to develop tools that can automatically generate attack graphs.

2.4.1 CMU - Attack graph toolkit

The *attack graph toolkit* from Carnegie Mellon University uses a *model checker* to generate attack graphs that match an attacker's capabilities with respect to the current state of the network. This system is similar to CAULDRON in that it accepts Nessus vulnerability reports as input in order to gain information about the network's topology and network vulnerabilities, and it also requires the manual creation of exploit models based on vulnerability gathering services. It differs in that it can interface with host based programs such as MITRE Corporation's Outpost and Lockheed Martin's ANGI systems to display vulnerabilities which might not be externally visible from the network, and can (attempt to) identify all possible attack paths to the target node rather than paths from one particular attacker (Sheyner 2004).

2.4.2 Lincoln laboratory – NetSPA

Network Security Planning Architecture (NetSPA) from Lincoln Laboratory generates attack graphs that show all possible paths from a particular attacker to all reachable nodes; in contrast, CAULDRON performs this step and performs further analysis to eliminate all reachable nodes that cannot reach the goal node. NetSPA uses the same general types of inputs as CAULDRON. The developers plan to incorporate the connectivity rules generated from imported firewall and router rules, but this feature has not yet been implemented. NetSPA also avoids the manual entry of an exploit database, such as is required by CAULDRON, by gathering this information from vulnerability descriptions used by Nessus, the Mitre CVE dictionary, and the NIST ICAT database. Lincoln Laboratory, reports favorable performance results (Ingols 2005).

2.4.3 Skybox - Skybox view

Skybox View is a commercially available tool developed by Skybox Security that can automatically generate attack graphs through the use of host-based agents, management interfaces, and an analysis server located on the target network (Skybox 2006). This product is similar to CAULDRON in that it requires a database of exploits and probing prior to analysis to discover which vulnerabilities are present on each host. It differs from CAULDRON because it analyzes live networks. While CAULDRON relies on data collected by network vulnerability scanners, this has the potential security advantage of being air-gapped from the target network. Although no performance details were available from Skybox, an examination of recent patents submitted by Skybox identified the algorithmic complexity of the product as n^4, where n represents the number of nodes present on the network (Lippmann 2005).

3. Experiment

Our hypothesis was that CAULDRON is efficient enough that it could be the basis for a product suitable for use in large-scale industrial (e.g., enterprise) IT environments. In this section, we describe the experiment to measure how changes to the inputs, such as degree of network connectivity and the number of vulnerabilities present per node, effect the time required to produce attack graphs, and the resources required to do so. If these effects are found to be reasonable, e.g., its processing is not on the order of n^p, then we can conclude that CAULDRON is at least a candidate for use in large IT environments.

3.1 Performance measurements

Two primary effects, or *dependent variables*, that could potentially limit the use of CAULDRON are its processing time and memory usage during the analysis of the input data. Processing measurements were recorded for the following phases of CAULDRON's execution:

- Total runtime – reading input files, performing attack graph analysis, and writing the results to the attack graph output file.
- Analysis runtime – running CAULDRON's attack graph analysis algorithm.
- Initialization runtime – reading input files, and determining which exploits were applicable to the network model.
- Forward Chaining Algorithm – running the first half of CAULDRON's attack graph analysis algorithm.
- Backward Chaining Algorithm – running the second half of CAULDRON's attack graph analysis algorithm.

We measured the maximum amount of physical memory in use by CAULDRON per test by measuring the maximum size of CAULDRON's *working set*. The working set is the smallest set of memory referenced by a process for efficient execution (Denning 1968). The test system was fitted with enough memory (2GB) so that performance issues related to exhaustion of memory were not noticed.

We also measured the maximum amount of virtual memory used by CAULDRON per each test.

3.2 High-level experiment description

The experiment was divided into five separate phases, with each phase testing the effects of different characteristics of the inputs, which comprise the experiment's *independent variables*: heap size, number of network nodes, degree of connectivity, vulnerabilities per node, and the size of the exploit database. Additionally, since minor instrumentation of CAULDRON was required, tests were performed to ensure that the modifications did not affect the test results.

3.2.1 Heap size

Since CAULDRON is implemented in Java, all live memory objects are stored in Java's memory *heap*. The Java Virtual Machine (JVM) *initial* and *maximum heap size* are established in CAULDRON's configuration file, and remain constant during runtime. Setting the initial and maximum heap size to the same value is said to increase a process's predictability by freeing heap-size management from the chores of the virtual machine (Sun 2003). In this phase the initial and maximum heap sizes were set equal to each other and gradually incremented from 128MB to 1GB during subsequent tests, leaving other variables constant. Before proceeding to the next phase of the experiment, the results were analyzed, and the heap size that exhibited the most stability was selected for use for the remainder of the experiment.

3.2.2 Number of network nodes

The second phase of the experiment measured how performance of CAULDRON changed when the number of nodes varied, and other variables were constant. Full network connectivity between nodes was configured. Each node contained in the network had a single vulnerability and a corresponding exploit associated with it. The tests included networks that ranged from 2 to 100 nodes and each test was repeated five times in order to verify that results were consistent.

3.2.3 Degree of connectivity

The third phase of the experiment examined how performance changed when the connectivity between nodes varied. In order to modulate connectivity, three variables were utilized: the number of fully connected *node-sets* (i.e., subnets) in the network; the number of nodes per fully connected node-set; and the number

of connections between each pair of fully connected node-sets. Selection of specific connections was randomized.

We define a *fully connected node-set* to be a collection of nodes in which every node has at least one vulnerable service, every node is connected to every other node, and access to vulnerable services is allowed. Multiple fully connected node-sets modelled networks in which connectivity between subnets is restricted, e.g., due to the presence of firewalls or filtering routers. A connection between fully connected node-sets allows a node belonging to one node-set to exploit a vulnerable service on a node belonging to another node-set, if it has access to any node in that node-set. This provided a simple way to define different degrees of connectivity. Although it was possible to model the size of each node-set individually, which would provide a more realistic network model, the effort to analyze the combinations of different sized node-sets was not considered worth the potential information to be gained, so a uniform size was used.

Multiple tests were conducted for each number of connections between node-sets, so that different connection pairs were used each time, lowering the impact the characteristics of specific connection pairs would have on the performance measurements.

3.2.4 Vulnerabilities per node

The fourth phase of the experiment measured how the number of vulnerabilities per node affected performance of CAULDRON. As with the size of node-sets for a given test (above), the number of vulnerabilities per node was uniform for all nodes for a given test. Each test was repeated five times in order to verify the accuracy of results due to expected performance variability.

3.2.5 Size of the exploit database

The final phase of the experiment measured how performance was affected by the size of the exploit database. Exploits were incrementally added that would not appear on the attack graph, but nevertheless would cause the database search space to increase.

3.2.6 CAULDRON baseline

To measure the impact that modifications to CAULDRON had on performance, additional experimentation was conducted using the modified version of CAULDRON and the unmodified version of CAULDRON on an identical network, attack scenario, and exploit database.

3.3 Implementation

The experiment was conducted on a standalone Dell Dimension 5150 3.2 GHz personal computer containing 2GB of memory and running the Microsoft Windows XP SP2 operating system. The experiment utilized simulated network data, rather than data from a real-world production network, or from a laboratory network. By simulating the experimental networks, it was possible to avoid issues that might arise such as revealing a real-world network's vulnerabilities to the public, or spending a lot of time to configure hardware and software, and validate that a laboratory network was setup correctly. Similarly, simulated exploit models were used. An experimental network was created to scale each variable contained in Section 4 independent of one another. By combining the results of each experiment, it is expected that the reader can infer the results of a multiple variable scenario.

Various software tools were required. Cygwin (Cygwin 2006), a Linux-like environment for Windows was used to provide access to string processing tools such as *awk*, *grep*, and *sed*, along and the *bash* script environment. Bash scripts were created to generate the input files, to automate each phase of the experiment, and to collect results. The Windows Performance Monitor was used to measure the peak working set, and the amount of virtual memory allocated exclusively to CAULDRON. To automate the experiment and collect additional performance data, modifications to the CAULDRON source code were made such that the program would accept command line input, terminate after analyzing the attack graph, collect and report the amount of time required to complete the different phases, and record the process ID of CAULDRON.

4. Results and analysis

The nominal test parameters, when not varied for a particular test, were as follows:

- An exploit database containing 1500 entries

- A fully connected network model that consisted of 60 nodes (one node-set containing 60 nodes)
- Each node possessed a single vulnerability that could be exploited by a unique exploit model contained in the exploit database
- The attack scenario was created such that the first node was the attacker and the last node was the victim.
- Each test is repeated five times to average out background noise.

4.1 Heap size

The heap size was changed in 128MB increments from 128MB to 1024MB. A heap size of 768MB was selected for the remainder of the tests since its resource usage was generally most stable: its variability was lowest in memory measurements and was low for time measurements as well.

4.2 Network size

The experiment was conducted using network models constructed with a variable number of nodes ranging from 2-100 nodes.

4.2.1 Observations

The number of nodes present on a network appears to play a significant role in the performance of CAULDRON, and the forward pass of the analysis algorithm appears to be the most significant factor in processing time.

Curve fitting analysis of the plots of the analysis time, total time, and forward pass time measurements showed that each measurement scaled with the number of nodes in approximately the same way. The amount of time to complete the forward pass algorithm is the dominant factor in analysis of the network model. A fourth order polynomial was the closest fit to the data, but the 3^{rd} and 4^{th} order terms are quite small – and the data is close quadratic. Figures A-1, A-2, and A-3 show how the analysis, total, and forward pass time measurements scaled as well as the fit equations that were used.

Curve fitting the backward pass algorithm data revealed that a third order polynomial provided a reasonable fit for the data obtained, but again the higher order terms are very small (see Figure A-4). The actual time required to run the backward pass algorithm was negligible in comparison to the forward pass algorithm. As shown in Figure A-5, the amount of time required to complete the initialization phase appears to scale linearly as the number of nodes are increased. Using the fit equation, which was calculated after removing the one outlier, it was observed that it would require at least 250 nodes to raise the initialization time by one second.

Analysis of the data collected for private memory storage indicates that the number of private bytes used does not scale as nodes are increased. Although a curve fit of the data suggested that private bytes scaled linearly, the slope of the line was shallow enough to cast doubt on whether virtual memory requirements scaled at all. Either way, it appears that the impact of adding nodes to the network is negligible in terms of private memory storage allocated to the CAULDRON process. As the number of nodes increase on the network, a larger working set was requested by CAULDRON (see Figure A-6). The size of the working set requested by CAULDRON appears to scale as a third order polynomial.

4.3 Connectivity

This experiment was designed to simulate the presence of firewalls, filtering routers, or any other network device that imposes connectivity restrictions on a real-world network. This experiment allowed the creation of hierarchical networks, rather than flat, or fully-connected networks, which were used in other experiments. In this test, the number of node-sets and the number of nodes per node-sets were varied. Connections between node-sets were generated at random, and multiple networks were tested in an attempt to identify any individual networks that impacted performance.

4.3.1 Observations

Analysis time is influenced by the number of nodes that are reachable from the attacking node, and the number of connections that are present in the network model. Although connectivity did not affect the amount of memory required, it affects analysis time, mostly during the forward pass algorithm.

The data shows that the size of the node-set, the maximum distance between nodes, and the number of connections have the largest effect on *overall performance*. The size of the node-set has the largest impact on the amount of time required to perform analysis.

Analysis runtime depends on the number of nodes that are reachable from the attacking node, and the number of connections present in the network model. Stratifications in the amount of time required to perform analysis were observed depending primarily on the number of node-sets that were reachable from the node-set containing the attacker. The amount of time required to perform the backward pass algorithm was small for all tests that were run. The time required to perform initialization, the amount of private bytes used, and size of the working set requested by CAULDRON did not appear to be affected by changes to connectivity allowed on the network.

4.4 Vulnerabilities per node

This experiment varied the number of vulnerabilities per node.

4.4.1 Observations

Curve fitting analysis of the data showed that the analysis time, total time, and forward pass time measurements scaled similar to one another just as in previous phases of experimentation. Each measurement increased as a second order polynomial, with the forward pass algorithm requiring the most time during analysis (see Figures A-9, A-10, and A-11). The impact of multiple vulnerabilities per node on memory requirements was small.

The amount of time required to perform initialization increased linearly with a very shallow slope, but was insignificant in relation to that of the forward pass algorithm. Examination revealed that the number of vulnerabilities per node did not affect the total amount of private memory used. The total size of pages belonging to the working set scaled linearly as more vulnerabilities were present per node. Although the size of the working set memory grew to approximately 50MB when all nodes present on the network gained an additional vulnerability, the amount of memory was small compared to other phases of execution.

4.5 Size of exploit database

This experiment varied the number of exploit models present in the exploit database. The smallest exploit database contained 1500 entries and the largest contained 31,500.

4.5.1 Observations

Curve fitting analysis of the data showed that the analysis time, total time, and forward pass time measurements increased linearly as more exploit models were added to the exploit database; however, the slope was very shallow, and minor (see Figure A-12), and does not have a significant effect on the amount of memory needed by CAULDRON to complete these tasks.

Initialization increases linearly as more exploit models are added to the exploit database. The slope appears to be approximately the same as for the forward pass algorithm, analysis and total time measurements (see Figure A-13). The time required to perform the backward pass algorithm, and the size of private bytes did not appear to scale as exploit models were added to the exploit database (see Figure A-14). The size of the working set increased linearly as exploit models are added to the exploit database, which was most accurate as the exploit database grew larger.

4.6 Modifications to CAULDRON

This test was conducted with inputs that would exercise memory and processor usage: a fully connected network model containing 60 nodes, and an exploit database containing 1500 exploit models.

A *Student's t-Test* was performed on analysis time, total time, private bytes, and working set measurements of the two versions of CAULDRON. For analysis time, total time and working set, the student's t-Test rejected the null hypothesis with greater than 99% confidence that there was no difference between the means for each data set. Although the Student's t-Test showed that working set data was influenced by the version of CAULDRON used, it appears that the graph visualization activities present in the unmodified CAULDRON, but not activated in the script-driven test version, account for the difference. In any case, the

data showed that the changes made to CAULDRON resulted in insignificant changes to the time measurements and size of the working set (less than one percent).

5. Conclusion

This performance analysis of CAULDRON provides practitioners with an understanding of CAULDRON's capabilities and limitations and offers developers insight into areas which may need further development. Our analysis shows that CAULDRON's processing time and memory usage were directly responsive to several of the independent variables. Processing and memory use increased polynomially as the number of nodes in the network increased, as did processing with respect to the number of vulnerabilities per node. The number of nodes that were reachable from the attacking node had a pronounced effect on the amount of time required to analyze a network. Other factors scaled linearly, or had minimal effect. Considering that simulated networks contained in our experiments were comprised entirely of vulnerable nodes and assuming that real-world networks would likely not be as saturated with vulnerabilities, we believe that CAULDRON could be used on much larger networks than those used during our experiment. In addition, it was observed that hierarchical networks could be analyzed in far less time than flat networks, suggesting that large hierarchical networks can be processed in a reasonable amount of time. This indicates that CAULDRON has the potential to scale well in large enterprise IT environments.

5.1 Future work

Analysis of performance measurements revealed input characteristics with the most impact on performance; however, additional research questions were also raised:

- In this experiment, all nodes in the network model possessed a unique vulnerability-exploit pair. It is uncertain how performance would have scaled if all nodes possessed the same vulnerability-exploit pair.
- An analysis of the CAULDRON algorithms could confirm and shed light on the results of this experiment.
- CAULDRON is capable of creating exploit models which consist of complex pre-conditions; however, it is unknown what effect the inclusion of more complex exploit models will have on performance.
- Since the experiment indicates that the forward pass algorithm takes the most time, efforts to improve performance should focus on the efficiency of this algorithm.

References

Cygwin (2006) Cygwin Information and Installation, http://www.cygwin.com/.

Denning, P. (1968) The Working Set Model for Program Behavior. Comm. ACM 11, 5, pp 323-333.

Ingols, K., Lippman, R., Scott, C., Piwowarski, K., Kratkiewicz, K., Artz, M., Cunningham, R. (2005) "Evaluating and Strengthening Enterprise Network Security Using Attack Graphs," Lexington, Massachusetts, October.

Jajodia, S., Noel, S., O'Berry, B. (2003) "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Kluwer Academic Publisher.

Jajodia, S., Noel, S. (2004) "Managing Attack Graph Complexity Through Visual Hierarchical Aggregation," in *Proceedings of the ACM CCS Workshop on Visualization and Data Mining for Computer Security*, Fairfax, Virginia, October.

Lippmann R., Ingols, K. (2005) "An Annotated Review of Past Papers on Attack Graphs," Lexington, Massachusetts, March.

Noel, S., O'Berry, B., Hutchinson, C., Jajodia, S., Keuthan, L., Nguyen, A. (2002) "Combinatorial Analysis of Network Security," in *Proceedings of the 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, Orlando, Florida, April.

Noel, S., Robertson, E., Jajodia, S. (2004) "Correlating Intrusion Events and Building Attack Scenarios through Attack Graph Distances," in *Proceedings of the 20th Annual Computer Security Applications Conference*, Tucson, Arizona, December.

Ritchey, R., O'Berry, B., Noel, S. (2002) "Representing TCP/IP Connectivity for Topological Analysis of Network Security," in *Proceedings of 18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December.

Sheyner, O., Wing, J. (2004) "Tools for Generating and Analyzing Attack Graphs," in *Proceedings of Formal Methods for Components and Objects*, pp 344-371.

Skybox (2006) Skybox View Suite Brochure, [online], http://www.skyboxsecurity.com/data_sheets/Skybox%20View%20Suite%20Brochure%20July%202006.pdf.

Sun (2006) Tuning Garbage Collection with the 5.0 Java[tm] Virtual Machine, [online], http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html.

Appendix: Performance graphs

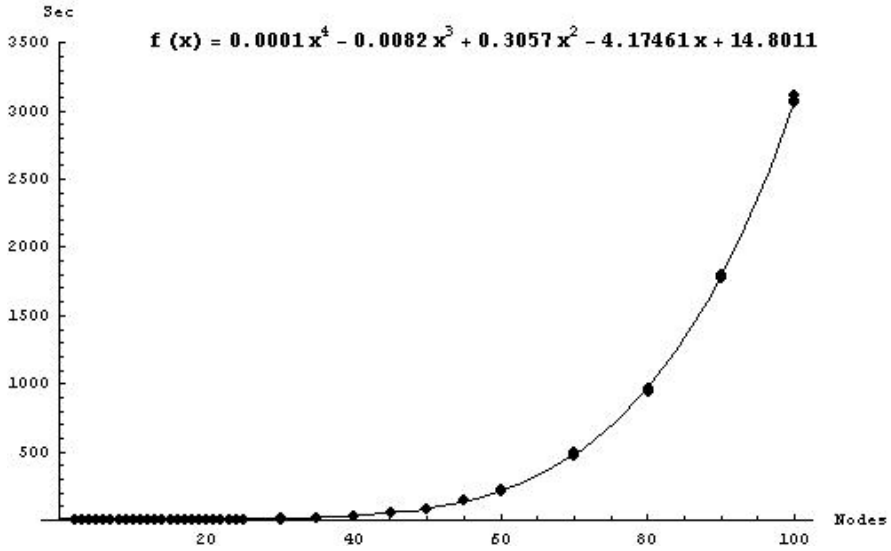

Figure A-1: Time required to complete the Forward Pass Algorithm

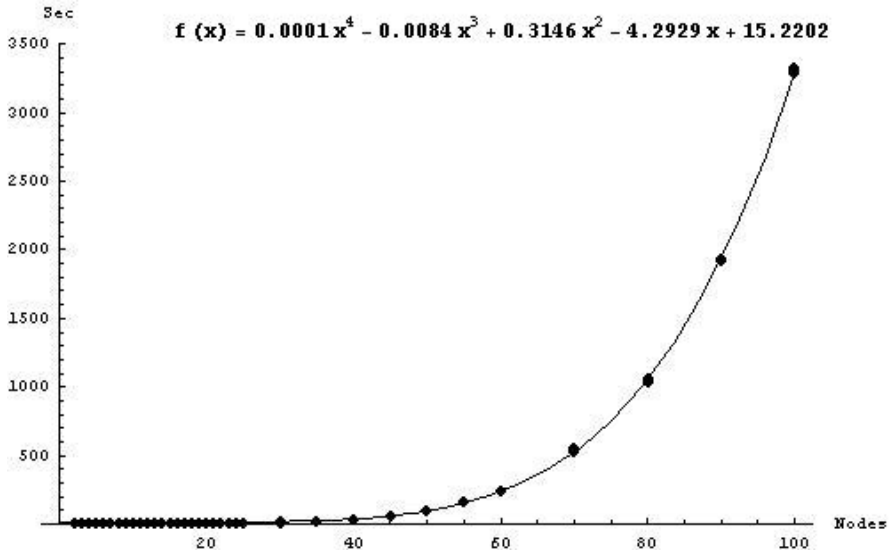

Figure A-2: Time required to complete the Analysis Algorithm

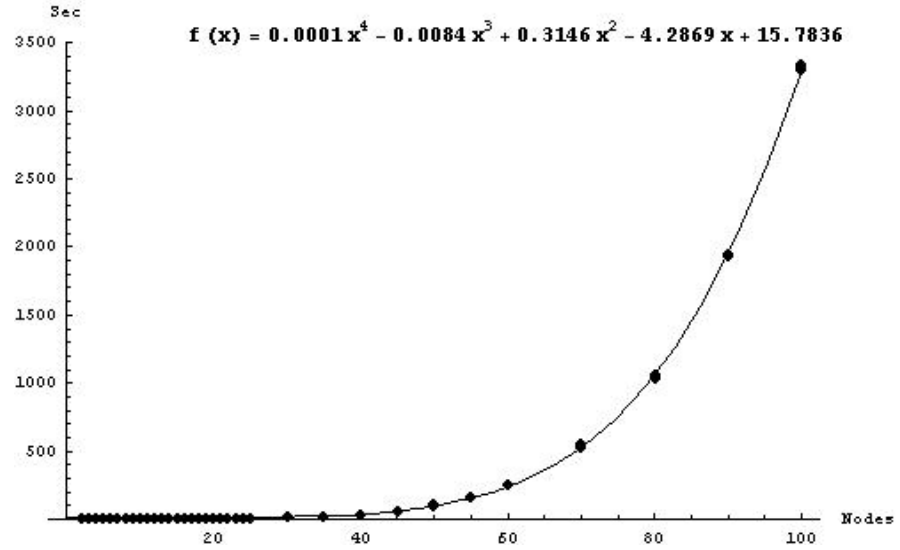

Figure A-3: Total Time required to Analyze

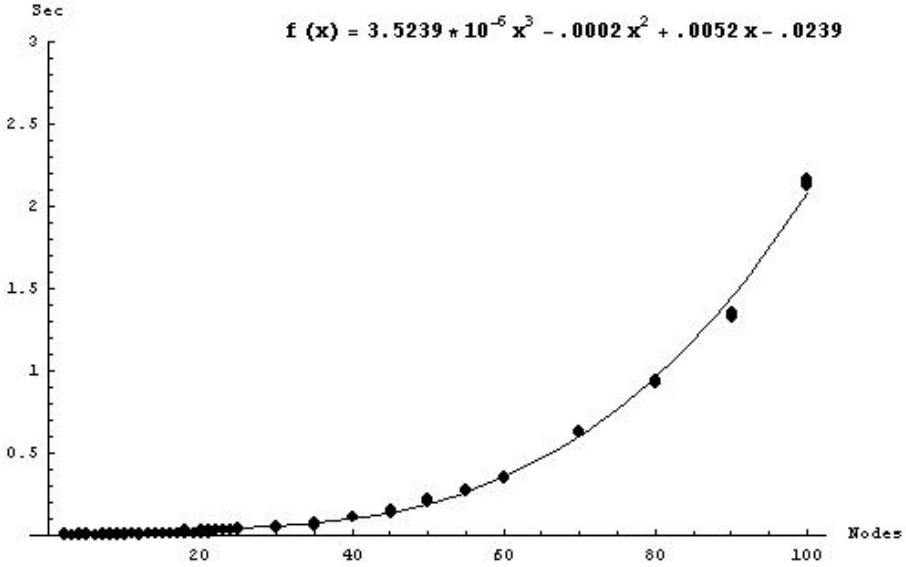

Figure A-4: Time required to complete the Backward Pass Algorithm

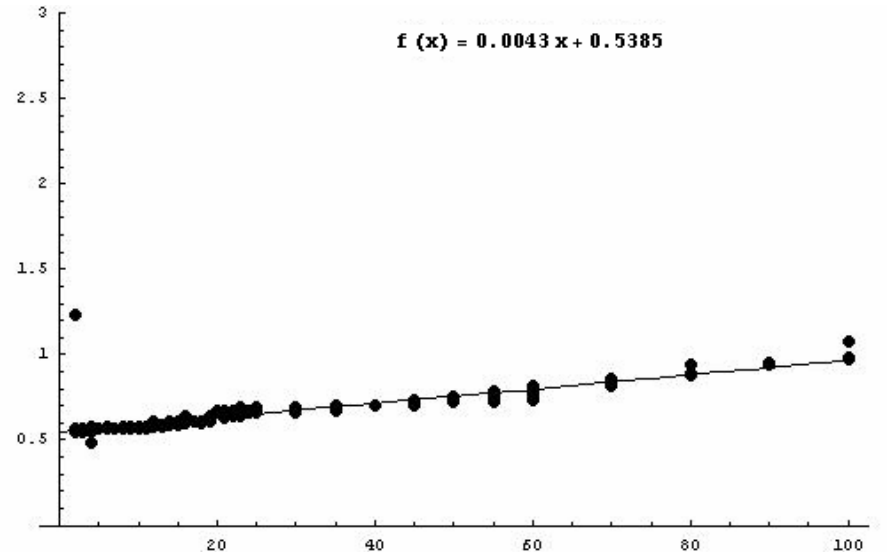

Figure A-5: Time required to complete the Initialization Algorithm

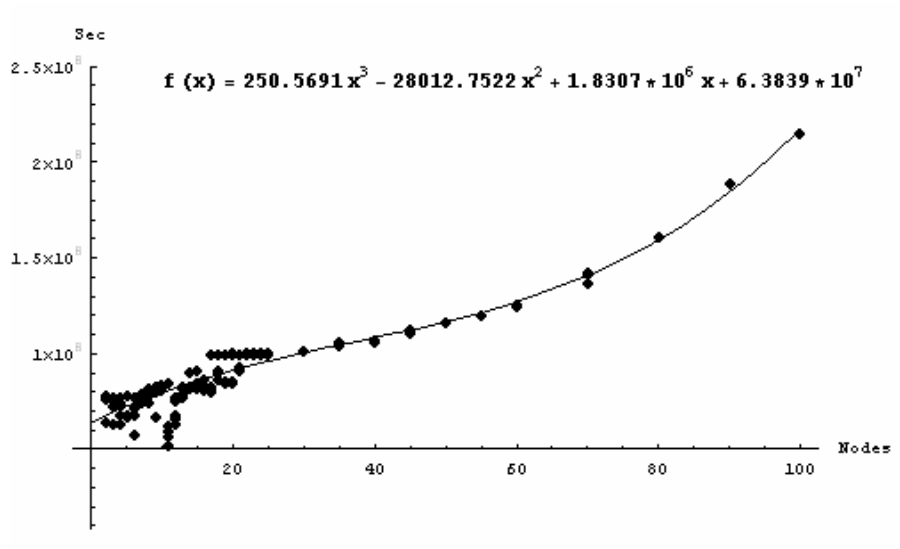

Figure A-6: Working Set Size as Network Size Grows

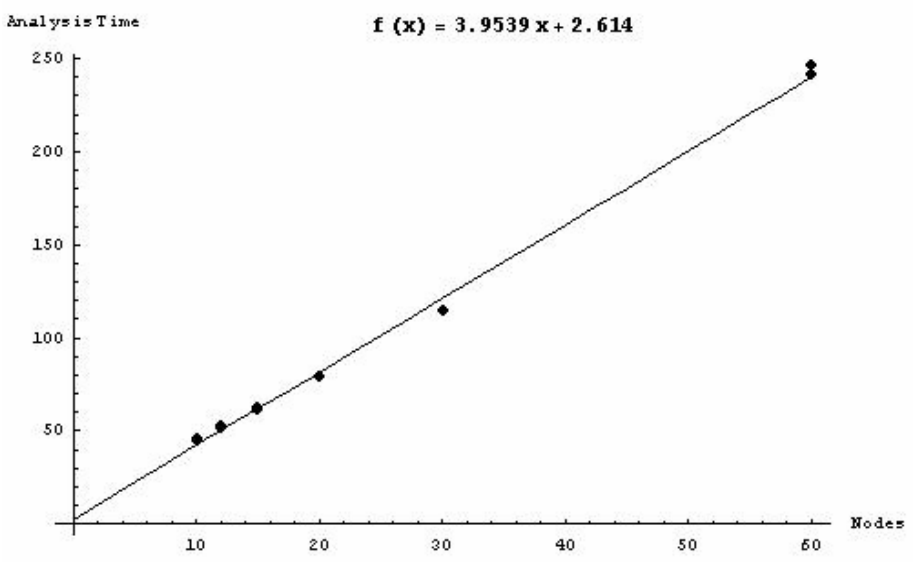

Figure A-7: Time required to perform analysis as node-set size increases

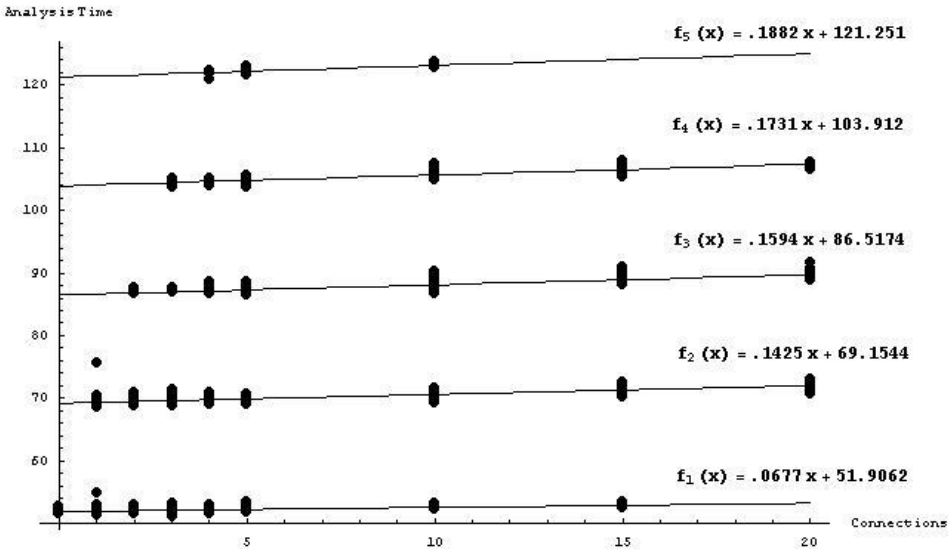

Figure A-8: Stratifications in time required to perform analysis

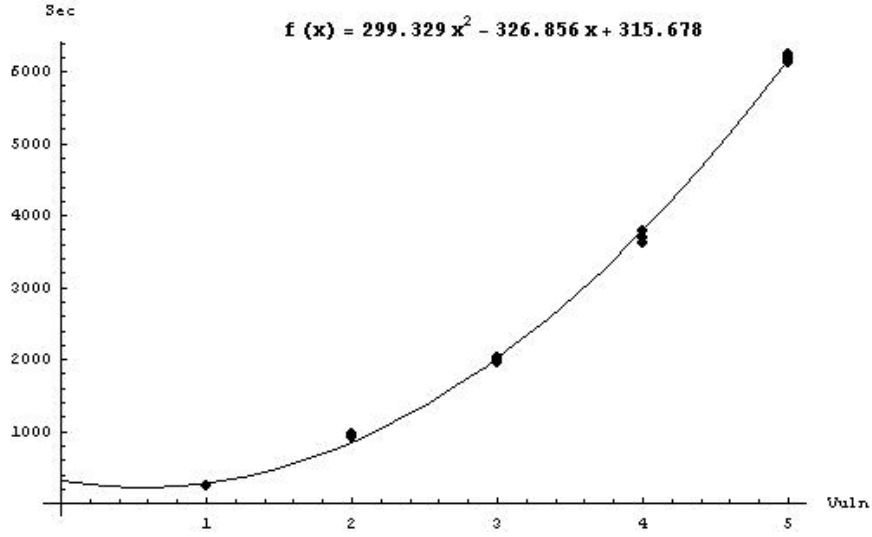

Figure A-9: Analysis time required as the number of vulnerabilities per node scaled

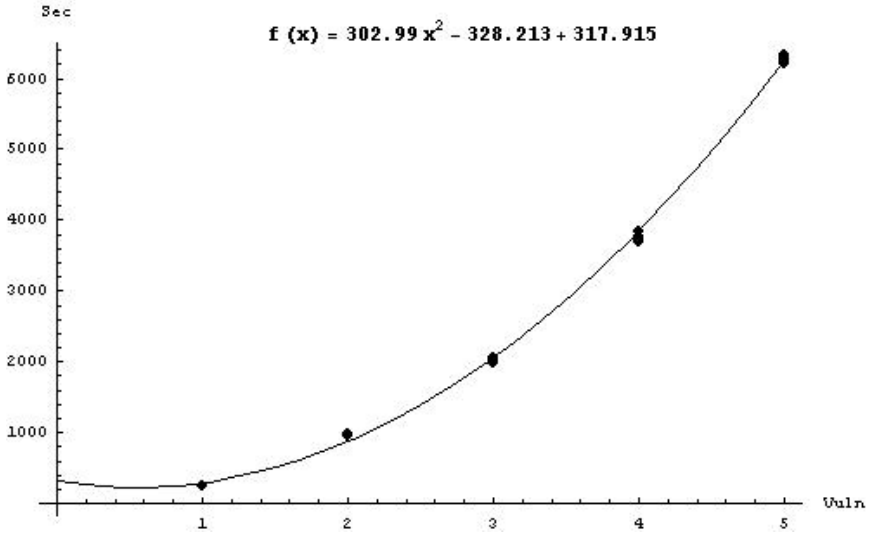

Figure A-10: Total time to complete tests as the number of vulnerabilities per node scaled

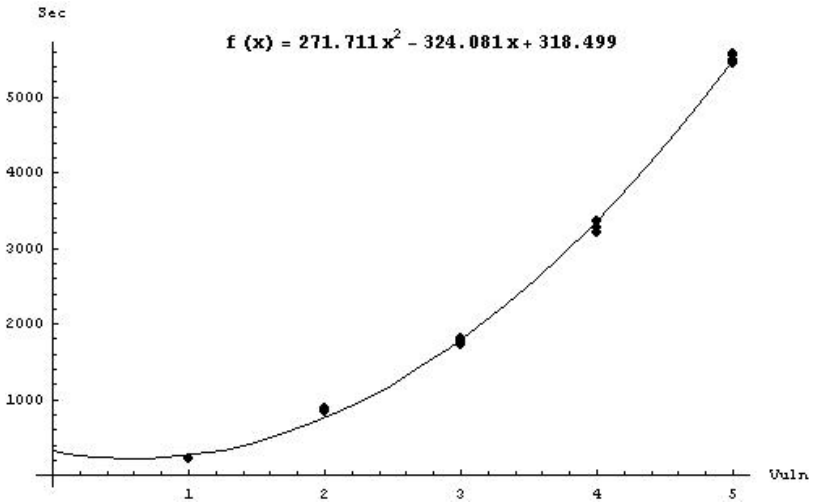

Figure A-11: Time to complete the forward algorithm as the number of vulnerabilities per node scaled

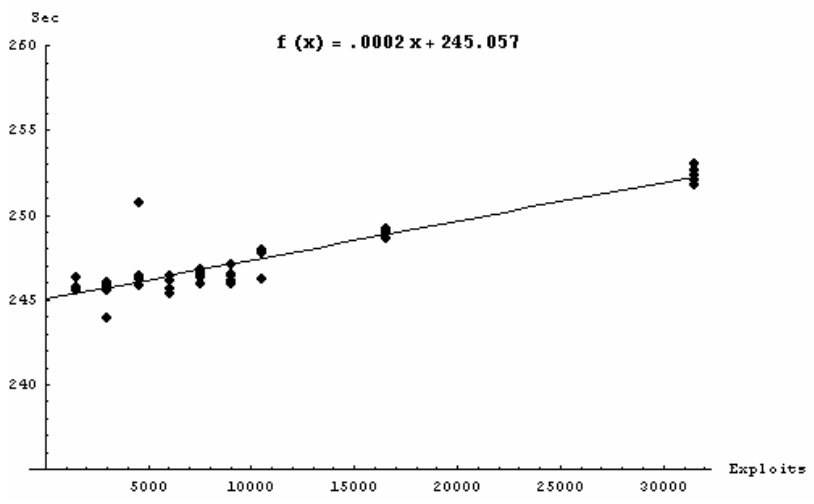

Figure A-12: Analysis time as the exploit models were added to the exploit database

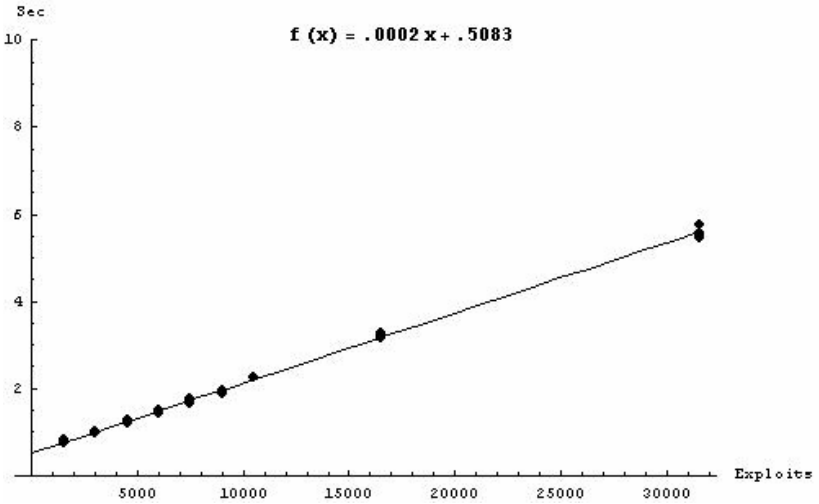

Figure A-13: Time required to perform initialization as size of the exploit database grows

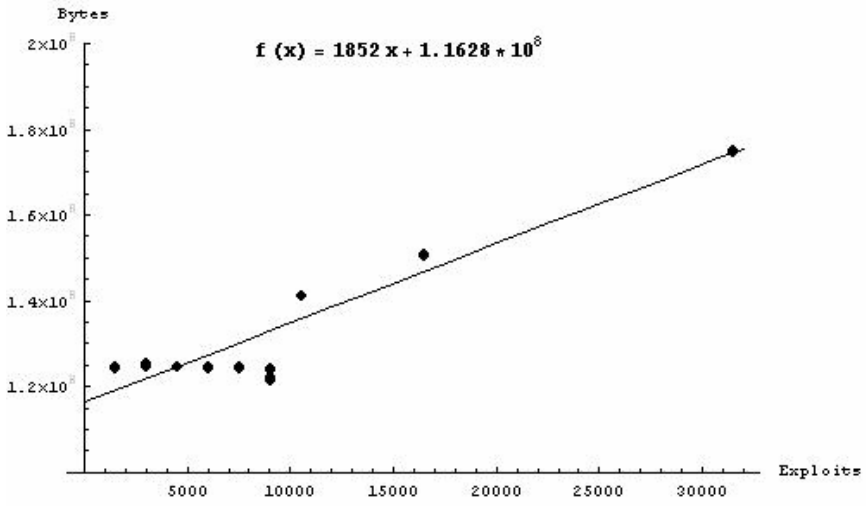

Figure A-14: Working set size as exploits are added to the exploit database