A HOST-BASED SECURITY ASSESSMENT ARCHITECTURE FOR INDUSTRIAL CONTROL SYSTEMS

Abhishek Rakshit
Department of Computing and Information Sciences
Kansas State University
abhirak@ksu.edu

Xinming Ou
Department of Computing and Information Sciences
Kansas State University
xou@ksu.edu

Abstract—**Computerized control systems perform vital functions across many critical infrastructures throughout the nation. These systems can be vulnerable to a variety of attacks leading to devastating consequences like loss of production, interruption in distribution of public utilities and most importantly endangering public safety. This calls for an approach to halt attacks in their tracks before being able to do any harm to these systems. Vulnerability assessment performed on these systems can identify and assess potential vulnerabilities in a control system network, before they are exploited by malicious intruders. An effective vulnerability assessment architecture should assimilate security knowledge from multiple sources to uncover all the vulnerabilities present on a host. Legitimate concerns arise since host-based security scanners typically need to run at administrative privileges, and takes input from external knowledge sources for the analysis making it imperative that the scanner be trustworthy. Intentionally or otherwise, ill-formed input may compromise the scanner and the whole system if the scanner is susceptible to, or carries one or more vulnerability itself. We have implemented the scanning architecture in the context of an enterprise-level security analyzer.The analyzer finds security vulnerabilities present on a host according to the third-party security knowledge specified in Open Vulnerability Assessment Language(OVAL). This paper presents an architecture where a host-based security scanner's code base can be minimized to an extent where its correctness can be verified by adequate vetting. Moreover, the architecture also allows for leveraging third-party security knowledge efficiently and supports various higher-level security analysis.**

I. Introduction

Most of the *Industrial Control* computer systems are high priority targets for cyber-attacks. One of the main reasons for this is their involvement in a country's critical infrastructures such as electrical, telephone, water, energy, *etc.* In addition, these systems are specific points of vulnerability for they were meant to be stand alone systems to begin with and hence, the security measures implemented were sub par. However, with the rise in interconnectivity among these infrastructure systems, we often find ourselves in a situation where millions of infrastructure components are connected to networks, allowing hackers to access the systems that were never designed to be exposed to cyber-attacks [1], [2].

This work was supported by the U.S. National Science Foundation under Grant No. CNS-0716665. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

With rapid growth in both the number and sophistication of cyber-attacks, it has become imperative that cyber defenders be equipped with highly effective tools that identify security vulnerabilities before they are exploited. It is not hard to perceive that successful attacks on control systems will have devastating consequences, for instance endangering public health and safety, damaging the environment, or causing a loss of production, generation, or distribution of public utilities [1]. This makes it crucial for control systems to be secured before they are attacked.

"Vulnerability assessment"(VA) is one such approach. VA analyzers are tools that scan a host system or an network to check for the presence of security vulnerabilities. The term host/target used throughout, refers to the system which is scanned for vulnerabilities. VA analyzers can be broadly classified into two categories; Network-based and Host-based Analyzer.

A network-based scanner(*e.g.* Nessus [3]) probes a machine remotely to find vulnerabilities. A host-based scanner on the other hand, is installed on the host system itself. The latter examines host data against security knowledge to assess the vulnerabilities in the respective systems. The term "security knowledge" used throughout refers to the definitions, which determine the conditions for a known vulnerability to be true(or false) on a host(or target) system. The scanner gathers the host's configuration information and performs various analysis based on the received security knowledge.

Being installed on the hosts, host-based scanners can directly access the configuration information and various services running on the host. However, conventional host-based scanners require regular installation/updates for the agents on every machine of an enterprise network. This becomes a rather daunting task as the network scales up. Moreover, as the clients run on the host machine utilizing its resources, careful consideration has to done while configuring these clients to avoid loss of data or crashing the host.

As new vulnerabilities are uncovered at a frequent rate and maintained at multiple knowledge-bases (e.g NVD [4], OSVD [5], CVE [6]), a comprehensive security analysis requires knowledge from all those sources. But since this knowledge is being provided by a third party and is consumed on the end hosts, any vulnerability in the agent could render the end host vulnerable. This acts as an hindrance to the approach of using security knowledge from multiple third party knowledge for security analysis. Specifically, the system administrators are skeptical to trust the agent code (with thousands of lines of code), not to harm the host itself. Moreover, consumption of knowledge on each end host introduces a great amount of replicated effort and makes it hard to combine knowledge from various sources and conduct a global security analysis at the enterprise level. Thus, we need an architecture which enables incorporation of security knowledge from multiple knowledge-bases and uses it efficiently to provide a comprehensive vulnerability analysis for the end host without interfering its normal functioning.

In this paper, we propose an architecture for host-based security analysis, which not only addresses the above stated concerns but also supports other high level security analysis tools. We accomplish this by separating the process of gathering host configuration information from the analysis of the configuration information. The proposed system is designed as a bi-component architecture for host-based vulnerability analysis. The first component is the scanner (or the agent as we often call it) that needs to be installed on the host. The *Agent* serves the sole purpose of gathering information from the host and does not perform any analysis. The second component is the *Analyzer*, which resides on a server or a cluster of servers depending on the size of the network. The sole responsibility of the analyzer is to perform security analysis on the information. The analyzer produces a comprehensive security report for every host on the network, leveraging third-party security knowledge efficiently. This comprehensive report can become the basis for network administrator's decisions to safeguard the host (and the network) against reported vulnerabilities and be the input to a higher-level global security analysis tool.

The proposed architecture has the following advantages:

- The architecture efficiently leverages third party security knowledge at the same time addressing the concern of replication of knowledge by eliminating the need to update each agent as security knowledge updates are performed at the analyzer at a centralized location.
- Significant reduction in the size of the code-base on the end host makes the installation and configuration of the agent easy and less likely to disrupt active services. In addition, small code-base reduces the likelihood of introduction of programming errors/bugs.
- The small code base makes it possible for the administrator (or some other trusted party) to check for flaws, malicious content, making it a good candidate for static code analysis as well as making it amenable to manual inspection, restoring the trust in the code.
- This architecture supports other high level security analysis on the data collected from all the hosts on the network (e.g MulVAL [7]).

Our research is conducted within the context of the MulVAL (Multi-host, multi-stage Vulnerability Analysis) [7] project. MulVAL is an enterprise-level security analyzer that can automatically compute all possible multi-step, multi-host attack paths in an enterprise network-based on security vulnerabilities discovered on end hosts. The input to MulVAL is the result of host-based vulnerability assessment performed on each and every managed machine in the enterprise network. The original MulVAL work used an adapted OVAL (Open Vulnerability Assessment Language) [8] interpreter released by the MITRE corporation to analyze each end host. The external security knowledge is specified in OVAL language and needs to be sent to all the end hosts. When we tried to deploy the MulVAL tool on some enterprise networks, the first concern from the system administrators was always the trustworthiness of the adapted OVAL interpreter. This motivated us to develop a vulnerability assessment architecture, where the scanner that needs to be run on the host has minimal code base, and thus its correctness can be verified through adequate *Code Vetting*. At the same time, allowing efficient leveraging of third-party security knowledge by providing the option of maintaining a compilation of security knowledge from multiple sources.

II. Centralized Host Based Architecture

As the name suggests, the architecture takes a centralized approach when analyzing a host with regards to the security definitions from the knowledge base. Security definitions are different from virus definitions used in virus scanners. Virus signatures typically refer to file name and contents, whereas conditions to determine the existence of a security vulnerability (*i.e.* security definitions) refer to configuration parameters with arbitrary logical relationships. For example, the OVAL [8] language for the Windows platform could be used to specify conditions on any Windows registry entry, on any file's attributes or on any process running on a machine. It also has a number of logical connectors and attributes which can specify the full propositional relations as well as limited first-order logic semantics. Moreover, a conventional host-based security scanner often has a large code-base, due to the need to support various kinds of checking and analysis tasks. For example, the code size of MITRE's reference implementation of an OVAL scanner [9] developed in C++ programming language has around 35,000 lines of code. One can imagine than an application of this size makes it very hard for the developers to keep it totally flawless, and it is hard if not impossible to verify that there is no security vulnerabilities in the same.

Figure 1 shows an architecture consisting of a central analyzer, a configuration inventory and a highly stripped-down agent on the target host. The configuration inventory holds the configuration information obtained from the hosts. The architecture employs an *agent* which performs basic functionalities with regards to the configuration gathering, such as dumping the whole Windows registry, querying for a file's attribute and getting a process's status. Most of these functionalities have readily available system commands or application program-

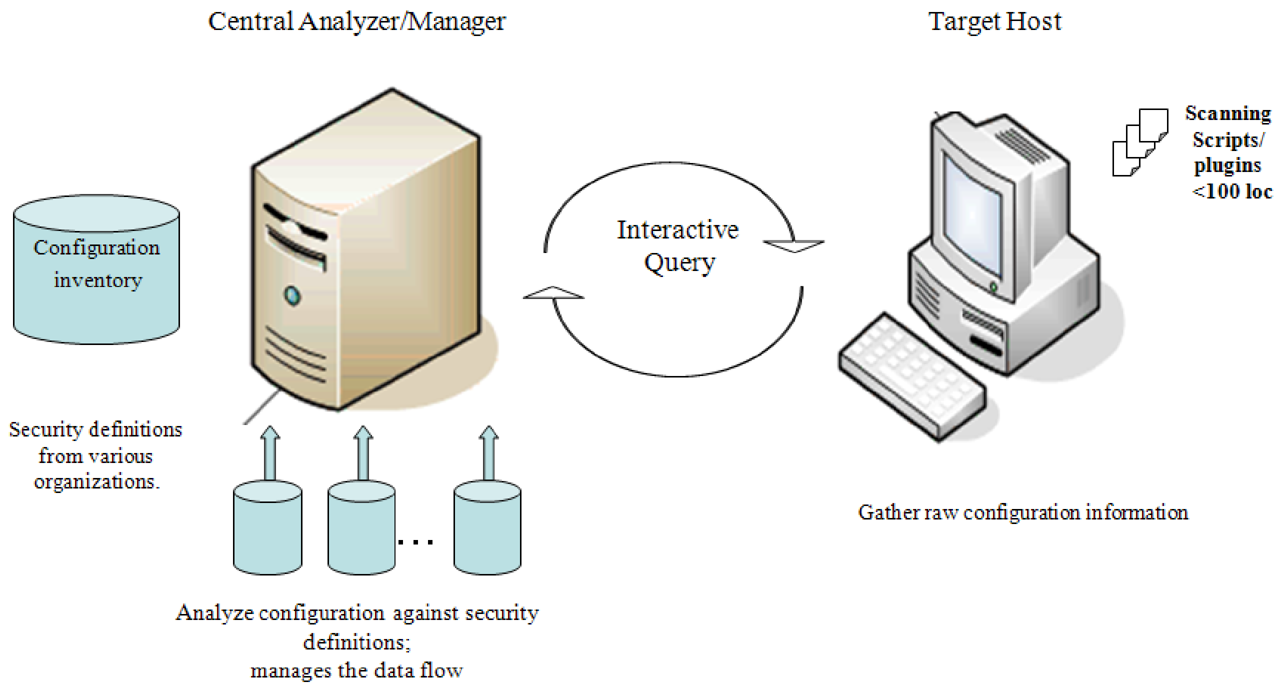
Central Analyzer/Manager
Target Host
Configuration inventory
Interactive Query
Scanning Scripts/ plugins <100 loc
Security definitions from various organizations.
Gather raw configuration information
Analyze configuration against security definitions; manages the data flow

Fig. 1. Proposed Architecture

ming interfaces (APIs) and thus the amount of code needed to accomplish this is minimal. The agent scripts are initiated by the server according to the data required for the analysis through the interactive query. The agent is less than a hundred lines of code thereby significantly reducing the possibilities of programming flaws/bugs in the code. Moreover, the small code base can be rigorously vetted for any kind of flaws therefore increasing the trustworthiness of the agent. These light weight scripts also overcome the problem of resource consumption on the host by adding no significant burden on the hosts resources. Both the properties of minimalized resource consumption on the host and possibility of rigorous code vetting are critical properties in context to control systems, as it is vital that there be no interruptions in their normal functioning. Finally, the agent reports end-host configuration information to the central analyzer for the analysis.

The *Analyzer* is the most important part of the architecture. Its functionalities include scheduling the scan for all the target hosts on the network, managing the configuration inventory and storing all the information obtained from the hosts in the inventory. The analyzer gathers, converts and stores security knowledge from multiple knowledge sources in Datalog format acceptable to the logic-based comparator. It then initiates the logic-based comparator by providing it with both the configuration information of the target (received from the agent) as well as the compiled security knowledge to get the results for the respective host.

A significant advantage of this architecture is that the central analyzer has a complete view of the configuration of every managed host in the network, and can conduct various high level security analysis on the configuration information. In this architecture we can look across multiple hosts and see a chain or work-flow that is vulnerable, not just an individual host/process. This is how the architecture supports high level security analysis tools which can notice that two items that are relatively safe (even if vulnerable), are highly vulnerable when put together.

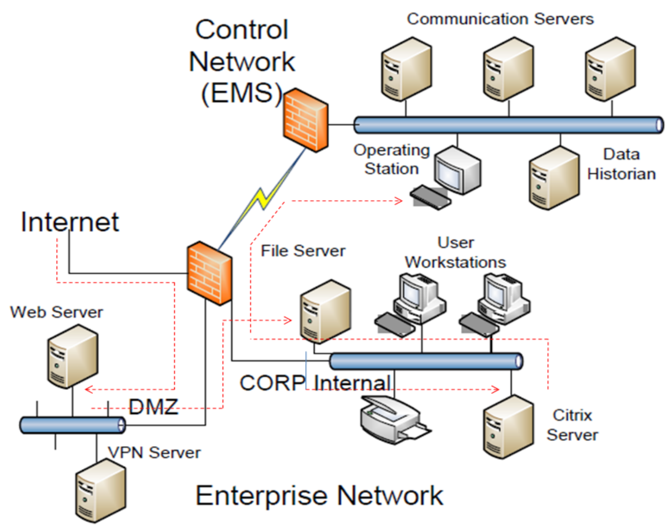

Fig. 2. Control System Network

Let us take an example [10] shown in Figure 2 which depicts an example based on a real (and much bigger) control system. The network includes three subnets: a DMZ (Demilitarized Zone), an internal subnet, and an EMS (Energy Management System) subnet, which is a control-system network for power grids. Both the web server and the VPN server are directly accessible from the Internet. The web server can access the file server through the NFS file-sharing protocol. Access to the EMS subnet is only allowed from the Citrix server in the internal subnet. An attack scenario where we assume that the attacker's goal is to gain privileges to execute code on the communication servers can be shown as follows. An attacker first compromises webServer by remotely exploiting a vulnerability to get local access on the webServer. The webServer can access the file server through vulnerable NFS service deamons. Once the attacker is in the internal subnet he can easily reach the Citrix server and hence the communication servers. From the communication servers, an attacker could send commands to physical facilities such as power-generating turbines, which can cause grave damage to critical infrastructures. Thus, we see that just finding vulnerabilities on separate hosts doesn't provide us with a whole picture and advanced security analysis tools like MulVAL are essential for the safeguard of control system networks.

Moreover, since all hosts' configuration is centrally stored at one place, the application of security knowledge is more efficient. This is due to the fact that the processing of the raw security knowledge from multiple sources is performed once and is used for all the machines on the network. This cost of pre-processing of the knowledge is amortized over all the machines under scan. In our implementation, we compile the knowledge into executable code, so the compilation is done only once and the code is directly applied to all the hosts' configuration data to perform the analysis.

The proposed architecture suits especially well to the recent trend of sharing security knowledge in an open and standard format. Apart from aiding to a more wider and comprehensive analysis for security vulnerabilities, our architecture also supports diverse research and development in the area of Information Security. Open Vulnerability Assessment Language (OVAL) is one such effort to enable a "community approach" to security management of enterprise systems. The effort has echoed well in the IT security management industry, with vendors like GFI LANguard [11] and SofCheck [12] which already offer "OVAL-compatible" products. This is a significant departure from the conventional business model where the vendors of vulnerability assessment tools provide security definitions in their own proprietary format. With the rapid growth in cyber security threats, it is evident that no single organization can provide a holistic solution to all security problems faced by the enterprise network systems. The ability to share security knowledge efficiently is the key to win the "Cyber War" against the Internet miscreants.

The security-scanning architecture presented in this paper facilitates knowledge sharing since it separates the two distinct phases in security analysis, configuration information gathering and vulnerability analysis. If the updated knowledge-base or analysis tools need the same set of configuration information from the end hosts, there is no need to update the agent or run new scans at all. This separated architecture avoids re-inventing the wheels in security analysis, facilitates knowledge sharing, and thus, maximize the benefit from all efforts involved in security management. The fact that our architecture holds configuration inventory also opens up avenues for its diverse usage in fields like "Enterprise Inventory Management" and "Troubleshooting" [13].

III. Implementation

Let us first be familiar with the security knowledge base used as an example in our discssion.

A. The OVAL Language

The Open Vulnerability and Assessment Language (OVAL) [8] is an international information security community standard to promote open and publicly available security content, and to standardize the transfer of this information across the entire spectrum of security tools and services. OVAL includes a language used to encode system details, and an assortment of content repositories held throughout the community. The language standardizes the three main steps of the assessment process: representing configuration information of systems for testing; analyzing the system for the presence of the specified machine state (vulnerability, configuration, patch state, *etc.*); and reporting the results of this assessment. It is an XML-based language and specifies vulnerable machine configuration for almost all types of platforms such as Windows, Linux, HP-UX, Cisco IOS and Sun Solaris. The definitions provided by OVAL stipulate the conditions that, when satisfied by the host, confirm the presence (or absence) of vulnerabilities on the same[1].

B. Architecture

As described in Section II the architecture is branched into data collection and analysis parts. The data collection is performed by the host resident agent and the analysis by the analyzer described hereon.

1) Configuration data collection: The agent in our implementation consists of a set of shell and Visual Basic scripts which are less than a hundred lines of code and reside on the host machine. It has a dedicated functionality of gathering information from the target/host. The information needed by the OVAL definitions is gathered from the registry dump, and the version numbers for specific files obtained by checking the file's attributes. The residing agent collects registry entry information and file version numbers from the attributes using Windows system commands and forwards it to the analyzer.

As described in this architecture all the system configurations and other information reside on the central analyzer. This could provide an opportunity to keep a check on what changes have been made on the machine from the time of the previous scan. This information is valuable for security forensics, as well as can be used by system administrators to track down configuration changes on machines for troubleshooting. The architecture supports convenient addition of scripts to the agent when needed, to get more information with respect to network,

[1]The complete documentation of the OVAL language can be found at http://oval.mitre.org/

port and file system status without incurring a lot of addition to its code weight. This in turn opens up new avenues for contributors to participate in the development of the agent, to gather knowledge specific to other applications such as inventory tracking and troubleshooting [13].

2) The Analyzer: The Analyzer is a centralized server connected to all the hosts on the network. It is responsible for performing a comprehensive security analysis of the hosts and reporting all the vulnerabilities present on them. The analyzer accomplishes this task by analyzing the information received from the agents (residing on the host) against the preprocessed security knowledge obtained from the information gathered from various security knowledge bases. The analyzer is further divided into two parts: *knowledge convertor* and *logical comparator*.

a) Knowledge Convertor: The knowledge convertor converts the updated security definitions (XML format) and raw host configuration (ascii format) into Datalog format. The datalog format of host configuration information taken as input by the comparator is shown below.

```
win_Reg_Entry(Path,Name,Value)
```

In the example shown above, "Path" is the *hive* and *key* values from a registry entry of the host machine. The "Path+Name"/ "Value" pair is matched against the specified data in the security definitions by the comparator. The value part is either a string or a numerical value. The convertor's next task is to process security knowledge and convert it into Datalog clauses. The OVAL repository is updated regularly and our architecture is well suited to adapt the constantly changing knowledge. Once we have both the host information and the security definitions in the required format, these are provided to the logical comparator for analysis.

b) Logical Comparator: The logical comparator accepts both the processed host information and the security knowledge and compares them using the XSB [14] engine, which is a Logic Programming and Deductive Database system. The comparator analyzes the information from the host with respect to the criteria from the security definitions to report all the vulnerabilities present on that host.

The reason for using a logic-based approach is that Datalog is both a declarative specification and an executable program. Thus, the security knowledge can be compiled into executables, during which significant optimization can be done to speed up the analysis process. This optimization cost is only paid once in converting the knowledge into Datalog bytecode, and can be amortized over the repeated application of the knowledge to a large number of machines over a period of time. As mentioned in section I our work is the underlying research for the MulVAL [7] project, which is a Datalog-based framework for modeling the interaction of software bugs with system and network configurations. It is thus a natural choice for us to use the same logical language for individual host's vulnerability assessment. Moreover, the saving gained through amortizing knowledge preprocessing cost (compilation in this case) applies to other internal knowledge representation as well.

IV. Results

The primary goal of our work is to investigate the effectiveness of central host-based architecture described in section III when compared with the MITRE's reference scanner [9]. The results indicate that the centralized scanner is not only effective in vulnerability assessment but also is more efficient with respect to time taken to perform comprehensive analysis when compared with the reference scanner.

The test bed for our experiments consists a client-server network with of host/client machines having *Windows XP* operating system and *Windows 2000 Server* operating system and the server is a dedicated *Linux* machine. The hosts were left un-patched with vulnerabilities present for testing purposes. Table I shows the system architecture of the machines on the test network with their processor, memory and operating system information.

TABLE I
Test Bed

Architecture	Analyzer	Host
Operating System	Linux	Win_XP/Win_2000
Processor	Dual Opteron 2.2ghz(x3)	2.2ghz AMD Opteron
Memory(GB)	16	2

During the analysis, the files transmitted to the analyzer from the agent residing on the host correspond to registry information and version numbers of the files specified in the security definitions for the host. The registry files for Windows XP machines were 38MB and for Windows 2000 Server machines were 26MB in a compressed format. File attribute data for both the platforms were less than 10KB.

Vulnerability detection capabilities of the central host-based analyzer are at par with MITRE's reference implementation of OVAL scanner. Both the scanners reported 260 vulnerabilities for the host with Windows 2000 Server operating system. For hosts with Windows XP operating system, the reference scanner reported 223 vulnerabilities. The centralized scanner has reported 224 vulnerabilities, including one vulnerability that was missed by the reference scanner but manually verified to exist on the host.

The centralized scanner came better when compared against the time taken to analyze a host machine by the reference OVAL scanner. Table II shows average time (in minutes) to complete the vulnerability analysis of the hosts with Windows 2000 Server and Windows XP operating system. A point worth mentioning here is that in the case of the reference scanner, the time reported reflects the time it executed on the host machine engaging the host's resources. On the contrary, in our scanner, only a fraction of the total analysis time shown is the time when resources on the host were used to get its configuration information. Rest of the time reflects the analysis part which is performed by our dedicated analyzer.

The security definitions are updated regularly by the OVAL community. We pre-compile and store the converted OVAL XSB bytecode whenever a new OVAL definition file is released. The compilation (with optimization) for an OVAL Windows definition takes 102 seconds and this time is not

TABLE II
ANALYSIS TIME COMPARISON(MINUTES)

Operating System	Reference scanner	Centralized Scanner
Win_XP	3:11	2:48
Win_2000	3:16	2:22

included in the above data. This overhead is a one-time investment and the cost can be amortized when the system is running on a large network because the XSB bytecode will be used for all the machines on the network. In addition one may also incrementally compile the newly added OVAL definitions, which can further reduce the knowledge pre-processing time.

V. DISCUSSION AND FUTURE WORK

The architecture proposed in this paper supports advanced security analysis such as attack-graph tools (*e.g.* MulVAL) [15], [16], [17]. Attack graph tools are high-level enterprise network security analyzers which take the baseline vulnerability information provided by the VA scanners as input. To determine the impact of the discovered vulnerabilities, the tools are equipped with high-level knowledge on reasoning about security interactions in an enterprise network. Application of this knowledge often needs additional configuration information beyond what the VA scanner can provide. Our architecture can easily accommodate these needs since the resident scripts can collect desired configuration information from the host.

Further, the centralized architecture of our system assures access to all the hosts on the network from the server. Thus, the server is also capable of hosting a network-based scanner to harness the advantages of the same. This would complement the host-based scanner and provide a extensive security analysis for the enterprise network.

The proposed scanning architecture can also be applied beyond security applications. A likely candidate is "Inventory Management". The agents provide all the configuration information which is stored on the centralized server and this information can be used for inventory management purposes to keep track of all the resources on the hosts throughout the enterprise network.

Trouble-shooting configuration problems in a network is another application which can be applied to the proposed architecture. Trouble-shooting often needs collecting a range of configuration parameters and sending them to a centralized place for analysis [13]. Since the agent on the host has a small code base, it is easier to guarantee that it will not further disrupt service while attempting to fix an existing problem. Moreover, configuration changes on a particular host can be tracked as configuration information before and after the problem occurred is available in the configuration inventory. The changes made in configuration settings is a logical starting point when troubleshooting the respective host.

Thus, the proposed architecture provides a novel approach to host-based vulnerability analysis along with a lot of potential for further research and development in many other fields including advanced security analysis.

VI. CONCLUSION

Security threats and breaches in a control system network of critical infrastructure can cause serious disruption in important processes and lead to grave consequences. A potent security system is imperative for such networks and vulnerability assessment is an important element for the same. The current security scenario demands an approach which focus not only on being able to assimilate data from multiple knowledge sources but also become the foundation for advanced security analysis.

The centralized host-based security scanning architecture proposed in the paper is one such approach. It supports knowledge assimilation from various sources providing a comprehensive security analysis. The centralized architecture overcomes the issue of knowledge replication and eradicates the need to regularly update the client due to its separation of analysis from the data collection part and further performing all the analysis on the centralized server. The agent residing on the host uses minimal resources and is simple enough to install and maintain. The reduced code base of the agent makes it less susceptible to programming flaws and also allows rigorous code vetting to gain the administrators' confidence. We empirically show that the centralized vulnerability analyzer to be at par with MITRE's reference scanner in terms of vulnerability detection and time taken to do the assessment.

REFERENCES

[1] R. F. Dacey, "Challenges in securing control systems," U.S. Government Accountability Office GAO, Tech. Rep., October 2003.
[2] A. Turner, "U.S. critical infrastructure in serious jeopardy," July 2007. [Online]. Available: http://www.csoonline.com
[3] "Nessus," http://www.nessus.org/nessus.
[4] "National Vulnerability Database," http://nvd.nist.gov/.
[5] "The Open Source Vulnerability Database," http://osvdb.org/.
[6] "Common Vulnerabilities and Exposures," http://cve.mitre.org/.
[7] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: A logic-based network security analyzer," in *14th USENIX Security Symposium*, Baltimore, Aug 2005, pp. 113–128.
[8] "OVAL (Open Vulnerabilities and Assessment Language)," http://oval.mitre.org/.
[9] "OVAL Interpreter," http://oval.mitre.org/language/download/interpreter/index.html.
[10] J. Homer, A. Varikuti, X. Ou, and M. McQueen., "Improving attack graph visualization through data reduction and attack grouping," in *5th International Workshop on Visualization for Cyber Security VizSec'08*, Sep 2008.
[11] "GFILANguard," http://www.gfi.com/lannetscan/.
[12] "SofCheck," http://www.sofcheck.com/.
[13] H. Huang, R. Jennings, Y. Ruan, R. Sahoo, S. Sahu, and A. Shaikh, "PDA: A tool for automated problem determination," in *21st Large Installation System Administration Conference, (LISA)*, Dallas, Nov 2007, pp. 153–166.
[14] "XSB: A Logic Programming and Deductive Database system," http://xsb.sourceforge.net/.
[15] S. Jajodia, S. Noel, and B. O'Berry, "Topological analysis of network attack vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challanges*, V. Kumar, J. Srivastava, and A. Lazarevic, Eds. Kluwer Academic Publisher, 2003, ch. 5.
[16] R. Lippmann, K. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, and R. Cunningham, "Evaluating and strengthening enterprise network security using attack graphs," MIT Lincoln Laboratory, Tech. Rep. ESC-TR-2005-064, October 2005.
[17] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *13th ACM Conference on Computer and Communications Security, (CCS)*, 2006, pp. 336–345.