A Graph-Theoretic Visualization Approach to Network Risk Analysis

Scott O'Hare[1], Steven Noel[2], and Kenneth Prole[1]

[1] Secure Decisions, Division of Applied Visions Inc., 6 Bayview Ave., Northport, NY, USA
[2] Center for Secure Information Systems, George Mason University, Fairfax, VA, USA
{ScottO,KennyP}@securedecisions.avi.com, snoel@gmu.edu

Abstract. This paper describes a software system that provides significant new capabilities for visualization and analysis of network attack graphs produced through Topological Vulnerability Analysis (TVA). The TVA approach draws on a database of known exploits and system vulnerabilities to provide a connected graph representing possible cyber-attack paths within a given network. Our visualization approach builds on the extensive functionality of the yWorks suite of graphing tools, providing customized new capabilities for importing, displaying, and interacting with large scale attack graphs, to facilitate comprehensive network security analysis. These visualization capabilities include clustering of attack graph elements for reducing visual complexity, a hierarchical dictionary of attack graph elements, high-level overview with detail drill-down, interactive on-graph hardening of attacker exploits, and interactive graph layouts. This new visualization system is an integrated component of the CAULDRON attack graph tool developed at George Mason University.

Keywords: network security, attack graph, exploit analysis, vulnerability assessment, visualization, situational awareness.

1 Introduction

Powerful analytic tools generally require well-designed user interfaces in order to be fully effective in their designated applications. This is especially true for tools designed to enhance network security: There are order-of-magnitude complexity issues associated with network topology maps, traffic data collection, and expert systems built around attack profile data and traffic correlation. For such analysis to be comprehensible, we need sophisticated visualization and interaction mechanisms.

We describe a visualization component for network attack graph analysis. This is an integrated component of the CAULDRON tool developed at the Center for Secure Information Systems at George Mason University. CAULDRON analyzes network topology and vulnerability data, combined with a comprehensive attack profile database. This Topological Vulnerability Analysis (TVA) [1] generates a complete *attack graph* showing all possible attack paths through a given network. The attack graph represents vulnerable network hosts and exploits that may be launched against them, with attack state transition data determined by the exploits. This attack graph, is still generally too large to be viewed and understood in its entirety.

Our approach to the visualization problem for TVA is based on tools explicitly designed for displaying and interactively analyzing graphs of interconnected nodes. A number of such tools are available, including Tom Sawyer, JGraph, Prefuse, Jviews Diagrammer, and yWorks. After a careful survey, we selected yWorks [2] for this application, based on its extensive feature set, deep and comprehensive Java API, and attractive deployment licensing terms.

The resulting software component provides powerful visualization capabilities for CAULDRON TVA attack graphs. A key feature is the implementation of hierarchical node and edge grouping along lines of *protection domains*. Protection domains are sets of machines with *unrestricted access* to one another's vulnerabilities, forming a completely connected sub-graph. Within each domain, it is sufficient to encode a particular host exploit only once, then implicitly, all hosts within the domain can carry out that exploit. Across domains, the exploits are all explicit. Thus our protection domain abstraction preserves all the information of the complete (ungrouped) graph, including intra-domain exploits. This abstraction reduces complexity within each protection domain from quadratic to linear, providing significant scalability, facilitating analyst navigation and cognition [3].

In our visualization system, a high-level view clearly displays exploit relationships among protection domains, which can be opened individually or in groups for deeper views of attack properties and relationships. A complete listing of active exploits and their associated details is available at all times relative to any selected component. Interactive hardening of on-graph nodes and exploits can be emulated to study the effects of remediation and "what-if" scenarios. Additionally, a suite of interactive layout tools, including manual repositioning of entities, along with full-scale layout algorithms, is continuously available to restructure or simply clean up the display.

The next section gives an overview of the CAULDRON tool, including the nature and utility of network attack graphs. In Section 3, we discuss yWorks architecture and capabilities, in relation to requirements for CAULDRON attack graph visualization. We also describe custom components that meet special requirements, as well as architectural features for performance and visual comprehension. Section 4 then describes the resulting visualization capabilities, including operational scenarios and ideas for future improvements.

2 CAULDRON Tool Capabilities

The CAULDRON TVA approach simulates incremental network penetration, showing all possible attack paths through a network. This simulation is based on a detailed model of the network configuration, attacker capabilities, and desired attack scenario. Because of the inherent interdependencies of vulnerability across a network, such a topological approach is necessary for a full understanding of attack risk.

CAULDRON captures configuration details for a network by processing the output of network scanning tools (Fig.1). It combines scans from various network locations, building a complete map of connectivity to vulnerable services throughout the network. It integrates with Nessus, FoundScan, and Retina, and Symantec Discovery. Integration with Altiris is currently under development.

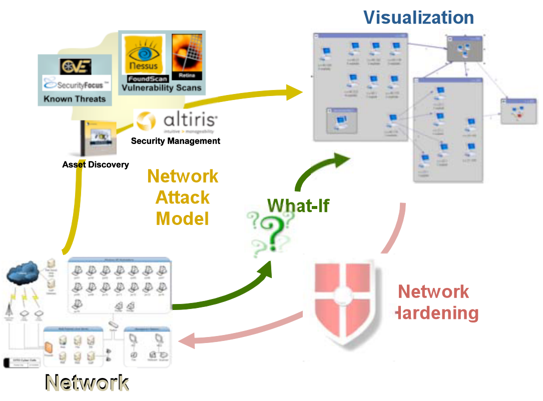


Fig. 1. CAULDRON architecture

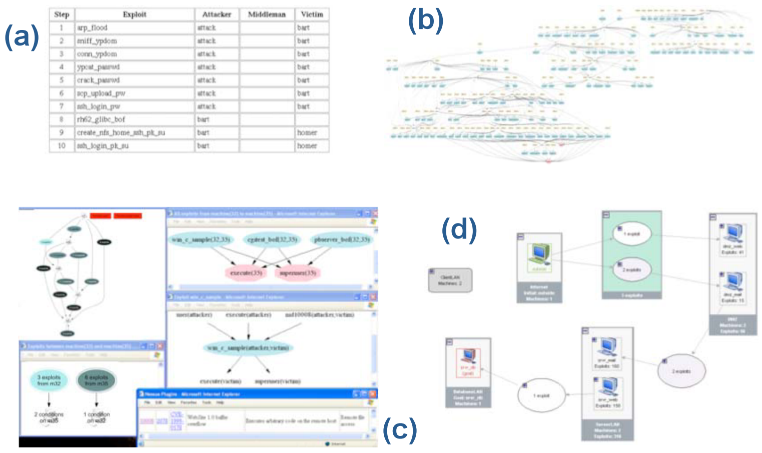
CAULDRON maintains a comprehensive database of modeled attacker exploits (currently over 20,000), based on software vulnerabilities reported in various sources, including Symantec DeepSight (a direct XML feed of Bugtraq along with other data), and MITRE's Common Vulnerabilities and Exposures (CVE). From the input model of network configuration and attacker exploits, CAULDRON computes a graph comprising all possible attack paths through the network. This graph is computed through simulated multi-step attacks according to a given user scenario.

The TVA approach as implemented by CAULDRON is not simply a cross-referencing of security data. Rather, it is a simulation of multi-step network penetration, with a full range of host vulnerability types and network configuration variations. For example, we have implemented exploit rules for buffer overflows, user logins, file transfers, port forwarding, traffic sniffing, spoofing attacks, client-side attacks, and denial-of-service.

Further, the ability to experiment through such what-if analyses is a powerful CAULDRON capability. The analyst can specify a starting point for the attack (the presumed threat source), as well as an attack goal (critical network asset to protect). The analyst can also model the effects of software patches or other mitigation solutions, which are included in the CAULDRON database. Once an attack graph has been computed, CAULDRON analyzes the results and provides recommendations for optimal network defenses [4].

3 Attack Graph Visualization

Capabilities for generating and visualizing TVA attack graphs have undergone significant evolution over time. TVA technology was originally limited to computing single attack paths, and the original presentation was a simple table, as in Fig.2(a). Later, the capability for efficient computation of all possible paths was developed, but visualization of the resulting graphs in their full detail is difficult to assimilate, e.g., Fig.2(b). Therefore primitive graph clustering techniques were developed, in special-purpose code, which was cumbersome and had limited interactive capabilities, as shown in Fig.2(c). Later, a more advanced visualization capability was developed



Step	Exploit	Attacker	Middleman	Victim
1	arp_flood	attack		bart
2	sniff_ypdom	attack		bart
3	conn_ypdom	attack		bart
4	ypcat_passwd	attack		bart
5	crack_passwd	attack		bart
6	scp_upload_pw	attack		bart
7	ssh_login_pw	attack		bart
8	rh62_glibc_bof	bart		
9	create_nfs_home_ssh_pk_su	bart		homer
10	ssh_login_pk_su	bart		homer

Fig. 2. Evolution of attack graph visualization capabilities

using Tom Sawyer, shown in Fig.2(d), although this exhibited performance problems for larger attack graphs. Our implementation using yFiles adds new analytic and display capabilities, while addressing these problems.

3.1 Loading the Attack Graph

The attack graph structure is delivered in an XML document conforming to a specific XML Schema Definition (XSD). We leverage Apache's XMLBeans technology, a Java-to-XML binding framework, to import the attack graph XML. Existing yFiles XML import capabilities require the existence of an Extensible Stylesheet Language Transformation (XSLT) into one of several standardized graph representations, which is not sufficiently general for our purposes. The XMLBeans component uses an XSD file to build a class library corresponding to the internal document structure. Utilities parse XML data structures and ensure conformance with the XSD, allowing us to acquire an attack graph as an organized collection of instantiated Java objects.

3.2 The Node Hierarchy

The attack graph is built within yWorks by transforming attack graph machine (host) objects into graph *nodes*, and exploit objects into graph *edges*. Several layers of graphical nesting are also performed. The most fundamental of these is based on protection domains, which are represented within the yFiles graph as *group* or *folder* nodes. (From a display or layout perspective, a group node is essentially a folder node that has been opened, and whose child elements are visible.) Machine nodes within a protection domain are represented as child nodes of the corresponding group or folder. We perform the layout of nodes within a group as the graph is assembled and initially present the graph in its *top-level* layout, in which only protection domains and the exploits connecting them are visible.

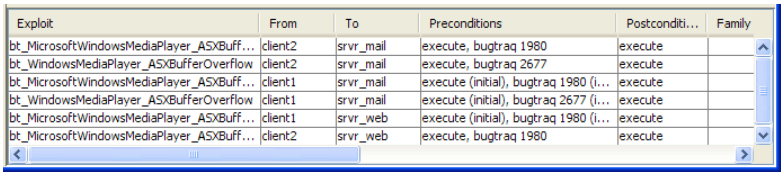
Presenting an initial top-level view yields an enormous improvement in the initial performance of the graph layout algorithm. Execution time of layout algorithms increases rapidly with the number of visible nodes and edges to be displayed. Earlier efforts to import large attack graphs were quite time-consuming, primarily due to this initial layout overhead. A top-level layout approach, combined with yFile’s ability to perform so-called *incremental* layout algorithms allow us to import and display large attack graphs in seconds that formerly had taken several hours to load.

3.3 The Edge Hierarchy

Another feature to enhance layout performance involves the manner in which exploits are represented as edges. Only exploits that connect machines in different protection domains appear as edges in our graph, that is, we suppress edge creation for *intra-domain* edges. This significantly accelerates initial graph setup, as well as subsequent layout steps resulting from interactive and redrawing operations. We also aggregate into a single edge any multiple edges connecting the same pair of nodes. These edge policies provide significant improvements in graph readability in addition to performance enhancements.

In the hierarchical navigation of nodes, no information is lost; one has merely to expand a folder node to acquire information hidden at a lower level. With the edge representation policies described above, it is not possible, in general, to recover a full edge set through simple expansion. Special mechanisms have been implemented to remedy this. Aggregated edges are labeled with an edge count, and edge line thickness also indicates the total number of exploits being represented. Additionally, the tool contains an *exploit table* that displays the full list of exploits, with complete attributes, associated with any single aggregated edge. Simply selecting an edge of the graph populates this table, as shown in Fig.3.

The exploit table also tracks node selection: all exploits associated with a given machine (node), or protection domain (folder or group node) are displayed in the exploit table whenever the node is selected. This includes the display of *intra-domain* exploits, even though these are not explicitly represented by edges. Thus the full set of information provided by an attack graph is always available, and can be viewed in an intuitive way within the user interface. The exploit table allows a “microscopic” analysis of exploit details, while fundamental topology and network relationships are kept simple and understandable within the graph view.



Exploit	From	To	Preconditions	Postconditi...	Family
bt_MicrosoftWindowsMediaPlayer_ASXBuff...	client2	srvr_mail	execute, bugtraq 1980	execute	
bt_WindowsMediaPlayer_ASXBufferOverflow	client2	srvr_mail	execute, bugtraq 2677	execute	
bt_MicrosoftWindowsMediaPlayer_ASXBuff...	client1	srvr_mail	execute (initial), bugtraq 1980 (i...	execute	
bt_WindowsMediaPlayer_ASXBufferOverflow	client1	srvr_mail	execute (initial), bugtraq 2677 (i...	execute	
bt_MicrosoftWindowsMediaPlayer_ASXBuff...	client1	srvr_web	execute (initial), bugtraq 1980 (i...	execute	
bt_MicrosoftWindowsMediaPlayer_ASXBuff...	client2	srvr_web	execute, bugtraq 1980	execute	

Fig. 3. Exploit table from the attack graph visualization tool

3.4 Additional Graph Visualization Features

Hardening: Viewing vulnerabilities or potential exploits within a network, the analyst is generally faced with multiple options for remediation. These options often involve choosing a machine or set of machines to protect (harden), or identifying specific exploits to protect against. We visually display the effects, in graphical terms, that occur when a specific node or protection domain is hardened or when a specific exploit is neutralized. This involves determining which elements are no longer vulnerable after the hardening, and removing these elements from the attack graph. These elements are placed into a separate list, which in effect quantifies the benefits to be obtained from the specific hardening operation.

Layout Algorithms: The yFiles engine incorporates an impressive architecture for implementing layout algorithms on a given attack graph. Many of these algorithms are the end product of significant mathematical and computational research. Having a rich palette of alternate layouts to choose from greatly strengthens the analytic benefits of graph visualization, since viewing data with different layout schemes can often enable recognition of fundamental underlying patterns that might otherwise be invisible. *Incremental* layout algorithms are intended to optimize the results of small operations, such as opening a folder or dragging a node, while *from-scratch* or *global* layout algorithms generally produce radical transformations of the entire set into an entirely new view. We permit the invocation at any time of hierarchical, organic, circular, or orthogonal layout algorithms.

Aggregation: The ability to apply additional levels of aggregation to an existing display can be useful to an analyst wishing to study larger-scale behavior or simplify an existing region of the graph. We allow selection of multiple entities and aggregation into a single folder. This requires incremental layout to be performed, and edge aggregation quantities to be re-computed.

4 Visualization Features

Fig.4 shows the components of our attack graph visualization tool. The *main graph* view is the attack graph showing all possible (directed) paths through the network, in which the analyst may drilldown, perform what-if analysis, etc. In the scenario shown, a particular attack starting point (green) and ending point (red) are specified. Two protection domains are expanded to show their member hosts and the exploits among them. The *exploit table* displays the relevant exploits (as both attackers and victims) for the selected protection domain. Mouse hovering over an *exploit field* shows the full data for that field. The *overview pane* maintains the context of the overall graph. The *tree view* represents the entire attack graph in the form of a directory hierarchy. The *harden list* logs interactive what-if network hardening decisions, while the *defense* shows optimal network hardening recommendations automatically computed by CAULDRON.

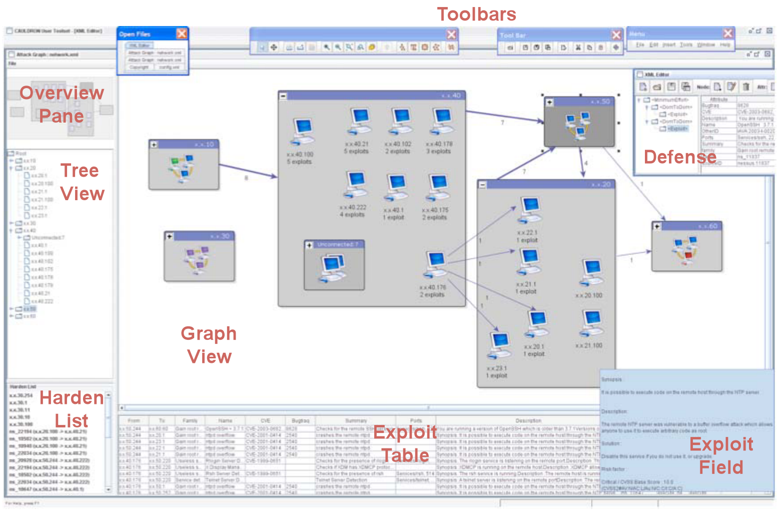


Fig. 4. Major components of the attack graph visualization tool

Graph interactions: The graph view supports a number of interactive edit mode features, including selection, deletion, relocation, and resizing of elements. Protection domains and other higher level nodes are opened and closed by clicking the +/- icon in the upper left corner. The graph can be zoomed to any magnification and positioned arbitrarily. The main graph view, tree view, and exploit table are all linked, so that user focus on any one component shifts focus on the others.

Context Menu: Context menu options are available by right-clicking an item or whitespace in the display, for network hardening simulation, deleting nodes, manipulating folders, etc. The context menu is also supports aggregating nodes into new folders. Another feature, called t*raversal*, initiates an animated trace (in red) of all exploits originating or terminating in a selected node, providing focus on specific attack scenarios within a complex attack graph display.

Toolbar Features: A number of other useful features are implemented in the application toolbar, such as buttons for invoking layout algorithms, and the interactive functions of edit mode. The export function exports the graph view in *.JPG, *.GIF, or *.SVG formats. The copy to clipboard transfers either the graph view or the entire graph to the clipboard. The magnifying glass zoom tool has arbitrary radius and power, and the cursor remains active at the center of the magnified view.

5 Related Work

Early work in automated generation of attack graphs involved explicit enumeration of attack states, which had serious scalability problems [5][6][7]. Under reasonable assumptions, complexity of attack graph generation was shown to be polynomial [8]. Attack graphs have also been generated efficiently through relational [9] and rule-based [10] approaches. Attack graph research has generally focused on efficiency, rather than visualization methods. The approach in [11] visualizes single-step attacks and reachability only. Attack graph visualization capabilities in commercial tools remain limited [12][13]. Our work is unique in that it is the first practical application of the attack graph visual clustering framework proposed in [3].

6 Summary

This paper describes a graph-theoretic approach to the problem of network risk visualization. This provides powerful new capabilities for visual analysis of attack graphs, and is an integrated component of the CAULDRON tool developed at George Mason University. This approach is exemplary in that it leverages the comprehensive yFiles architecture, bringing flexible new visualization and analysis capabilities to the network security realm. This provides essential building blocks for analyzing, visualizing, editing, and drawing network attack graphs, opening the door to a wide range of new analytic capabilities.

References

1. Jajodia, S., Noel, S.: Topological Vulnerability Analysis: A Powerful New Approach for Network Attack Prevention, Detection, and Response. Indian Statistical Institute Monograph Series. World Scientific Press, Singapore (2008)
2. yWorks – The Diagramming Company, `http://www.yworks.com/en/index.html`
3. Noel, S., Jajodia, S.: Managing Attack Graph Complexity through Visual Hierarchical Aggregation. In: Workshop on Visualization and Data Mining for Computer Security (2004)
4. Wang, L., Noel, S., Jajodia, S.: Minimum-Cost Network Hardening Using Attack Graphs. Computer Communications 29(18), 3812–3824 (2006)
5. Phillips, C., Swiler, L.: A Graph-Based System for Network-Vulnerability Analysis. In: New Security Paradigms Workshop (1998)
6. Ritchey, R., Ammann, P.: Using Model Checking to Analyze Network Vulnerabilities. In: IEEE Symposium on Security and Privacy (2000)
7. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.: Automated Generation and Analysis of Attack Graphs. In: Proceedings of the IEEE Symposium on Security and Privacy (2002)
8. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, Graph-Based Network Vulnerability Analysis. In: 9th ACM Conference on Computer and Communications Security (2002)
9. Wang, L., Yao, C., Singhal, A., Jajodia, S.: Interactive Analysis of Attack Graphs Using Relational Queries. In: Data and Applications Security XX (2006)
10. Ou, X., Boyer, W., McQueen, M.: A Scalable Approach to Attack Graph Generation. In: 13th ACM Conference on Computer and Communications Security (2006)
11. Williams, L., Lippmann, R., Ingols, K.: An Interactive Attack Graph Cascade and Reachability Display. In: Workshop on Visualization for Computer Security (2007)
12. Skybox Security, `http://www.skyboxsecurity.com/`
13. RedSeal Systems, `http://www.redseal.net/`