Automated Tracing and Visualization of Software Security Structure and Properties

Wenbin Fang Barton P. Miller James A. Kupsch
Computer Sciences Department, University of Wisconsin
Madison, WI, USA
{wenbin,bart,kupsch}@cs.wisc.edu

ABSTRACT

Visualizing a program's structure and security characteristics is the intrinsic part of in-depth software security assessment. Such an assessment is typically an analyst-driven task. The visualization for security analysis is usually labor-intensive, since analysts need to read documents and source code, synthesize trace data from multiple sources (e.g., system utilities like lsof or strace). To help address this problem, we propose SecSTAR, a tool that dynamically collects the key information from a system and automatically produces the necessary diagrams to support the first steps of widely-used security analysis methodologies, such as Microsoft Threat Modeling and UW/UAB First Principles Vulnerability Assessment (FPVA). SecSTAR uses an efficient dynamic binary instrumentation technique, self-propelled instrumentation, to collect trace data from production systems during runtime then automatically produces diagrams. Furthermore, SecSTAR allows analysts to interactively view and explore diagrams in a web browser. For example, analysts can navigate the diagrams through time and at different levels of detail. We demonstrated the usefulness of using SecSTAR to produce FPVA-style diagrams for a widely used and complex distributed middleware system, the Condor high-throughput scheduling system. Compared with the original manual approach in FPVA, SecSTAR shortened the initial diagram construction time from months to hours and constructed a more accurate diagram visualizing the complete runtime structure of Condor.

1. INTRODUCTION

In-depth assessment of software security is typically an analyst-driven task, and visualization of the program's structure and security characteristics is the intrinsic part of this assessment task. For example, two widely used assessment methodologies, Microsoft Threat Modeling (MTM) [15] and the UW/UAB First Principles Vulnerability Assessment (FPVA) [5] both require developers or analysts to construct diagrams of software systems to represent key architectural components of the system, interactions between these components, the privilege levels of each component, delegation of privilege, and how components interact with high-value resources such as files, databases and external services. These diagrams are used as a road map for later in depth (often manual) inspection of the code.

However, constructing such diagrams requires extensive manual effort, causing the analyst to conduct a careful reading of source code looking for key operations such as those that operate on files and sockets or modify privilege levels. These manual static inspections of the source code are often augmented by trace data from a variety of tools such as lsof or strace. Data is collected and synthesized from a variety of sources, in different formats, to provide the basis for the analyst to manually draw the various structural diagrams. To help address this problem, we are developing an approach to dynamically collect the key information for such diagrams and automatically produce the necessary diagrams to support the first steps of MTM or FPVA. This approach is embodied in a tool called the Security System Tracing, Analysis and Reporting (SecSTAR).

SecSTAR operates in two steps, data collection and visualization. We leverage a flexible and efficient dynamic binary instrumentation technique, self-propelled instrumentation [9], to collect trace data from production systems. Using self-propelled instrumentation, SecSTAR produces trace data for analyst-specified events in unmodified production systems during runtime. We monitor and trace key events, such as those involved with process creation and destruction, socket creation and connection, privilege level changes, and file I/O operations. The trace data also contains temporal information, so that the later analysis and visualization steps can analyze for time-based risks such as race conditions and animate the visualizations of the program structure.

The second step of SecSTAR is a postmortem operation that visualizes the trace data generated from the data collection step. In the visualization, the trace data is used to generate SVG [18] diagrams using Graphviz dot layout [2], then these diagrams can be interactively viewed and explored in a web browser. The SecSTAR visualizer allows the analyst to navigate the diagrams through time and at different levels of detail. This process is summarized in Figure 1.

To evaluate the effectiveness of SecSTAR, we conducted a case study by using it to automatically produce FPVA analysis diagrams for a real world, widely used, and complex distributed middleware system: the Condor high-throughput

VizSec'12 October 15 2012, Seattle, WA, USA

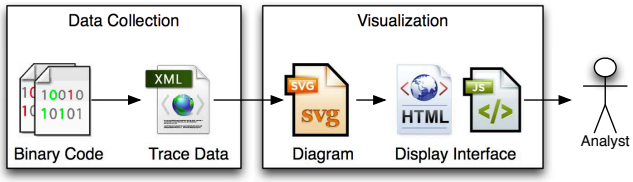


Figure 1: Data flow of SecSTAR

scheduling system [6, 16]. We had previously conducted an in depth, manual FPVA assessment on Condor. This assessment was conducted over a nine month period by an experienced analyst, and resulted in finding several serious vulnerabilities that were not able to be found using current source code analysis tools [4]. As part of the FPVA assessment, the analyst manually produced the associated analysis diagrams. Our use of SecSTAR allowed us to compress these initial steps of FPVA to a brief and interactive task, significantly improving the productivity of security analysts.

This paper is organized as follows. Section 2 surveys related topics on visualization for security analysis and introduces FPVA and MTM. Section 3 and Section 4 describe the two steps of SecSTAR: data collection and visualization. The case study on constructing FPVA diagrams is presented in Section 5. We conclude and list our path for future work in Section 6.

2. RELATED WORK

This section first surveys previous work on visualization in security analysis, then we introduce two widely-used security analysis methodologies that could benefit from SecSTAR, Microsoft Threat Modeling (MTM) and UW/UAB First Principles Vulnerability Assessment (FPVA).

2.1 Visualization for Security Analysis

Visualization in security analysis is typically divided into two phases, data collection and visualization. We survey four pieces of related work [3, 17, 19, 20] and compare our SecSTAR with them.

Xia et al. [19] visualize program flow to aid forensics experts in detecting and identifying significant events for intrusion detection and recovery. For the data collection phase, they modify the Linux kernel to trace system calls related to process creation and file access. By contrast, our SecSTAR performs lightweight data collection using self-propelled instrumentation that collects data only for user-specified processes and directly instruments the program binary without modifying the kernel. In addition, they store trace data in a relational database system for ease of data analysis, while SecSTAR stores trace data in XML files that are both program- and human-readable and can be easily disseminated to the public. For the visualization, they show process trees, where each process is a node, and its child processes and accessed files are child nodes of it. They highlight the processes and files that are tainted after taint propagation for estimation of damage suffered following an intrusion. They do not include information about interprocess communication or privilege level of each process. Their visualization is static, showing temporal information only for file access events.

Yurcik et al. [20] designed and implemented a visualization framework, NVisionCC, to help system administrators monitor the security status of a cluster. NVisionCC only monitors and visualizes process status, because they believe that an attacker cannot gain access to cluster nodes without running at least some processes, and the appearance of unexpected processes on a cluster node is a strong indicator that the security of the node has been compromised. The visualization in NVisionCC is coarse-grained, showing only the presence or absence of certain processes. NVisionCC does not visualize relationships among processes (e.g., the parent-child relationship and communications) or other security characteristics (e.g., privilege level changes).

Fink et al. [3] designed a tool to help system administrators visually correlate process activities and network traffic for ease of identifying processes that have communication patterns outside their normal behaviors. The data collection step is performed by either using a modified Linux kernel or third party tools on Windows such as IP Helper [7] and WinPcap [13] that can monitor only a limited number of events. For the visualization, they show a client/server layout of communications, where client processes are on the left side in the display and server processes are on the right. Edges connecting two processes represent network communications. Their visualization does not show process hierarchy and other important security characteristics (e.g., privilege level).

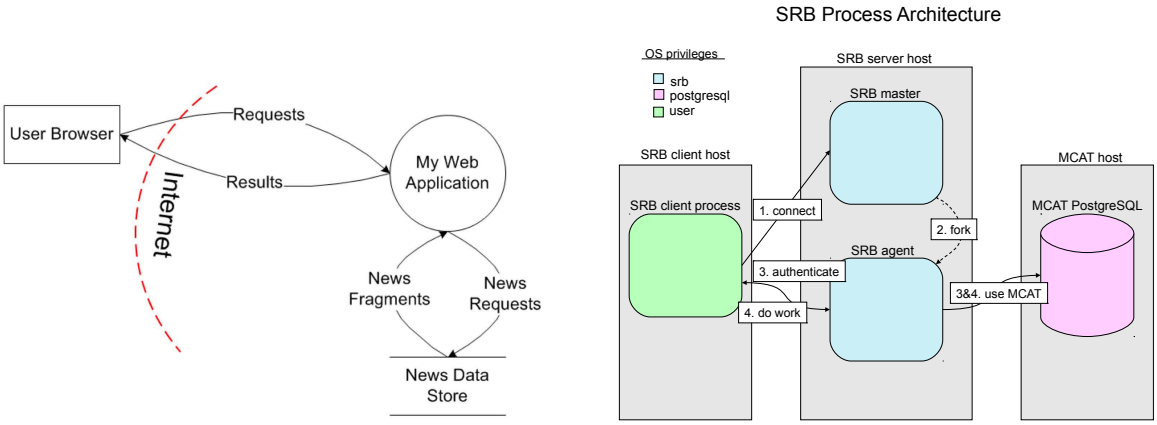
Trinius et al. [17] designed an automatic technique to analyze and visualize malware behavior. In the data collection phase, a single process runs in a sandboxed environment, where the system calls made by the process, such as changes to the filesystem or network activity, are recorded. Their visualization approach uses a treemap to summarize the actions performed by the malware sample and a thread graph to summarize the temporal behavior of each thread in the sample. Their approach does not support distributed systems and animation.

2.2 MTM and FPVA

MTM [15] is a security analysis methodology that is aimed at identifying and rating the most likely threats affecting an application. This methodology requires developers to use the Threat Modeling Tool [8] to manually create a high-level architecture diagram that depicts the structure of software systems, including subsystems and physical deployment characteristics. Figure 2a[1] is an example of such diagram. After developing the architectural overview of the application, this methodology applies a list of pre-defined and known possible threats and tries to see if the application is vulnerable to these threats. Microsoft suggests that the developers participate in the threat identification.

FPVA [5] is an analyst-centric (often manual) technique. It aims to focus the analyst's attention on the parts of the software system and its resources that are most likely to contain vulnerabilities. Initial steps in FPVA produce diagrams that show the system's key architectural components, interactions between these components, each component's privilege level, privilege delegation, and how components interact with high-value resources. These diagrams form the basis for

[1]From http://technet.microsoft.com/en-us/security/hh855044.aspx



(a) MTM (b) FPVA

Figure 2: Sample Diagrams of MTM and FPVA

a risk assessment of the system, identifying which parts in the system most immediately need evaluation. Figure 2b is an example of such diagrams, showing the key components of the Storage Resource Broker (SRB) [1, 14], the interactions between these components, and each component's privilege level. FPVA has a different philosophy from MTM, as it is performed by an assessment team independent from the developers. Nevertheless, the analysis products produced by both methodologies are quite similar.

As we design SecSTAR to be general purpose for security analysis methodologies, security analysts can apply it to both MTM and FPVA. The case study in our paper focuses on generating diagrams of FPVA. Nevertheless, analysts can create diagrams in the MTM style by slightly modifying the script in SecSTAR.

3. DATA COLLECTION

We discuss the data collection step for SecSTAR in this section. We introduce self-propelled instrumentation as an infrastructure, then we describe the specific tool based on this infrastructure for collecting trace data in SecSTAR.

3.1 Self-propelled Instrumentation

Self-propelled instrumentation is a binary instrumentation technique that dynamically injects a fragment of code into an application process on demand [10,11]. The instrumentation is inserted ahead of the control flow within the process and is propagated into other processes, following communication events, crossing host boundaries, and collecting a distributed function-level trace of the execution. Self-propelled instrumentation contains two major components. The first is the *Agent*. It is a shared library that automatically inserts and propagates a piece of payload code at function call events in a running process, where the payload code contains user-defined logic, such as generating trace data for later inspection. The second component is the *Injector*. It is a process that causes an application process to load the Agent shared library, where the Injector should have at least the same privilege as the application process. Self-propelled instrumentation does binary instrumentation within the application process's address space, avoiding use of the debugging interfaces (e.g., Linux ptrace and Windows debug interface) and costly interprocess communications. Therefore, self-propelled instrumentation does not add significant overhead to a process during runtime.

To enable data collection in a system, analysts use the Injector to load the Agent into the system's processes, so that the payload code in the Agent can be triggered to collect trace data. The earlier that analysts load the Agent into the system in the communication flow, the more complete trace data they can collect, thus the more complete architectural diagram they can construct from the trace data. For example, in a client/server system, if analysts inject the Agent into a client process that is about to make a request to a server process, then they can capture a complete flow of events resulting from that request.

Previously, we prototyped and used self-propelled instrumentation for problem diagnosis in distributed systems [9]. In our current project, we have re-engineered self-propelled instrumentation for robustness and portability, and now we have been applying it to security analysis.

3.2 Data Collection Tool

With self-propelled instrumentation, security analysts can easily build custom data collection tools by writing payload code in C or C++. The payload code is encapsulated in a payload function that will be invoked at function call events. In a payload function, analysts can use self-propelled instrumentation's API to get the arguments and the return value for a function call where the payload function is invoked. The most useful functions to instrument are typically those in system libraries. For example, when a Linux user process invokes *fork*, we can get the process ID (or PID) of the newly created child process from *fork*'s return value and write a trace record with a variety of information, including the parent and child PID's. The payload code can also inspect the */proc* file system to get additional information about the current process, such as the executable name and command line arguments.

Each instrumented process produces an XML file containing trace data that records two types of information. First, it records the static information about a process at the time the Agent is loaded, such as the executable name and PID. Second, it records events as they occur during operation of the system. For each event, it records a timestamp (in microseconds) for visualizing temporal data and the data specific to an event type, such as PID for starting a new process, or host and port information when initiating a network connection. In our current design, SecSTAR supports a variety of Linux events, including process creation (*fork*, *clone*, and *fork-then-exec*), process destruction (*exit*), privilege level changes (*setuid*), connection establishment (*connect* and *accept*), network communication (*send* and *recv*), and file access (*open*, *read*, *write* and *close*).

4. VISUALIZATION

Visualization in SecSTAR is a postmortem operation on the per-process XML trace data collected during runtime. SecSTAR constructs diagrams in SVG format, mapping from trace data to a representation that is rendered on the display. Security analysts then use a web-based interface to view those diagrams and interactively explore details in them for security analysis.

4.1 Diagram Construction

SecSTAR first parses all per-process XML trace data then extracts necessary information for generating diagrams using Graphviz dot layout that is suitable for representing hierarchical or layered drawings of directed graphs.

Parsing XML trace data, SecSTAR extracts time-ordered events (e.g., process creation and connection establishment) and each process's static information such as the executable name, PID, and initial privilege level. Based on the extracted information, SecSTAR uses Graphviz to produce directed diagrams in SVG format visualizing the system architecture and security characteristics. The reason to use SVG format is two-fold. First, as SVG is an XML-based graphic format, we are able to render it in browsers and run JavaScript to control the way an SVG image is displayed. For example, we can programmatically move a packet along a communication line in the diagram. Second, an SVG image can be scaled without degrading quality because it is vector-based.

SecSTAR aims to generate simple, informative, and compact diagrams for security analysis. Figure 3 shows a sample diagram generated by SecSTAR. The major elements in the diagram are concise, including only circle nodes (processes), edges (relationships among processes), and rectangles (hosts). A variety of security characteristics can be visualized on the same diagram by augmenting basic elements with labels, colors and styles. For example, different edge styles represent distinct relationships among processes (e.g., the parent-child relationship and network communications), and the node color represents a process's privilege level. Furthermore, we apply two optimizations in the diagram to make it compact, so that the display space is well utilized. First, we use one or a few letters in a node to abbreviate the name of the process's executable. By doing this, the node size stays small, making the whole diagram compact. Looking at these letters, security analysts can easily identify the processes having the same executable and refer

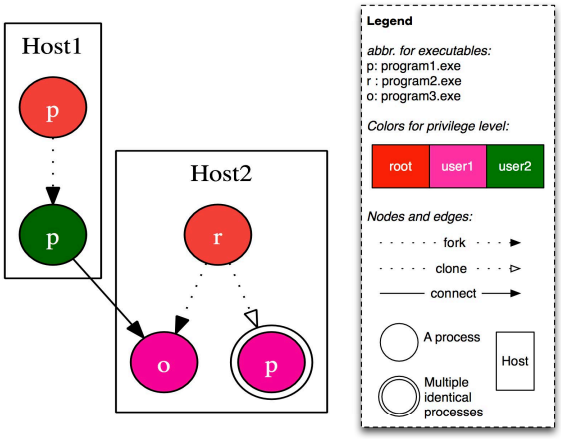


Figure 3: Sample diagram generated by SecSTAR

to the legend for the full name. Second, we use a double-circle node to represent a group of nodes having the same parent, the same executable, and the same privilege level. This optimization prevents the diagram from becoming too wide and too sparse when there are many identical child processes in the system, for example, when a server process forks a new child process to handle each request.

The diagrams in SVG format are the building blocks of animating temporal data in the system. For our current implementation, SecSTAR generates a list of static SVG diagrams, each of which is a snapshot of the system architecture after an event is triggered. Thus, each diagram becomes a frame in an animation that will be achieved when these diagrams are displayed in sequence in a web browser. In the future, we plan to directly animate elements in a SVG diagram using JavaScript for more efficient storage and processing.

4.2 Interactive Interfaces

SecSTAR's web-based interface displays the architecture, resource, and privilege diagram in the middle, descriptions of events on the left, the legend on the right, and control widgets on the bottom (Figure 4). SecSTAR provides five types of controls for the security analyst, **Navigate**, **Elaborate**, **Annotate**, **Filter**, and **Relate**. In this section, we describe these controls.

Navigate. Using SecSTAR, security analysts can navigate through changes of the system's runtime structure and security characteristics over time, which enables analysts to explore temporal event patterns for anomalous behaviors in the system or to get an overview of the system's runtime behavior. SecSTAR provides two forms of **Navigate**. First, analysts can manually navigate by clicking on *Previous* and *Next* buttons or adjusting the slider bar that represents the timeline for the system's execution lifetime. Second, they can click on the *Play* button to enable animation of the history.

Elaborate. After getting an overview of the system's runtime behavior via **Navigate**, analysts can explore details of particular components in the system on demand. SecSTAR allows analysts to zoom in or out of the diagram. Zooming changes the scale of the diagram but does not fun-

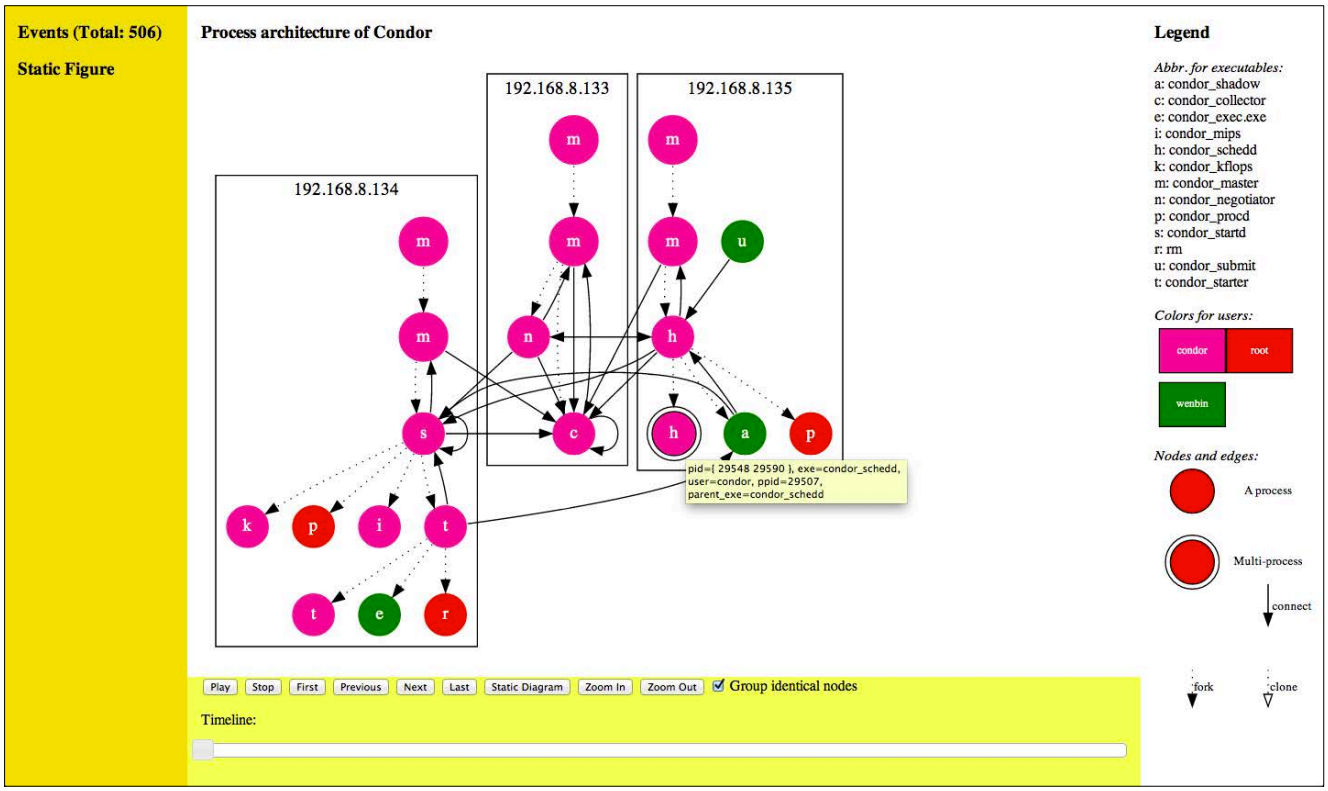


Figure 4: A screenshot of SecSTAR interactive interface

damentally alter the representation. Furthermore, when the analyst hovers the mouse over an element on the diagram, a tooltip is shown that provides detailed information about that element such as the full executable name and PID for a process.

Annotate. It can be difficult to infer the accurate semantics of elements on the diagram, e.g., the exact intention for a specific operation such as why the system changes a process's privilege at a certain point, or why the system initiates a network connection. To further improve its visualization, SecSTAR allows analysts to annotate the diagram by creating a label for an element, incorporating information from other sources such as development documents and source code.

Filter. Analysts can narrow down the set of elements being displayed on specific conditions such as showing only processes from a particular host, having a particular privilege level, running the same executable, or accessing a particular file.

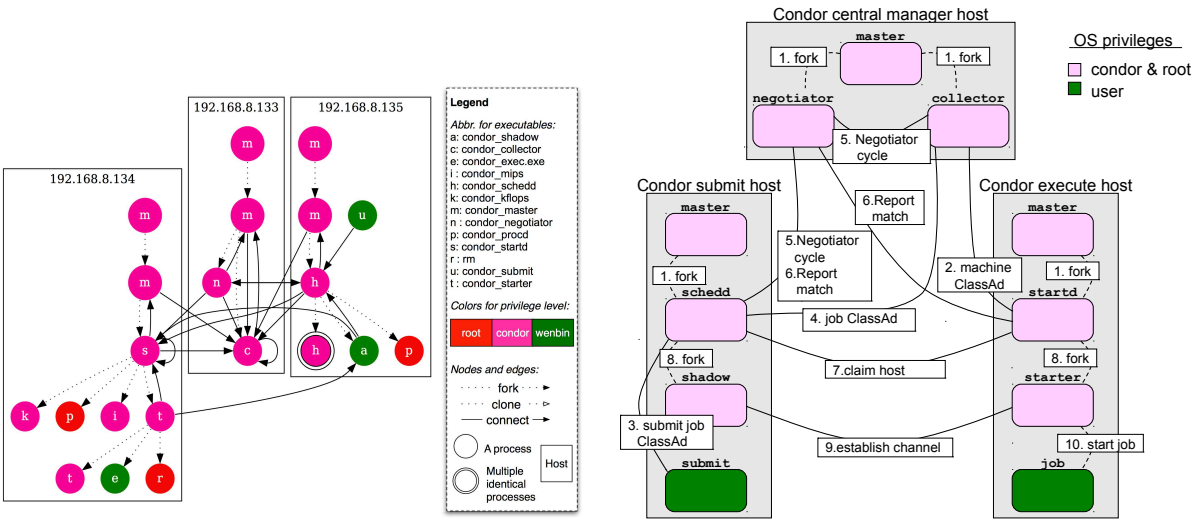
Relate. When analysts click on an element on the diagram, SecSTAR can automatically highlight related elements, such as a node's immediate neighbor nodes and all edges incident to it, all nodes with the same color as a node, and all edges with the same style as an edge. **Relate** saves the analyst's time to figure out questions like what processes are talking to this particular one, what processes are children of this process, and whether this process connects to a remote host. **Relate** is especially useful when there are many crossing communication edges.

5. CASE STUDY

We conducted a case study using SecSTAR to produce FPVA-style visualization for a complex, real world middleware system, the Condor high-throughput scheduling system [5, 12]. Condor is used worldwide to enable scientists, engineers, financiers, and social scientists to reliably execute extremely large and complex computational jobs. In this section, we describe the use case of generating diagrams using both SecSTAR and the original manual approach from FPVA. We then compare the cost of constructing diagrams and the quality of these diagrams in both approaches. Finally, we discuss the use of SecSTAR's interactive interface.

In our study, we used SecSTAR to recreate the diagrams created during the original manual assessment of Condor using FPVA in 2005. The use case diagrammed is starting the Condor system on all hosts, submission of a job by a user, and the execution of the job. The system diagrammed was a simple three host Condor system with one host acting as the central manager, one as a submit host, and the last as an execute host.

The original study using FPVA was a manual process where an experienced security analyst collected information about the system under study, and then assembled these artifacts into manually created diagrams. The analyst used multiple means to collect this information including reading documentation and presentations, and talking to developers and knowledgeable users. As the information from these sources is often incomplete or inaccurate, the analyst refined and enhanced this information by observing a running sys-



(a) FPVA Diagram by SecSTAR

(b) Original FPVA Diagram

Figure 5: Comparison of FPVA diagrams between SecSTAR and the original manual approach

tem using log files, debugging techniques, and system analysis tools such as ps, strace, and lsof. Additionally the analyst selectively read portions of the code to further gain understanding of the system. Finally, after all the artifacts were gathered and understood, the diagrams were created by hand using a drawing tool, such as Microsoft PowerPoint.

The time to gather the information necessary to draw the diagrams is on the order of months for a system of the complexity of Condor. The Condor system has more than a thousand pages of documentation and more than 700,000 lines of source code. Gathering information from the running system is further complicated due to the multiple processes and especially the multiple host nature of Condor, as system analysis tools do not operate across host boundaries. The information gathered is in a multitude of formats including free form notes, log files, debug output, and various tool output. The analyst must correlate and distill these artifacts to the information required for the desired diagrams.

In contrast, an analyst using SecSTAR to create an initial diagram just needs to be familiar enough with the system to run it and make it perform something useful. The analyst injects self-propelled instrumentation's Agent into the system under study. SecSTAR automatically traces system behaviors, and diagrams are constructed automatically. SecSTAR is able to automatically collect data across hosts due to self-propelled instrumentation's ability of propagating payload code. Moreover, SecSTAR collects data in a structured XML format that can be automatically processed and visualized.

In SecSTAR, an analyst can create an initial diagram for a system such as Condor in hours, with most of that time being used to learn how to install and operate Condor, and perform some simple tasks such as submitting a job. Once this is learned, the subsequent creation of diagrams is on the order of minutes. The initial diagrams created by SecSTAR may not be quite as useful as the initial diagrams created by the manual approach, due to the lack of in-depth study of the system; however, the diagram is a good start. As analysts gain experience with the system with the aid of these diagrams, they can iteratively improve the use case to produce more complete diagrams. To guide the iteractive process, the analyst will still need to consult documentation and the source code, but their initial diagrams should reduce this burden. The low cost of diagram construction encourages analysts to visualize many more use cases. Another technique we expect to be fruitful is to run the system multiple times with either the same or distinct runtime configurations, which makes it easy for analysts to compare runtime instances and visually identify abnormal behaviors.

Figure 5 compares the Condor diagram generated by SecSTAR with that from the original manual study. Overall, the diagram generated by SecSTAR (Figure 5a) captures more information than the original FPVA diagram (Figure 5b) and faithfully reflects Condor's runtime structure. SecSTAR produces a more complete diagram by identifying all processes in Condor including undocumented processes (e.g., condor_mips in Figure 5a but not in Figure 5b) that are difficult to discover using the original approach, where analysts fail to find these processes either from documents or from the process of manually synthesizing trace data.

The diagrams produced by SecSTAR have two problems that are common to any automated approach but can be alleviated by our interactive interface. First, although the diagram of SecSTAR is more complete, it may include some processes in which the security analysts are not interested. This problem can be solved by the **Filter** feature in SecSTAR's interface, where analysts can control what to display in the diagram and hide those distracting elements. Second, SecSTAR diagrams lack semantic labeling of events, such as those found on edges in Figure 5b (e.g., submit job ClassAd). **Annotate** solves this problem, because it enables analysts to annotate the diagram by incorporating information from external sources.

Condor's structure, including process creation and destruction, and security characteristics, including privilege changes, evolve dynamically during runtime. However, the diagrams from the original FPVA provide a static view of

Figure 6: An animation of Condor's runtime progression; a dotted line connecting two screenshots indicates that intermediate screenshots are skipped

Condor showing limited progression information, with only numbered edges to order *fork* and communication events by time. By contrast, the **Navigate** feature in SecSTAR's interface allows analysts to walk through a variety of information in Condor's progression during runtime such as privilege changes, process creation and destruction, and file access. Figure 6 shows a series of diagrams illustrating the use of **Navigate** in SecSTAR to animate Condor's runtime progression.

6. CONCLUSION AND FUTURE WORK

We have presented a tool, SecSTAR that automatically traces and visualizes software security structure and properties, to support first steps of security analysis methodologies. Using self-propelled instrumentation, SecSTAR can automatically perform data collection in a distributed system during runtime right before important events occur. The web-based interface provided by SecSTAR allows analysts to interactively explore security properties in the diagrams that are generated based on the trace data. Our case study of using SecSTAR to produce FPVA diagrams for Condor demonstrates the efficacy and efficiency of SecSTAR, where the time for the initial diagram construction is shortened from months in the original manual approach to hours. Furthermore, interaction features provided by SecSTAR's interface enable analysts to view and explore diagrams through time and at different levels of detail, significantly improving the analyst's productivity.

We are currently extending SecSTAR to monitor and visualize more events in the system, for example, to support more inter-process communication mechanisms (e.g., pipe, shared memory, and UDP). We are exploring approaches to represent temporal order in static diagrams. We are also improving SecSTAR's interface by implementing more interaction features. Our ongoing work also includes integrating SecSTAR into both FPVA and MTM for assessing real world grid and cloud systems.

A demo of visualization provided by SecSTAR is online: *http://research.cs.wisc.edu/mist/projects/SecSTAR*

7. ACKNOWLEDGMENTS

We thank the reviewers for their insightful comments and suggestions. This research funded in part by Department of Homeland Security grant FA8750-10-2-0030 (funded through AFRL), National Science Foundation grants OCI-1032341, OCI-1032732, and OCI-1127210, and Department of Energy grants DE-SC0004061 and DE-SC0002154.

8. REFERENCES

[1] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, Toronto, Ontario, Canada, Nov.–Dec. 1998.

[2] Emden Gansner and Eleftherios Koutsofios and Stephnen North. *Drawing graphs with dot.* `http://www.graphviz.org/Documentation/dotguide.pdf`.

[3] G. A. Fink, P. Muessig, C. North, and C. North. Visual correlation of host processes and network traffic. In *IEEE Workshop on Visualization for Computer Security*, Minneapolis, Minnesota, USA, Oct. 2005.

[4] J. A. Kupsch and B. P. Miller. Manual vs. Automated Vulnerability Assessment: A Case Study. In *First International Workshop on Managing Insider Security Threats (MIST)*, West Lafayette, Indiana, USA, June 2009.

[5] J. A. Kupsch, B. P. Miller, E. Heymann, and E. César. First Principles Vulnerability Assessment. In *ACM Cloud Computing Security Workshop (CCSW)*, Chicago, Illinois, USA, Oct. 2010.

[6] M. Litzkow, M. Livny, and M. Mutka. Condor — A Hunter of Idle Workstations. San Jose, California, USA, June 1988.

[7] Microsoft Corporation. *The Internet Protocol Helper (IP Helper) API.* `http://msdn.microsoft.com/en-us/library/windows/desktop/aa366073(v=vs.85).aspx`.

[8] Microsoft Corporation. *Threat Modeling Tool.* `http://www.microsoft.com/en-us/download/details.aspx?id=2955`.

[9] A. V. Mirgorodskiy, N. Maruyama, and B. P. Miller. Problem diagnosis in large-scale computing environments. In *Supercomputing 2006*, Tampa, Florida, USA, Nov. 2006.

[10] A. V. Mirgorodskiy and B. P. Miller. Autonomous analysis of interactive systems with self-propelled instrumentation. In *12th Multimedia Computing and Networking*, San Jose, California, USA, Jan. 2005.

[11] A. V. Mirgorodskiy and B. P. Miller. Diagnosing distributed systems with self-propelled instrumentation. In *ACM/IFIP/USENIX 9th International Middleware Conference*, Leuven, Belgium, Dec. 2008.

[12] MIST Project, University of Wisconsin and UAB. *Condor Vulnerability Reports.* `http://research.cs.wisc.edu/condor/security/vulnerabilities/`.

[13] Riverbed Technology. *WinPcap.* `http://www.winpcap.org/`.

[14] SRB Team, San Diego Supercomputer Center. *Storage Resource Broker.* `http://www.sdsc.edu/srb/index.php/Main_Page`.

[15] F. Swiderski and W. Snyder. *Threat Modeling.* Microsoft Press, First edition, 2004.

[16] D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing in Practice: the Condor Experience. *Concurrency — Practice and Experience*, 17(2-4):323–356, 2005.

[17] P. Trinius, T. Holz, J. Gobel, and F. Freiling. Visual analysis of malware behavior using treemaps and thread graphs. In *Symposium on Visualization for Cyber Security*, Atlantic City, New Jersey, USA, Oct. 2009.

[18] World Wide Web Consortium. *Scalable Vector Graphics (SVG) 1.1 (Second Edition).* `http://www.w3.org/TR/SVG11/`.

[19] Y. H. Xia, K. D. Fairbanks, and H. Owen. Visual Analysis of Program Flow Data with Data Propagation. In *IEEE Workshop on Visualization for Computer Security*, Cambridge, Massachusetts, USA, Sept. 2008.

[20] W. Yurcik, X. Meng, and N. Kiyanclar. Nvisioncc: a visualization framework for high performance cluster security. In *Workshop on Visualization and data mining for computer security*, Washington DC, USA, Oct. 2004.