Proceedings of the 2004 IEEE
Workshop on Information Assurance
United States Military Academy, West Point, NY June 2004

The Manuka Project

Barbara Endicott-Popovsky, Dave Dittrich, Amelia Phillips, Deb Frincke, *Member, IEEE*,
Jose Chavez, W. Jenks Gibbons, Don Nguyen, Christian Seifert, Amy Shephard, Chris Abate, Shawn Loveland

During 2003-2004, the University of Washington (UW) and Seattle University (SU) collaborated to build a system for cataloging compromised system images under the auspices of the Pacific Northwest Honeynet (PNW-honeynet)[1] which is a Honeynet Project Research Alliance member group. The idea grew from the Honeynet Project's 'Forensic Challenge,[2]' a project designed to raise awareness, teach and inform those tasked with responding to threats of malicious network intrusion. Since teaching from evidence of actual incidents is far more powerful than the traditional approach of using contrived workbook exercises, the Manuka project called for the creation of a database that would store compromised system images for use in Incident Response and Computer Forensic courses. This is a case study of that development process, identifying the unique challenges overcome in completing Manuka by June, 2004. As an open source product that will be made available to the research and teaching community, it is hoped that through this paper interest will be stimulated to provide these researchers further ideas for use and enhancement.

I. Introduction

In 2003, a collaboration began between the University of Washington (UW) and Seattle University (SU) to build a system for cataloging compromised system images. The idea grew in part from the Forensic Challenge,[3] a project designed to teach incident response through hands-on analysis of a real-life compromise. [1]

The main teaching tools of the Forensic Challenge are structured questions and specific deliverables reports that focus an incident handler's attention on the important facts as they learn by "investigating" an actual compromise. The success of the Forensic Challenge led its creator, Dave Dittrich[4], to conceive of a university teaching environment that would train incident responders by using actual compromised system images, continually refreshed, in order to expose students to the most recent attacks. This was deemed a far more powerful teaching tool than the traditional approach of using contrived workbook exercises.

Key to supporting and maintaining this educational environment is a system for Cataloging Compromised System Images (CCSI), later renamed "Manuka[5]." In June 2003, a group of Seattle University graduate students, under the direction of Dave Dittrich and faculty advisor, Barbara Endicott-Popovsky[6], began the development of the Manuka Database as part of their capstone experience in Seattle University's Software Engineering Master's

Barbara Endicott-Popovsky, Lecturer, Seattle University;
Dave Dittrich, Affiliate Researcher, University of Washington
Amelia Phillips, Faculty, Highline Community College
Deb Frincke, Ph.D., Associate Professor, Computer Science Department, University of Idaho

Members of the Manuka Project Team: Jose Chavez, W. Jenks Gibbons, Don Nguyen, Christian Seifert, Amy Shepherd
Assisted by Members of the PNW-honeynet: Chris Abate, Shawn Loveland

[1] The initial focus of the Pacific Northwest Honeynet is to produce a reference set of hacked system images that will be used to seed the database of such images being built by PNW-honeynet members, the graduate students from Seattle University's Computer Science and Software Engineering Department mentioned previously.

[2] One of the Forensic Challenge's main missions is to educate incident handlers. It does so by presenting actual post compromise evidence and challenging participants "to see who can dig the most out of that system and communicate what they've found in a concise manner." The website at http://project.honeynet.org/challenge/#intro challenges participants to have fun, solve a real world problem, and learn.

[3] One of the Forensic Challenge's main missions is to educate incident handlers. It does so by providing participants with a bit-image copy obtained from a "live" honeypot that was compromised by a real intruder through the Internet and challenging participants "to see who can dig the most out of that system and communicate what they've found in a concise manner." Examples from the original participants provide a set of useful analyses for others to learn from. The website at http://project.honeynet.org/challenge/#intro challenges participants to have fun, solve a real world problem, and learn.

[4] Dave Dittrich is an Affiliate Researcher in the Information School at the University of Washington and one of the founding members of the Honeynet Project.

[5] Manuka (Maori language) honey from the Manuka tree flowers is renowned for its healing powers. It's a powerful antidote for bacteria, fungi and **viruses**.

[6] Barbara Endicott-Popovsky is a full time faculty member at Seattle University working on her doctoral research in computer forensics through the University of Idaho's Computer Science Department, and NSA Center of Academic Excellence.

program. Amelia Phillips[7] with Highline Community College (HCC) serves as a source for additional requirements from a classroom perspective. With the success of this project, the three schools (UW, SU and HCC) have extended their collaboration to develop curriculum for a complete certificate program in computer forensics, which is being underwritten through funding received recently from the National Science Foundation.

A. The Pacific Northwest Honeynet

The UW/SU project was formed under the auspices of the Pacific Northwest Honeynet (PNW-honeynet)[8] which is a Honeynet Project Research Alliance member group. PNW-honeynet currently is composed of students, faculty, and staff at the University of Washington, Seattle University, Idaho State University and the University of Idaho, along with other affiliates from the Northwest region who are involved in Honeynet research. [3]

The PNW-honeynet mission is to gather compromised system images using a distributed network of honeynets created using the Honeynet Project's GenII "honeywall" bootable CD-ROM technology.[9] The Manuka database will be used to collect and store these captured images and use them to support research programs at these and other institutions and to facilitate computer forensic course lab exercises.

B. Cataloging Compromised System Images

The initial project description called for the creation of a database that would obtain and catalog a series of compromised system images for use in Incident Response and Computer Forensic courses. Further the description stated that the database project would use, and therefore be compatible with, tools and techniques produced by the Honeynet Project.[10] The student team, operating as a Honeynet Project Alliance member group, would be required to set up and operate a series of honeypots (running Linux, Windows XP, Windows 2000, Solaris, etc.) in a controlled manner. Also, the group was required to monitor these systems to detect compromise, perform a basic forensic analysis of the host, identify a set of candidates for inclusion in the database, and produce a report on the facts surrounding the compromise to aid in cataloging the images for future study, or use as an educational tool.

There were several valuable elements of the project that created valuable student learning experiences. The balance of this paper identifies these elements in the event they might be of interest to others interested in using the tool.[11] First we will discuss the software project plan devised to produce the product, then we will look at the design of the database and user interface, and finally the administrative challenges of collecting images.

II. Software Project Plan

As part of a software engineering program, the team exercised good development practices from the beginning. As a research effort, initial requirements for this project were understandably unclear and expected to change throughout the course of the project. As a result, an agile software development process (eXtreme programming) was chosen and modified to drive development. [5]

The agile software development manifesto summarizes the team's approach:

- Individuals and interactions over processes and tools,
- Working software over comprehensive documentation,
- Customer collaboration over contract negotiation,
- Responding to change over following a plan.

Short (4 weeks) iterative development cycles were used, each completing with a release of functional software. Test driven development and close customer communication, throughout each iteration, guaranteed that proper software was built with the simplest possible design. Tracking and scope control guaranteed delivery on time as promised.

A. Iterations

The iteration is at the heart of the eXtreme programming methodology. Each short development cycle yielded a functional, releasable piece of software that performed the

[7] Amelia Phillips has taught computer forensics subjects for years in Washington State and is co-author of *Guide to Computer Forensics and Investigations*. [2]

[8] The initial focus of the Pacific Northwest Honeynet is to produce a reference set of hacked system images that will be used to seed the database of such images being built by pnw-honeynet members, the graduate students from Seattle University's Computer Science and Software Engineering Department mentioned previously.

[9] A draft guide for setting up such a honeynet can be found at http://staff.washington.edu/dittrich/pnw-honeynet/honeywall-hw-guide/ .

[10] (see http://www.honeynet.org) [4]

[11] The database developed by the UW/SU collaboration will be open source and, as such, available to other institutions interested in honeynet research.

requirements that the customer and development team agreed on at the beginning of the iteration.

In order to keep the long-term goal of the project in sight, a list of 9 iteration themes were maintained. Each theme roughly described the activity of the iteration. All 9 themes cover the entire functionality the customer and development team agreed upon at the beginning of the project. As functionality changed during the course of the project, the list of those themes was updated to reflect changes and to ensure that the long-term goals were kept visible throughout the project.

B. Iteration Methodology

Before planning each iteration, the customer selected which theme would be targeted for the current iteration (selection was constrained by dependencies among themes). After selecting the theme, the project manager, developers and customer created user stories[12] for the iteration.

With user stories in hand, iteration planning meetings were held on every 4th Wednesday starting with Wednesday 8/13/2003. During each meeting, team velocity[13] was calculated based on the last iteration. This was done by computing actual time spent for completed user stories during the last iteration for each developer, respectively. This number served as the velocity of the developer for the next iteration and no developer was permitted to accept user stories that exceeded this velocity. After calculating velocity, the user stories were prioritized by the customer and then assigned to developers starting with the highest priority user story. Each developer estimated their assignments, which were subtracted from the developer's velocity until the number reached zero. At that point the developer had all the user stories they could handle for that iteration.

Within the first couple of days after iteration planning, each developer investigated their own user stories in more detail, with the primary goal of validating the initial estimate. Besides becoming familiar with each task in more detail, activities during this stage also included writing small spike prototypes when appropriate. [14] Adjustments to estimates were relayed to the project manager, no later than the first Sunday after iteration planning. The project manager in turn would relay changes to the customer and make adjustments to the iteration plan.

Throughout each iteration, tracking was the key to creating a releasable, functional piece of software at the end of the iteration. Each Wednesday the project manager would collect information about time spent and current estimate of work remaining for each user story. This information was collected in the iteration plan documentation, which served as an overview of user stories and schedule. If estimates revealed that certain user stories would not complete on time, the project manager would contact the customer to reduce scope to the extent that functional software still could be released at the end of the iteration.

C. Risk Management

Spikes (simple throwaway prototypes) were used to address any specific technical risk that arose. Thus risk was reduced by addressing it early, and often, throughout the project lifecycle.

D. Group communication

As a graduate school team composed of working adults, members did not interact on a personal level every day. For that reason, the team held regular weekly status meetings to track iterations and discuss emerging issues. To conserve time, the project manager created a meeting agenda and published meeting minutes for each meeting.

In addition to weekly status meetings, the team scheduled customer meetings every four weeks to present the software just created in the last iteration, plan the next iteration and address any issues or concerns. Additional meetings were held as need arose. Email and the phone were a primary means of communication throughout the project.

E. Test First Driven Development

Development followed a test-first driven approach where the developer wrote the test before writing the code. This ensured that the software's design remained the simplest possible.

The test first driven development approach prevents building software for future requirements that might never arise, avoiding valuable development time being invested in features that never will be used. If new requirements do arise and the current design does not support them, the code and design can be refactored, aggressively, because surrounding unit tests will ensure the continued functionality of the software.

[12] User stories describe simple development tasks that can be accomplished in one or two ideal days.
[13] Velocity is a measurement of assignable development time during an iteration. It is calculated based on the completed development time of the last iteration. A developer should not accept user stories that exceed her velocity.
[14] Spikes are simple throwaway prototypes that address technical risks.

F. Deviations from the XP Methodology

The eXtreme programming methodology was adjusted to accommodate the unique characteristics of a graduate student school project in the following ways:

- Weekly status meetings and email communication replaced daily standups since the team didn't work together on a daily basis.
- Frequent communication with the customer through meetings, phone calls and email created an available remote customer since the customer was not "on site" during the course of the project.
- Pair programming was optional, since individual schedules were difficult to synchronize.

Besides maintaining team focus, the method described above kept the customer apprised of the project's status. It resulted in customer satisfaction, schedule performance that exceeded expectations and functionality that was fully delivered.

III. Database Design

Due to the innovative characteristics required of the application, the database design faced many challenges including the need for flexibility, adaptability, integrity and security. The database design had to be flexible enough to be easily maintained and expanded without having big impact on the business rules or requiring continual re-factoring of the design. Since images come from different operating systems--Windows, Linux, Unix--the design required the ability to store images of different file system types. It also had to provide for storage of forensic analysis data that differs according to operating system. This made the data design challenging.

To maintain integrity of the database, relationships among tables were enforced. This ensured that no data elements or file images were unrelated to a clean/compromised installation record. Lookup tables that allowed users the ability to maintain lists of possible field values were used to reduce the number of data entry errors that would lead to a useable database.

For security reasons, logs are maintained by the system identifying when and who accesses the system and what actions they performed. The design incorporates timestamp fields and the name of each user who adds/modifies data, along with a detailed activity log for each table considered critical.

The Entity Relationship model for the database design is shown in Figure 1. Initially the team identified the main entities, their attributes and relationships. By studying the

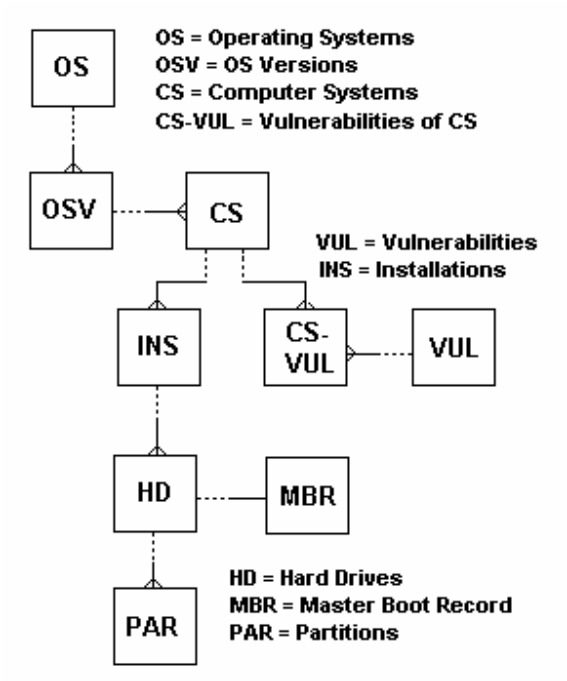

Figure 1. Current ERD Diagram.

different file systems used by Windows and Linux, the team determined additional required attributes for compromised image searches. Prototypes of reports and screens helped develop the customer's data requirements further.

The main entity in the data model is the **installation**, defined as a unique configuration of a computer system

including information about the vulnerabilities, operating system and operating system version installed. An installation can be clean or compromised and a unique ID number is generated for each installation uploaded into the system. This same ID is used later for the download process.

The **computer system** entity relates installations of the same computer system because an installation can be either clean or compromised. A computer system has **vulnerabilities** and the **operating system version** for its particular installation. Currently, the database of images can be queried by operating system, operating system version, vulnerability and computer system name.

Further work will require analyzing and designing tables for the forensics data characterizing compromised installations. This will necessitate adding more columns to the tables for additional search fields. A taxonomy of attacks should also be part of the database, as well as the type of exploits and attacks characteristic of a particular computer system installation. Further feedback from users

and additional input from computer forensics experts are needed in order to expand the current database design.

IV. User Interface Design

As explained above, since minimal requirements were known at the beginning of the project, a modification of the eXtreme Programming (XP) software development process was employed, where small pieces of functionality were implemented and delivered in frequent iterations. This raised two issues early in the project: 1) how to deliver the user interface in small pieces over many iterations and 2) given the previous constraint, what type of user interface would be appropriate.

Early iterations focused on mitigating the project's highest risks by delivering proof that the team's approach to uploading, downloading, and storing images would succeed. All initial efforts concentrated on implementing functionality; thus minimal time was available for user interface design. Therefore, it was decided that the initial user interface would be as simple as possible and a secondary focus during the early iterations. As functionality matured during later iterations, the user interface grew with the application and became a more primary focus. Delivering the user interface in small pieces, over many iterations, proved an ideal implementation technique.

Deciding on the type of user interface was a perceived risk that the team addressed early in the project. The team debated the virtues of using a Java Swing interface over a web interface. The XP process surfaced the issues and one main requirement emerged: that the application architecture had to be flexible enough to support different interfaces easily. The team deferred the balance of user interface design decisions until more information was available later. This allowed focus on higher risk items.

A. Implementation

During initial iterations, the application had a very simple command line interface that required users to enter several parameters. Only uploading and downloading system images was supported. This was acceptable during the proof of concept phase. Follow on phases, while still focused on functionality, added a more user friendly command line script that prompted the user for information.

When the application's core functionality had been implemented, team attention turned to creating a more user friendly interface. A Java Swing interface was agreed upon. The application was implemented using a tiered architecture: i.e., separate tiers for the user interface, application functionality, data access, and the data itself. This designs-in flexibility, allowing for easy additions of new types of interfaces and making support of several different user interfaces very maintainable. This proved true as the team migrated the basic functionality through support by a command line interface to a user interface implementation in Java Swing.

As the complexity of additional functional requirements increased (i.e., search requests and editing stored data), functionality was added to the Java Swing user interface, only. The command line interface continues to provide access to basic functionality of the application, which enables users to execute basic features through scripting.

B. Future Options

Having the user interface grow with the application proved to be ideal for this product. Core application functionality was built first, with the user interface evolving as the application's complexity increased. Architecture issues were addressed up front, but implementation decisions were deferred until time of implementation. This resulted in a very flexible design and implementation that would easily support future enhancements to either add new functionality to the application, or to add new types of interfaces such as a web interface

V. Storage of Compromised System Images

The storage and transfer of both clean and compromised system images presented design and development challenges for the Manuka database. A typical system image from a honeypot can range from 800 megabytes to 4 gigabytes depending on the operating system, applications and services installed.[15]

The initial design stored these images in database tables as binary large objects, or BLOB's, establishing the database as an independent filesystem for managing these images. With this model the database was responsible for both security controls on the images, as well as handling concurrent multiple access to these images. The advantages of using this model included centralized security, transactional control to ensure system image consistency with meta-data, a platform independent filesystem and the built-in ability of a database to handle multiple concurrent access. This design however, proved unusable for a variety of technical reasons, some the direct result of technology choices for the Manuka system.

[15] Although many servers in a real world situation may be much larger in size, this project focused on system images for honeypots.

Having chosen MySQL [16]as the database system, the team discovered that the maximum practical size for a BLOB was 2 gigabytes; this was inadequate for handling the sizes of some system image files. Further, the additional requirement to use encryption would create a roughly 50% increase in image download/upload time. Testing a 2-3 gigabyte system image, transferring between two systems on a fast Ethernet LAN (100 mb/s), took approximately 25 minutes. Including MD5 sum calculations and the comparison of MD5 sums to ensure integrity, the average transfer time on the testbed was approximately one hour. With the additional overhead of encryption using MySQL, the time would be too long in the team's opinion.

In order to solve both the storage issue, as well as download/upload performance problems, the team adopted a hybrid solution. Instead of containing system images as BLOB's, the database tables contained pointers to the location of these images on the filesystem. This gives several benefits. 1) File storage could be moved from the database server, giving the ability to create a distributed system where the file store could be spread across multiple servers, SAN and/or NAS devices. 2) Unlike with BLOB's, this provides a much larger, more practical file size limitation for storing compromised images, dependent on the underlying file system of the operating system as opposed to the database.

This solution had its own set of technical challenges, however. Unlike the database solution, the underlying filesystem assumes responsibility for security control of the images. This risk was mitigated by creating a system or daemon account that owns the images. Access to these images is handled with a database permissions table; compromised system images are stored in encrypted form, thus only MD5 hashes are stored in the database as meta-data for these images.

Image size also presented technical and practical challenges. Streaming multi-gig files concurrently to 30 or 40 systems would be somewhat taxing to any network and, as with any archival system, the accumulation of system images over time will result in substantially large storage requirements. Manuka addresses this issue by creating a compressed filestream for network transfer. In order to maintain integrity, an MD5 sum is generated from the raw bytes before compression as well as on the compressed image. Both are compared for integrity and stored as additional meta-data about these images. By taking advantage of the stream libraries inherent in the Java IO package, the performance hit was very minor.

[16] Since the system will be used primarily in academic settings, this choice was deemed cost effective.

The actual file transfer functionality of the Manuka system is handled by a custom file transfer utility modeled after the TFTP protocol. The transfers are protected using SSH tunneling with strict host key checking and RSA keys. Using the built in functionality of SSH for encryption added roughly 10% to the transfer times, as opposed to over 50% with the database encryption streams.

VI. Lesson Learned: Planning Effectively for Honeypot Data Capture

As a lesson learned and to assist others wishing to deploy the Manuka solution in the future, the team discovered several administrative obstacles when deploying honeypots for data capture. These were encountered when attempting to capture a new rsync exploit that surfaced in December 2003. To collect the exploit, a honeypot running Red Hat 9.0 and the vulnerable version of rsync was deployed. After a week with no compromise, the team investigated moving the honeypot to the public side of the university's network. (The honeypot was located on the university's private network behind a firewall.) While this machine was still at risk for compromise, the private subnet and firewall significantly decreased the ability to capture compromised system images effectively.

Positioning the honeypot on a public network seemed simple enough and would have provided an opportunity to collect the exploit and a compromised image; however, there were several administrative obstacles to overcome that the team had not anticipated: 1) obtaining the resources necessary to deploy the honeypot [i.e. it can take months to obtain public IP's] and 2) obtaining permission from the organization to deploy the honeypot external to the firewall.

The team learned a valuable lesson in both planning and in anticipating the legal confusion surrounding honeypots. At project inception the team should have requested permission in writing to deploy honeypots on the public side of the school's network and should have applied for public IP addresses. In addition, the team might have taken the initiative on researching the security of honeypots and the legality of deploying them.[17]

The Manuka project solves the issue of capturing and storing compromised system images; however, the team has experienced limitations to data capture on a private network due to administrative concerns. As a lesson learned, settling these issues within an organization structure is time consuming and should have been

[17] Some information may be obtained at *The Honeynet Projects'* site (http://www.honeynet.org/misc/project.html).

resolved at the outset. Hopefully, others participating in this project can learn from this lesson.

VII. Future Directions

As mentioned earlier, work will be needed to add search fields to the database tables to include forensic data that will further characterize compromised installations. A taxonomy of attacks, as well as the type of exploits and attacks characteristic of a particular computer system installation, will also be required. Feedback from users and additional input from computer forensics experts will help with this expansion to the database design.

Having the user interface grow with the application resulted in a very flexible design and implementation that will easily support future enhancements to either add new functionality to the application, or to add new types of interfaces such as a web interface. The team expects to leave a list of future enhancements when this phase of the project ends officially in June 2004.

As it is now, the database is available to the PNW-honeynet and the Honeynet Project Research Alliance for capturing and storing clean and compromised system images. These images can be available to researchers and to those interested in teaching computer forensics using actual compromised systems.

VIII. References

[1] Dave Dittrich, The Honeynet Project: The Forensics Challenge(Retrieved from the Web March 20, 2004). http://www.honeynet.org/challenge/#intro .

[2] Phillips, Nelson, Enfinger, Stuart. (2003). *Guide to Computer Forensics and Investigations*. New York: Course Technology.

[3] Dittrich, Dave: Pacific Northwest Honeynet (Retrieved from the Web March 20, 2004). http://staff.washington.edu/dittrich/pnw-honeynet/ .

[4] Honeynet Project (Retrieved from the Web March 20, 2004). http://www.honeynet.org .

[5] Beck, Kent. (1999), Extreme Programming Explained: Embrace Change. New York: Addison-Wesley.