Federal Voting Assistance Program (FVAP) Technology Projects

FVAP.GOV
FEDERAL VOTING ASSISTANCE PROGRAM

Software Assurance Final Report

Prepared For:
Federal Voting Assistance Program
Department of Defense
4800 Mark Center Drive
Mailbox 10
Alexandria, VA 22350-5000

Prepared By:
CALIBRE
6354 Walker Lane, Suite 300
Metro Park
Alexandria, Virginia 22310-3252
www.calibresys.com

Contract No.:
GS-35F-5833H

[redacted]

INVESTIGATION OF THE USE OF SOFTWARE ASSURANCE TOOLS ON INTERNET VOTING SOFTWARE APPLICATIONS

CONTRACT # GS-35F-5833H

Task # 2.5.5

Final Report

16 May 2014

[redacted]

Executive Summary

This report documents that existing software assurance (SA) tools provide a viable means of identifying potential security and coding best practice weaknesses of existing internet voting system vendors' software. The report documents a successful, and verified, methodology for conducting SA tool testing of voting system vendor software. The report also identifies challenges encountered and identifies resolutions that led to successful testing.

Under the Uniformed and Overseas Citizens Absentee Voting Act (UOCAVA) of 1986, the Federal Voting Assistance Program (FVAP) assists active duty uniformed service members, their families, and United States citizens residing outside the United States (U.S.) in exercising their right to vote by absentee ballot when they are away from their permanent address. In accordance with the 2002 and 2005 National Defense Authorization Acts (NDAAs) and the 2009 Military and Overseas Voters Empowerment (MOVE) Act, FVAP is investigating online voting support tools that might assist UOCAVA voters to securely and accurately cast their votes in a timely fashion. As part of this series of studies and analyses, FVAP is assessing supporting information technology (IT) and system security infrastructures, and the specific benefits of software assurance tools to document weaknesses in coding practices and overall election software security.

This report presents a testing methodology and accompanying analysis on the viability and effectiveness of five static analysis tools and two dynamic analysis tools in identifying weaknesses in coding practices and security of internet voting system vendor software. The tools and testing process examined system integrity and identified potential defects and weaknesses associated with election software source code from the three Election Assistance Commission (EAC)-registered internet voting system vendors. Additionally, this effort was intended to test the hypothesis that the use of suites of tools (as opposed to an individual tool) results in greater software security and reliability by significantly increasing the detection rate of actual coding weaknesses and defects (True Positives). A secondary goal was to examine the potential for optimizing SA tools through rules-based filtering of False Positives,[i] resulting in reduced time and effort reviewers and testers spend on labor intensive manual code reviews.

While this effort was not intended to assess, nor draw any opinions or conclusions on the security posture of the voting systems, all vendors were identified to possess defects in their source code.

[i] A False Positive is a result that is reported/identified as a defect when in fact it is not.

This report describes:

- The project's background, including the development, and subsequent modification of the toolbox of analysis tools;
- The protocol and methodology used to test the vendor software by the analysis tools and a discussion of tool optimization;
- The results of the testing conducted against the three vendors' source code, by tool, and an analysis of the results; and
- The validation of the protocol, methodology, and results by Pro V&V, a National Institute of Standards and Technology (NIST)-certified, EAC-registered test laboratory.

The following are conclusions based on the testing effort and the analysis of the test results for this project:

- If using commercial SA tools, it is imperative to understand the programming language(s) and other technical details utilized by the voting system vendors prior to tool acquisition.
- Using multiple static code analysis tools increased the number of potential defects identified in the source code for all severity ratings (*High, Medium, and Low*).
- However, if one considers the use of HP Fortify as the primary static analysis tool, the additional tools utilized for this analysis did not increase the number of True Positive *High* severity defects identified.
- For the C# and Java coding languages, HP Fortify identified the vast majority of potential defects. The open source tools used were of marginal value.
- Of the tools that were utilized for this analysis, the commercial tools (HP Fortify and Coverity) provided varying levels of customization/optimization that the open source tools often did not.
- Customizing static analysis tools to reduce/eliminate False Positives can be done in the development phase of coding (in the Integrated Development Environment or IDE); however when the tools are used for the current type of analysis, post development, all defects must be examined to determine a True/False Positive finding.

Having proven the viability and utility of SA tools in testing voting system vendor software, CALIBRE presents recommendations for either additional research with regard to the use of software assurance tools, other areas of research and analysis that may be useful regarding testing tools, and/or additional security analysis of voting system vendors. Among the recommendations are:

- Examine the use of multiple open source tools to provide the same level of analysis as a single commercial tool.
- Compare and contrast dynamic scanners on operational systems with known defects and scan with dynamic toolkit.

- Conduct a cost/benefit analysis of the use of automated tools versus manual code review in the testing and certification process.
- Analyze incorporating the requirement for a toolkit of assurance tools as part of the voting system testing and certification process in lieu of a complete line-by-line review of code.
- Develop an overall, comprehensive toolkit balanced with commercial and open source static and dynamic tools that can meet a wide variety of operational environments and software coding languages.

Table of Contents

1 Introduction

Under the Uniformed and Overseas Citizens Absentee Voting Act (UOCAVA) of 1986, the Federal Voting Assistance Program (FVAP) assists active duty uniformed service members, their families, and United States citizens residing outside the United States (U.S.) in exercising their right to vote by absentee ballot when they are away from their permanent address. In accordance with the 2002 and 2005 National Defense Authorization Acts (NDAAs) and the 2009 Military and Overseas Voters Empowerment (MOVE) Act, FVAP is investigating online voting support tools that might assist UOCAVA voters to securely and accurately cast their votes in a timely fashion. As part of this series of studies and analyses, FVAP is assessing supporting information technology (IT) and system security infrastructures, and the specific benefits of software assurance (SA) tools to document weaknesses in coding practices and overall election software security.

1.1 Software Assurance

Throughout the software development lifecycle (SDLC), SA tools identify defects, malicious code, and/or other flaws that could bring harm to the end user, which, in the case of online voting systems could influence election results. Currently, a multitude of commercial and open source SA tools are available in the marketplace,[2] and no single tool or application covers the entire gamut of software assurance and defect assessment. In 2012, under Phase 1 of this contract, CALIBRE conducted industry and market research to narrow the field of all available SA tools to those most likely to meet the specific needs of federal entities, including FVAP, the U.S. Election Assistance Commission (EAC), and the National Institute of Standards and Technology (NIST). Additionally, SA tools were selected based on their ability to assess the three EAC-registered internet voting system vendors. This previous effort resulted in a subset of 23 tools, from which tailored SA toolkits were developed based on each voting system software under examination.[3] The purpose of the research presented in this report was to evaluate these previously recommended SA tools to determine their effectiveness in identifying and mitigating potential defects in the context of internet voting systems' architectures deployed on the internet.

1.2 Purpose of this Report

The primary goal of this report is to provide a testing methodology and accompanying analysis on the viability and effectiveness of SA tools in documenting weaknesses in coding practices and security of internet voting system vendor software. This testing process examines system integrity and identifies potential weaknesses and resulting defects associated with election

[2] NIST SAMATE: National Institute for Standards and Technology Software Assurance Metrics and Tool Evaluation. http://samate.nist.gov/Main_Page.html

[3] FVAP. Assessment of Software Assurance Tools for Improving the Security of Voting Systems, 16 December 2012.

software source code from the three EAC-registered internet voting system vendors. Additionally, this effort was intended to test the hypothesis that the use of suites of tools (as opposed to an individual tool) results in greater software security and reliability by significantly increasing the detection rate of actual coding weaknesses and defects (True Positives). The outcome of greater True Positive detection is increased confidence in the trustworthiness and predictable execution of the application. A secondary goal is to examine the potential for optimizing SA tools through rules-based filtering of False Positives,[4] resulting in reduced time and effort reviewers and testers spend on labor intensive manual code reviews.

Additionally, FVAP recognized that an evaluation of SA tools could be helpful to entities beyond voting system manufacturers. For example, the EAC currently conducts an extensive manual source code review as part of its certification procedures. It is likely that the use of a suite of automated or semi-automated SA tools could create a more streamlined and cost-saving certification process by allocating resources more efficiently and effectively while at the same time improving effectiveness of the certification process for identifying true defects and weaknesses. The EAC can benefit from the testing methodology documentation CALIBRE created, which could serve as the baseline for a standardization process that uses SA tools in the voting systems certification process.

The testing and validation of SA tool suites presented in this report provides FVAP with a methodology for evaluating the quality of internet voting solutions during future implementations of pilot programs and in the execution of the congressionally mandated voting demonstration project. The intent of the testing was not to assess the weaknesses and defects of specific voting systems, nor was it to provide a comprehensive security overview of such systems. Rather, this project was designed to evaluate the usefulness of the specific SA tools in the current web-based environments of the EAC-registered internet voting system vendors. As such, the source code used for this testing came from systems currently available in the marketplace. While the results of this methodology are not intended to provide a comprehensive report on voting system SA, the findings are actionable and vendors could benefit from the remediation of identified defects.

The SA tools used to test voting systems come at a great expense, making it generally unlikely that the vendors will be afforded the identification of weaknesses these scans provide outside of this project. If a testing program were made available as part of the EAC certification process, vendors may again have access to this valuable information that could have a huge impact on future elections – preventing system failures or contamination of voting data. If funding could be pooled from a combination of federal, state, and vendor-based for-fee reimbursement, a process could be established, whereby the vendors work with the Voting System Testing

[4] A False Positive is a result that is reported/identified as a defect when in fact it is not.

Laboratories (VSTLs) to scan their software source code prior to and during the EAC certification process. Greater tool availability would result in greater source code assuredness, increased cost efficiencies in performing automatic and manual code reviews, and more consistency in the development and application of standards applied to voting systems – an issue identified as problematic in a previous FVAP study.

1.3 Organization of this Report

This final report summarizes a project consisting of multiple tasks, sequentially listed below by applicable project work statement number.

- 2.5.1: Project Management Plan and Research Plan
  - 2.5.1.1: Setup Phase:
    - Access to source code and web application from three EAC-certified internet voting system vendors (IVS)
    - Setup and configuration of virtual hosting environment (VHE)
    - Installation of IVS source codes and web applications on VHE
    - Installation of software assurance tools (SATs) on VHE
    - Setup and configuration of external communication channel with subcontractor
  - 2.5.1.2: Baseline Analysis of IVS Source Codes and Web Applications
- 2.5.2: Testing and Validation of SAT Suites
- 2.5.3: Handbook for Software Assurance Testing

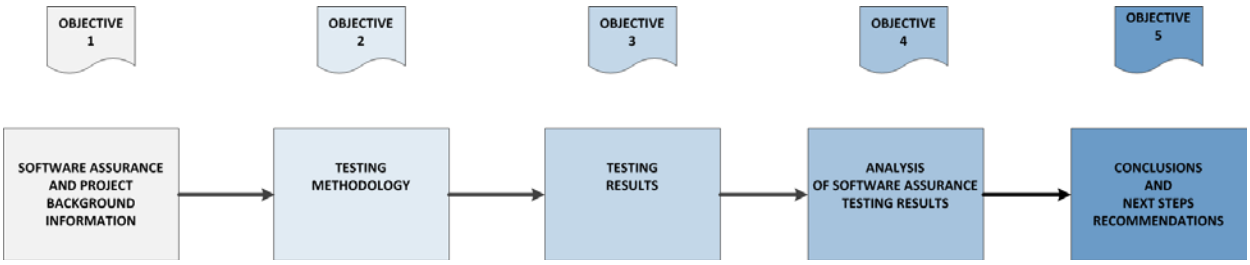


Figure 1.1: Report Workflow. *This report provides details as to why this research was conducted, how it was set-up, findings, and the implication of results.*

The full scope of this effort is intended to achieve five objectives, reflected in the following report outline, and illustrated in the workflow in Figure 1.1:

Chapter 2 Provide background information on software assurance, associated tools, software defects, corresponding catalogs, and risk ratings.

Chapter 3 Provide testing protocol and methodology, including tool and election software installation, baseline scans, tool modifications, third party methodology validation and user manual development.

Chapter 4	Provide in-house testing and third party validation results.
Chapter 5	Provide in-depth analysis of results aiming at the most efficient use of the proposed suite of tools, for each vendor.
Chapter 6	Provide conclusions and recommendations for next steps (e.g., additional research with regard to the use of multiple SA tools or additional security analysis of internet voting software).

2 Project Background

Under Phase 1 of the current contract, CALIBRE conducted in-depth research and developed reports on information assurance tools, specifically, SA tools, intrusion detection systems, and intrusion recovery systems, for potential application in the testing of internet voting systems in support of a potential internet voting demonstration or pilot project deployment. The first of the reports, *Assessment of Software Assurance Tools for Improving the Security of Voting Systems*, provided a thorough and comprehensive analysis of SA tools.

The analysis provided detailed information about SA tools available in the marketplace and developed a methodology for assessing their usefulness for FVAP's purpose. The report proposed a recommended toolbox of 23 tools that could be used for testing the three internet voting systems currently certified by the EAC. A tailored suite of five to six software assurance tools was developed for each voting system depending on their programming languages and operational environment. Each tailored suite consisted of three or four static code analysis tools, a dynamic analysis tool, and a database analysis tool. The rationale for the use of several static code analysis tools derives from research done by the National Security Agency (NSA) Center for Assured Software (CAS).[5] CAS has developed a methodology for the performance assessment of static analysis tools and performed testing with static analysis tools on code bases in several programming languages.[6] As discussed in the *Static Analysis Tool Study Methodology* report, the detection percentage of True Positive defects increased as additional tools were combined; however, this increase in detection efficiency plateaued once the combination of tools was greater than three. Based on these findings, a minimum of three static code analysis tools were selected for each suite used to test the EAC-registered election software.

The current effort outlined in this report examines the effectiveness of the tailored suite of tools against the code and application of three EAC-registered internet voting systems and the ability to optimize the tools in order to automatically reduce the number of False Positives detected during scans. This effort is not intended to assess, nor draw any opinions or conclusions, on the security posture of the voting systems. The remainder of this section provides background information on software assurance tools, associated tools, glossary, and election software tested during the project.

---

[5] National Security Agency, Center for Assured Software. 2011. CAS Static Analysis Tool Study – Methodology. http://samate.nist.gov/docs/CAS%202011%20Static%20Analysis%20Tool%20Study%20Methodology.pdf

[6] National Security Agency, Center for Assured Software. 2012. SATE IV Workshop March 29, 2012 - *Sticking to the Facts II: Scientific Study of Static Analysis Tools*. http://samate.nist.gov/docs/SATE4/SATE%20IV%206%20Stick%20to%20Facts%20II%20Erno.pdf

2.1 Software Assurance Tools

The NIST Software Assurance Metrics And Tool Evaluation (SAMATE) project defines software assurance as the planned and systematic set of activities that ensure that software processes conform to requirements, standards, and procedures to help achieve:[7]

- Trustworthiness – minimize exploitable defects, either of malicious or unintended origin, thereby enhancing the security of the software.
- Predictable execution – promotes confidence that software reliably functions as intended by eliminating coding flaws and weaknesses.

Software assurance activities include both processes (e.g., manual code review) and automated tools that constitute a critical portion of the SDLC to produce reliable software, as shown in Figure 2.1.

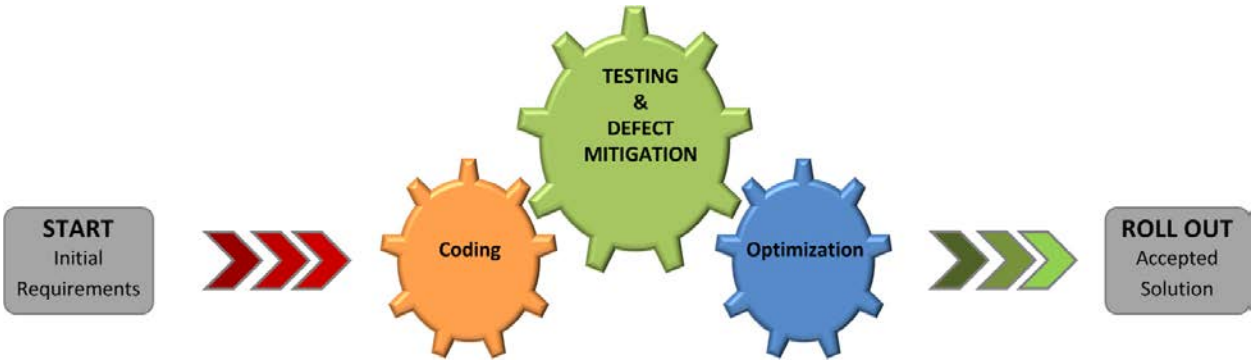


Figure 2.1: Software Development Lifecycle. *Once initial requirements are defined, coding, testing & defect mitigation, and optimization are essential to the process of determining an accepted solution.*

The amount of user effort required to apply SA tools varies significantly within a broad range of automated, semi-automated, or manual interfaces and inputs, and whether they are commercial products or open-source projects. Nevertheless, in each case, their purpose is to test for flaws within the software coding environment. SA tools are categorized and evaluated based on their analysis, methodology, and purpose.

Ultimately, no software is flawless but the proper application of SA analysis will help to identify defects and errors in code, and minimize or help mitigate known defects and weaknesses while delivering an acceptable level of risk conducive to the system deployment.

2.1.1 Static and Dynamic SA Tools

Software assurance involves the use of static analysis tools to perform code review and dynamic analysis tools to identify and/or exploit defects in operational applications and databases. NIST

[7] NIST. Software Assurance Metrics and Tool Evaluation. http://samate.nist.gov/Main_Page.html

defines static analysis as the examination of software code to determine its quality and the potential need for remediation or mitigation. It also states that dynamic analysis is used to examine the behavior of software in operation.[8] This NIST research involved the evaluation of both static and dynamic analysis tools to provide a holistic assessment of tool effectiveness. Below, static and dynamic analyses are described in further detail.

Static: Static analysis is conducted on code at rest, not in a run-time environment. The code is not executed or run but the SA tool itself is executed, and the source code is the input data to the tool. Static analysis tests software for code patterns that violate defined coding best practices, revealing defects, bugs, weaknesses, and security flaws. In addition to ensuring that code meets uniform expectations for regulatory compliance or internal initiatives, static analysis also helps to prevent defects such as resource/memory leaks, performance and security issues (e.g. buffer overruns), logical errors (e.g. misuse of negative integers), and application programming interface (API) misuse. Static analysis tools are generally used by developers to evaluate source code in the development and component testing processes.

Most static analysis tools can be integrated within a developer's respective Integrated Development Environment (IDE) to be used during development, increasing the potential of detecting and mitigating defects earlier in the SDLC. Static analysis tools may be utilized as soon as software code can be compiled; they do not require the program to be complete.

The following are features or characteristics of static analysis tools:

- Calculation of metrics such as cyclomatic complexity or nesting levels (which can help to identify where more testing may be needed due to increased risk).
- Enforcement of coding standards.
- Analysis of structures and dependencies.
- Help in code understanding.
- Identification of anomalies or defects in the code.

Static code analysis is valuable because it involves objectively exposing potential defects. It is important to note that static analysis tools have significant limitations. Like many other tools, static analysis cannot aid in identifying architectural-level flaws – the tools cannot detect when a system performs unexpected operations and is therefore functionally unreliable. In addition, these tools require the source code of the application be available for compiling. For the purpose of this effort, the vendors provided access to the source code of their otherwise proprietary software.

---

[8] NIST. Source Code Security Analysis Tool Functional Specification Version 1.1. NIST Special Publication 500-268 v1.1. February 2011. http://samate.nist.gov/docs/source_code_security_analysis_spec_SP500-268_v1.1.pdf

Dynamic: Dynamic analysis occurs when software is operating in a real or virtual operational environment. Unlike its static counterpart, dynamic analysis does not test at the code level, and instead attempts to detect and exploit defects or flaws by performing simulated attacks (e.g., injection or cookie testing). Through automation, dynamic analysis tools search for a range of defects including input/output validation, configuration errors, and application issues. This allows for organizations to detect defects in released software in ways similar to malicious attackers – by performing actual attacks against a running application based on known defects. While static analysis identifies specific lines of code with potential weaknesses and defects, dynamic analysis attempts to exploit the existing defects.

Dynamic analysis tools are used after development, and, unlike static tools, require complete systems to perform their tests. The dynamic analysis tools use test and evaluation scenarios on a program executing data in real-time, with the goal of identifying security defects while the application is running. Dynamic analysis tools often require an understanding of the application's build process and its composition in order to best identify False Positives. In most environments, the primary drawback associated with dynamic analysis is the high level of expertise required to discern False Positives.

Dynamic analysis is a critical component of any overarching security posture for potential internet voting demonstration projects, as critical defects in the software may only surface during dynamic testing. However, dynamic analysis tools are not tailored to a specific system, but are designed to run through a particular set of tests. Unlike static analysis, dynamic analysis does not look at source code and, therefore, does not examine coding errors that could have been added by malicious individuals**.**

2.1.2 False Positives

Because static code analysis tools are designed to look for patterns, they often fail to see the larger picture, and may report inaccurate or unexpected results. False Positives occur when automated tools identify defects that are ultimately deemed not defective code. When scanning a system, static analysis tools identify defects that they deem security issues or code weaknesses. The reviewer must assess the identified defect against the source code to determine whether it is a True or a False Positive (i.e., if the defect actually exists or if the tool is identifying a defect that is not truly present in the source code).

The value of tools can be rapidly diminished by too many False Positives – in theory, the more customizable a tool is, the easier it is to filter out False Positives, thus allowing development teams to focus their efforts on code with actual defects. Therefore, one of the original goals of this research was to assess baseline rates of False Positives returned by the SA tools, and to ascertain how easily the tools could be customized to filter out False Positives.

2.1.3 Combining Multiple Tools

A single SA tool may only detect a portion of the potential defects present in the source code, and likely does not provide a comprehensive security approach. The combination of multiple SA tools with differing capabilities, in conjunction with manual reviews by skilled software testers, can provide a more comprehensive security review of software and increase the detection of True Positives. Recent research supports this model – as mentioned earlier, the NSA CAS issued a 2011 report concluding that the combination of multiple static SA tools could significantly increase the detection rate of True Positives during testing.[9] However, this study used artificial test cases (source code intentionally injected with a known number of defects) instead of natural code (production source code from a commercially available application). The findings of the NSA study, therefore, do not reflect the reality of commercially available election software. The current research assesses the impact of combining multiple tools against the production source code from each of the three EAC-registered voting system vendors.

2.1.4 Previous Toolbox Development

As part of FVAP's 2012 industry and market research to narrow the field of available SA tools, 23 tools were selected as the most potentially useful in the examination of vendor software. These 23 tools included both static and dynamic analysis tools. This toolbox was then narrowed to the five to six tools most appropriate for each voting system vendor.[10] Figure 2.2 contains the initial toolbox CALIBRE formulated for this project, which includes the SA tools chosen based on an initial understanding of the vendor software.

Tool Type	Tool Name
Data Base Scanner	AppDetective Pro 7.X
Source Code Security Analyzer	DMS Software Re-Engineering Toolkit
Source Code Security Analyzer	HP Fortify SCA
Source Code Security Analyzer	Parasoft C/C++test, Jtest
Source Code Security Analyzer	Coverity
Web Application Scanner	HP WebInspect

Figure 2.2: Original Toolbox. *CALIBRE previously recommended a toolbox based on our initial knowledge of vendor software and SA tool capability.*

2.2 Risks and Defects

2.2.1 Catalogs of Software Defects

The IT security industry has experienced challenges in relation to the categorization of various defects and their resulting risks. Modern security ecosystems use varying and often incompatible

[9] National Security Agency, Center for Assured Software. 2011. CAS Static Analysis Study – Methodology. http://samate.nist.gov/docs/CAS%202011%20Static%20Analysis%20Tool%20Study%20Methodology.pdf

[10] FVAP. Assessment of Software Assurance Tools for Improving the Security of Voting Systems, 16 December 2012.

organizations, vendors, and security practitioners to detect, manage, and control threats in an ever-changing environment. In this complex environment, several taxonomies currently exist as industry standards for the identification and categorization of risks and defects. In order to evaluate and compare the effectiveness of multiple SA tools, the research team chose to use several common defect taxonomies to simplify and present results in this report.

CWE: The Common Weakness Enumeration (CWE) provides a unified and measurable catalog of over 900 software defects to serve as a common language for describing software security weaknesses in the operational system and source code.[11] The list is created from the input of security researchers all over the world and maintained by Mitre Corporation (Mitre). By creating a better understanding of architectural and design weakness, the project aims to assist SA tools in their identification, mitigation, and prevention efforts. However, the CWE is very detailed in its description of the weakness, and does not provide a rank order of most critical defects.

SANS Top 25: The SANS Top 25 Most Dangerous Programming/Software Errors is a ranking of the most widespread and critical errors that can lead to serious defects in software.[12] The annual list is the result of collaboration between the SANS Institute, Mitre, and many top software security experts in the U.S. and Europe. The SANS Top 25 list presents detailed descriptions of the top 25 programming errors along with authoritative guidance for mitigation and avoidance. The SANS Top 25 listing correlates exactly with the numbering and classification in the more detailed CWEs, but also provides a ranking scale of these defects. However, no new listing has been released since 2011.

OWASP Top Ten: The Open Web Application Security Project (OWASP) is an open-source web application security project that is emerging as a standards body for the field.[13] Every three years OWASP releases its list of the Top 10 most critical web defects, representing a broad consensus of the most critical web application security risks. Figure 2.3 depicts both the 2010 and 2013 OWASP rankings, which shows that over time many of the most threatening defects remain ranked at the top. The goal of the Top 10 project is to raise awareness about application security by identifying some of the most critical risks facing organizations. The Top 10 project is referenced by many standards, books, tools, and organizations, including the Defense Information Systems Agency (DISA), Mitre, the Payment Card Industry Data Security Standard (PCI DSS), and the Federal Trade Commission (FTC).

---

[11] Mitre Corporation. Common Weakness Enumeration: A Community-Developed Dictionary of Software Weakness Types. CWE Version 2.6. February 19, 2014. http://cwe.mitre.org/data/published/cwe_v2.6.pdf

[12] SANS Institute. CWE/SANS Top 25 Most Dangerous Software Errors, Version 3. June 27, 2011. http://www.sans.org/top25-software-errors/

[13] OWASP Foundation. 2013 Top 10 List. June 23, 2013. https://www.owasp.org/index.php/Top_10_2013-Top_10

Figure 2.3: OWASP Rankings - 2010 and 2013. *Every three years, OWASP releases new defect rankings to raise awareness about software security by identifying the 10 most critical risks.*

Coding Best Practices: In addition to identifying the most dangerous defects and programming errors, evaluations of software code often consider industry best practices as part of a complete review. Violations of these practices do not necessarily lead to defects, but SA tools often flag issues because they make code more susceptible to attacks or errors. Two of the leading industry best practices for secure coding are the Department of Homeland Security's (DHS) U.S. Computer Emergency Readiness Team's (U.S. CERT) Build Security In (BSI),[14] and the Software Engineering Institute CERT Division's Secure Coding Standards for commonly used programming languages (e.g., C, C++, Java, and Perl).[15]

BSI is intended for use by software developers and software development organizations who want information and practical guidance on how to produce secure and reliable software. BSI content is based on the principle that software security is fundamentally a software engineering problem and must be addressed in a systematic way throughout SDLC. BSI contains a broad range of information about best practices, tools, guidelines, rules, principles, and other knowledge to help organizations build secure and reliable software.

The CERT Secure Coding Standards itemize those coding errors that are the root causes of software defects and prioritizes them by severity, likelihood of exploitation, and remediation costs. Each guideline provides examples of insecure code as well as secure, alternative implementations. If uniformly applied, these guidelines eliminate critical coding errors that lead to buffer overflows, format string defects, integer overflow, and other common software defects.

[14] For more information: https://buildsecurityin.us-cert.gov/
[15] For more information: https://www.cert.org/secure-coding/

2.2.2 Defect Risk Ratings

It addition to categorizing security risks and defects in diverse ways, security tools often use multiple and differing scales to present the potential risk of discovered defects. As shown in Figure 2.4, below, for the SA tools evaluated for this report, two tools use a *Low*, *Medium*, *High* severity scale, three tools use a larger scale that incorporates *Critical*, and one tool presents a list of lowest to highest priority issues without assigning categories. Figure 2.4 indicates which levels of classification are present for the given SA tool to provide a more detailed understanding of their differences in severity ratings.

SA Tool	Critical	High	Medium	Standard	Low
HP Fortify	X	X	X		X
Coverity		X	X		X
VCG	X	X	X	X	X
RATS for Perl		X	X		X
Perl::Critic*					
HP WebInspect	X	X	X		X

Note: VCG had 2 additional categories: Suspicious Comments and Potentially Unsafe

Note: HP WebInspect had 2 additional categories: Informational and Best Practice

*Perl::Critic has five severity rankings, Severity 1 (lowest) through Severity 5 (highest)

Figure 2.4: SA Tools Defect Scale. *Due to the differences in severity categories the SA tools present, CALIBRE elected to re-assess all classifications into a High, Medium, Low scale to more easily compare findings across tools.*

In order to compare results across these different tools, the research team chose to use a single, three-value (*Low*, *Medium*, and *High*) Likert-type scale. This scale collapsed *Critical* issues into the *High* category, and assigned a category based on CWE and OWASP criteria to those tool results which presented an uncategorized list.

3 Testing Protocol/Methodology

To conduct this research, FVAP and the CALIBRE research team worked cooperatively with the three EAC-registered internet voting system vendors. It is important to note that there was no requirement or monetary incentive for vendor participation – the three vendors voluntarily shared their source code and provided developmental and operational virtual machines (VMs) as a testing environment for the SA tools, and in return, received valuable information on risks and defects detected by the tools.

This chapter outlines the general process for the installation of the election software and SA tools, the baseline scans, the subsequent modifications of the SA tools, and the final third-party validation of the results and methodology. Each section contains a summary of difficulties encountered, steps taken to remedy the issue(s), and/or key lessons learned. Lastly, this chapter contains a brief description of the Installation and User's Manual developed for each tool with each vendor's software.

3.1 Set-Up

3.1.1 Installation of Voting Systems

The research team requested two VMs from each voting system vendor; one containing the source code and compiler, and a second holding their operational system. After the secure transfer of the VMs, the research team uploaded and activated them on CALIBRE's secure internal network.

This method of environmental set-up was chosen due to several advantages:

- **Security**: VMs provided security for the proprietary source code and applications to be tested; authentication and access were logged and audited, and there was a mechanism for disaster recovery.
- **Accurate Representation**: Using the vendors' VMs enabled testing using source code, applications, and environments free of configuration issues.
- **Resource Utilization**: Multiple VMs could co-exist on the same host computer, in strong isolation from each other, providing improved availability for the research team, and easier maintenance and management due to the use of fewer servers.

Initial installation of the vendors' systems required frequent and extensive dialog with vendor technical representatives. Set-up took less time for vendors that provided instructions and detailed information, and more time for vendors who simply sent VMs with no associated documentation. The VMs provided by the vendors varied in their presentation; while one vendor sent a combined operational and developmental VM, another vendor provided a development machine and instructions on set-up, thereby requiring the research team to configure the machine. Therefore, set-up timelines ranged from two weeks to two months, with the availability of vendor representatives being a key factor to prompt set-up. Figure 3.1 shows a high-level depiction of the network set-up CALIBRE and Pro V&V utilized.

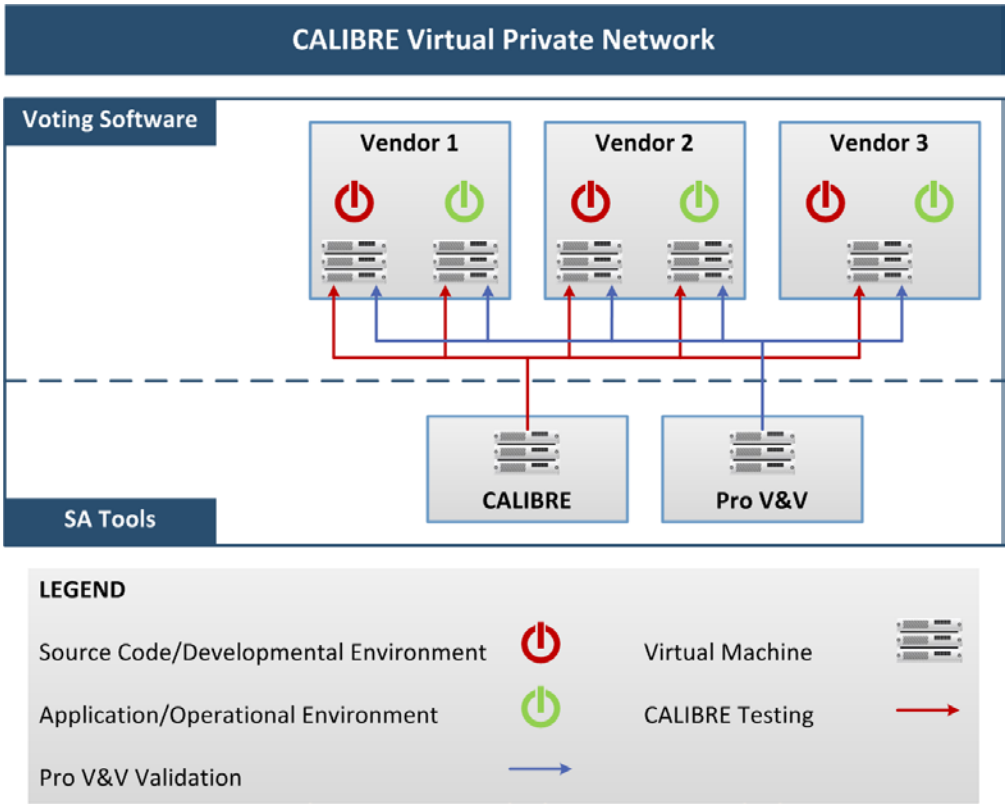


Figure 3.1: Virtual Private Networks. *Vendors provided their operational and developmental VMs, which were installed on CALIBRE and Pro V&V's machines.*

Key lessons learned for future efforts include:

- Use a detailed checklist prior to the effort to ensure accurate understanding of each system's technical details (e.g., source language, operating system, database, web server, etc.). A sample checklist has been developed and can be found in Appendix A.
- Ensure a single technical point of contact from each voting system vendor, who understands what the voting system vendor provided, is knowledgeable regarding the system specifics, and is available to answer questions within a reasonable timeframe.
- Require vendors to provide separate VMs for the development and operational environments, and complete documentation of what is on each VM and instructions on how to compile their code.
- Require vendors to provide detailed directions for VM set-up, including all required components and any required user IDs or passwords necessary for the operational and developmental systems.

3.1.2 SA Tool Installation

The SA tools used in this project evolved as the project progressed. The initially identified tools discussed in section 2.1.4 served as the starting point for building a toolbox of five or six tools

(three or four of which are static analysis tools) tailored to each vendor's software. This initial set of tools was based on the extensive research done on the tools for the *Assessment of Software Assurance Tools for Improving the Security of Voting Systems* report and the information provided by the voting system vendors regarding the source code language used by their respective systems. Based on the initial toolbox, CALIBRE purchased all applicable tool licenses[16] and downloaded instructions and keys from the tool manufacturer. The team then followed step-by-step installation instructions or guides provided by each tool manufacturer to install the tools on its scanning VM.

As the research team prepared to begin the baseline scans, several issues with the tool suites were discovered, despite a thorough preliminary vetting process. As detailed below, CALIBRE received incorrect information from both voting software and SA tool manufacturers and faced numerous challenges negotiating with the latter for issue resolution. As a result, with the approval of FVAP, some of the SA tools in the original toolbox were replaced in order to maintain the previously agreed toolbox format of three static code analyzers, one database scanner, and one web application scanner for each tested internet voting system:

- One of the tools selected to test the PERL programming language, Design Maintenance System (DMS), did not provide a defect assessment. Because DMS is one of the few commercially available scanners for PERL, CALIBRE, with FVAP approval, selected two open source products as replacements: Rough Auditing Tool for Security (RATS) and Perl::Critic.
- Parasoft C\C++ test was purchased with the intent of using the tool to scan the code of two voting system vendors and was based on information provided by these vendors; however, when the vendors' source code was received and initially tested, none of it was compatible with the Parasoft tool – the code was written in Java. CALIBRE attempted to negotiate with Parasoft to change the C++ tool/license for a Java scanning tool license. Parasoft was unwilling to switch CALIBRE's license. After much negotiation Parasoft relented and we were able to exchange the C/C++ license for a Jtest license. Even after acquiring the Parasoft tool in the correct language we were unable to get the two vendors' code to work with Parasoft. One vendor's code required an additional plug-in, and due to security concerns on their part, they were unwilling to supply the plug-in. Likewise, due to a compiling issue, we were unable to successfully use Jtest with the second vendor's code.

The final set of tools used for this analysis consisted of:

- Source Code Static Analysis Tools
    - HP Fortify

[16] CALIBRE was able to negotiate three thirty-day licenses for HP Fortify, rather than purchase a full year license.

    - Coverity
    - VCG
    - RATS for PERL
    - Perl::Critic
- Dynamic Analysis Tools
    - HP WebInspect (Web Application Scanner)
    - App Detective (Database Scanner)

Tool installation was generally straightforward, although the open source tools provided little documentation and instructions, and thus took longer to install. Figure 3.2 documents issues with tool installations.

SA Tool	Issue
App Detective	• Would frequently fail to install, requiring a new computer each time.
HP WebInspect	• No issues with installation.
HP Fortify	• No issues with installation.
Coverity	• No issues with installation.
Parasoft	• No issues with installation.
VCG	• Took longer to install due to lack of documentation.
RATS for PERL	• Took longer to install due to lack of documentation.
Perl::Critic	• Took longer to install due to lack of documentation.

Figure 3.2: Installation Issues by SA Tool. *The majority of SA tools were installed on the VMs without incident, though there was greater difficulty doing so with open-source tools.*

3.2 Test Plan and Scanning

The original test plan CALIBRE developed was based on the assumption that the SA tools would not be difficult to modify and/or customize to filter out False Positives from the final results. The plan consisted of the following steps:

- Conduct baseline scan using SA tool's default (i.e., out-of-the box) configuration on each internet voting system source code.
- Send baseline scan test results to internet voting system vendors for review/assessment; vendors to focus assessment on classifying *Critical*/*High* defects found by static tools as True or False Positive.
- Review vendor False Positive assessment and make final determination of True or False Positive.
- Modify/customize static tools to filter/eliminate False Positives.
- Re-run static tools resulting in an optimized scan.
- Analyze optimized test results against baseline to assess effectiveness of tool modification/customization.

Based on the findings discussed in section 3.3, and the recognition that there was no change to the actual code following True/False Positive assessment, the test plan was modified, and approved by FVAP, as follows:

- Conduct scan with minor modifications to SA tool's default configuration.
- Send baseline scan test results to internet voting system vendors for review/assessment; vendors to focus assessment on classifying *Critical*/*High* defects found by static tools as True or False Positive.
- Review vendor False Positive assessment and make final determination of True or False Positive.

The research team conducted a scan for each voting system vendor/tool combination using the default configuration provided by the SA tool manufacturer. The research team did not utilize any customization options offered by the tools (e.g., settings to turn various audits on or off), with one exception – where tools allowed it, the research team specified the most intense possible scan of the code.

Overall, the scans were conducted without major issues; however, several vendor-specific problems occurred while conducting these scans, as detailed in Figure 3.3.

SA Tool	Issue
App Detective	• Required high-level administrative access to scan the database – without such access, or with a low-privileged account, the tool would not function.
HP WebInspect	• Scans took much longer than other tools (almost a day versus approximately 15 minutes) due to number of tests and iterations performed.
HP Fortify	• Despite having a plug-in for Microsoft Visual Studio, HP Fortify failed to conduct scan of a vendor's code initially. o Running HP Fortify via command line did not fix this issue. o The only work-around, provided by HP Support, was to pre-compile the Visual Studio code and point HP Fortify to the compiled results for scanning.
Coverity	• In order to use Coverity with a vendor's compiler, an extra flag/command was necessary, making Coverity somewhat more complicated to use for this system. • Coverity required added memory (2500 MB) beyond the 1024MB default setting to scan a vendor's code. • Vendor 3 didn't provide us with compiling instructions so it couldn't run.
Parasoft	• Parasoft Jtest is designed for systems that compile with Eclipse. One vendor uses a different compiler, thus Parasoft Jtest could not scan the vendor's code due to its external repository settings and the need for a security plug-in, which the vendor was unwilling to supply for security reasons. • A second vendor did not provide compiling instructions and thus we were unable to utilize Parasoft on their software.
VCG	• No issues.
RATS for PERL	• No issues; Comment - RATS for PERL is a command-line tool, which is less user-friendly than other tools with graphical user interfaces.
Perl::Critic	• No issues; Comment - Perl::Critic is also a command-line tool.

Figure 3.3: Baseline Scan Issues by SA Tool. *Both the dynamic tools ran into issues across all vendors, while the static tools – with the exception of the open source tools – generally had issues with specific vendor software.*

3.3 Assessment and Adjudication of True/False Positives

After the scans were conducted, all tool reports were converted to readable formats and provided to the internet voting system vendors for True/False Positive assessment, as well as for their own situation awareness, as described in Figure 3.4.

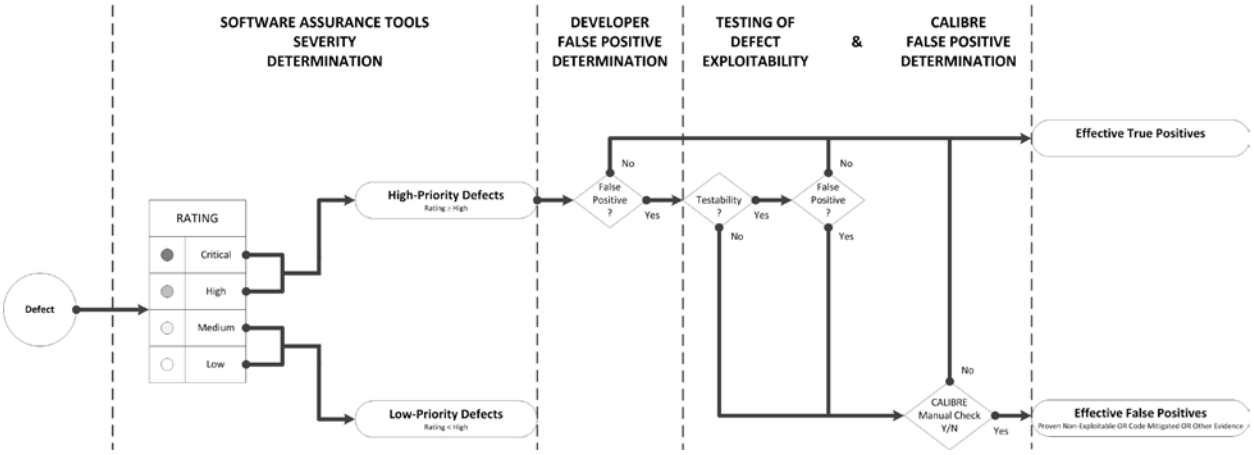


Figure 3.4: Adjudication Process for True/False Positives Defects. *To conduct False Positive adjudication, the research team provided the vendor with baseline scan results. If the vendor indicated a False Positive, CALIBRE manually checked the source code against the information provided by the vendor.*

It should be noted that while both static and dynamic test results were provided to the voting system vendors, CALIBRE only requested that the results of the static tools be assessed, as dynamic results are much more difficult to analyze, as discussed in section 2.1.1. The numbers of identified defects varied widely by voting system – with aggregate results from the three static code tools reporting over 1,300 *High*, *Medium*, and *Low* defects for one vendor, over 3,600 for another, and over 2,600 for the third. It should be noted that the defects identified included both security related issues and coding best practice issues.

Although the research team asked vendors to assess all results to identify False Positives, if vendors did not have resources to evaluate all identified defects, they were asked to focus on those categorized as *Critical*/*High* by the tools.

The False Positive identification and adjudication process required extensive and continuous dialog with the voting system vendors, and thus heavily depended on vendor cooperation. As vendors volunteered their code for testing, CALIBRE did not require a set process for True/False Positive assessment. This resulted in divergent levels of detail and feedback received from the vendors, as shown in the way each vendor conducted its assessment:

- One vendor looked at all reported defects regardless of severity level, and provided detailed feedback and evidence to the research team regarding False Positive assessments.
- A second vendor evaluated a random sample of the *Critical* and *High* defects in every reported category. When the random sampling revealed that all defects by category and tool were True Positives, this vendor classified all such defects as True Positives.
- The third vendor examined one defect per issue category (as identified by the scanning tool) before generalizing those results to the broader category. In addition their responses were vague (e.g., "This is a theoretical problem which arises based on how you use the JS. It would be good if the tool examined the usage of JS and only reported real issues").

However, the vendors usually followed the process described in Figure 3.5.

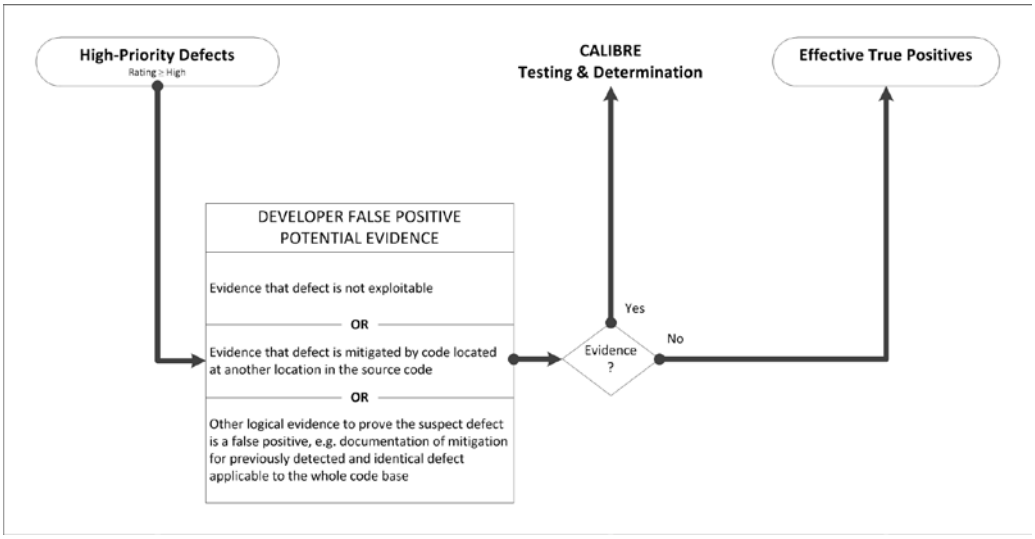


Figure 3.5: Process Used by Internet Voting System Vendors to Assess True/False Positives.
Vendors needed to provide sufficient evidence to justify a False Positive determination.

Following vendor assessment, the research team reviewed the documentation provided by the vendors and independently, via manual code review or using tools such as memory profilers, classified True and False Positives in the reports, as described in Figure 3.6.

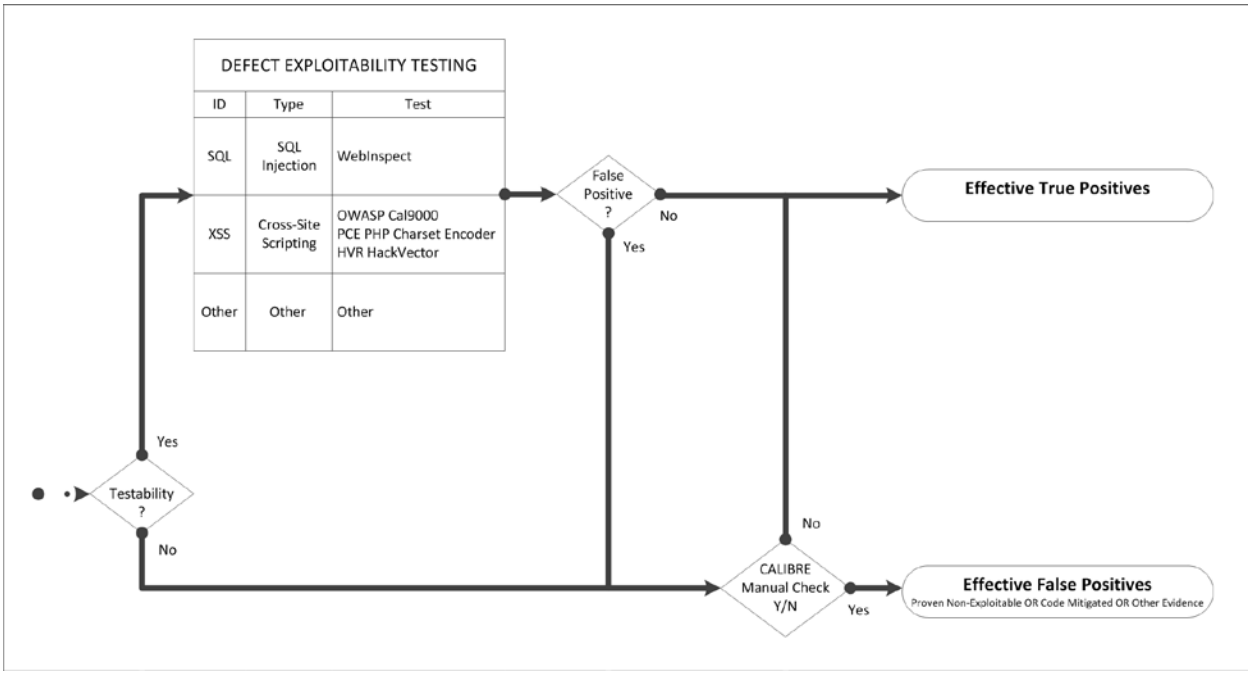


Figure 3.6: CALIBRE's Adjudication Process of Vendors' Assessment. *CALIBRE reviewed the voting system vendors' False Positive assessment and attempted to verify their findings using automated tools or manual code review.*

In all but one instance, CALIBRE agreed with the vendor's assessment. The assessment and adjudication process took approximately six weeks, on average, for each vendor.

Key lessons learned for future efforts include:

- Provide vendors detailed information regarding the True/False Positive assessment and adjudication process early in communications, including timeframes for when they will receive findings and requested dates by which to review and return their results. Build in time for back and forth discussions regarding their False Positive assessments.
- Provide vendors with sample feedback which outlines the level of detail requested from them for their False Positive assessment.
- Ensure a single technical point of contact from each system vendor, who is knowledgeable regarding the True/False Positive assessment process, and available to answer questions within a reasonable timeframe.
- Standardize SA tool output to the greatest extent possible before providing it to vendors, and include a template for response that provides sample results and thorough and complete example descriptions of False Positives feedback.

3.4 Tool Optimization

The SA tools utilized for this analysis provided for different approaches to modification or customization of the tool, including: custom rules/rule sets, filtering/customization of files scanned, scanning intensity settings, and suppression. These techniques are described in more

detail below and Figure 3.7 presents a table comparing the tools and techniques available in each tool:

- **Custom Rules/Rule Sets:** The user can construct custom rules capable of searching the source code for suspected coding issues or security defects, and eliminating issues. Two tools, HP Fortify and VCG, allowed for the creation of custom rules. Coverity and Parasoft allowed easy-to-use on/off options for various pre-defined rule sets making customization more user-friendly. It should be noted that little professional literature exists for the creation of custom rules with HP Fortify. And although HP Fortify provided the research team with a one-day training session for creating custom rules, the instructor had considerable difficulty constructing useable custom rules with the vendor's source code.
- **Filter/Customize Files Scanned:** Source code sometimes contains notes or documents that are not part of code execution. These portions of the code can produce False Positive defects when scanned by automated SA tools. File filtering allows users to tag these files to avoid False Positive defects. All six static analysis tools allowed for file filtering/customization, although the research team did not use this customization option because all files sent by the vendors were executable and therefore did not need to be filtered out of the scanning process.
- **Scanning Intensity:** SA tools are typically set to scan the code a set number of times. However, five tools (two dynamic; three static) allowed this number to be altered, potentially increasing the intensity of the scan to locate more issues. The default setting on the tools was usually *Medium*, but the research team increased scan intensity to the maximum allowed (*High*) for all tools where this was an option, in order to allow for the most intensive scanning allowed by each tool.
- **Suppression:** Suppression can be used to help fine tune scan results and keep displayed warnings relevant, by suppressing warnings for specific types of issues that might not be high priority or of immediate concern. The user may elect to suppress issues labeled as *Low* in priority to focus attention on issues of greater significance. Suppression alters the visibility of defects in produced reports, enabling users to quickly change the sorting and visibility of issues. Five tools (two dynamic; three static) allowed for suppression.

SA Tool	Custom Rules	Filter Files	Scanning Intensity	Suppression
App Detective	N/A	N/A	X	X
HP WebInspect	N/A	N/A	X	X
HP Fortify	X	X		X
Coverity	X	X	X	X
Parasoft	X	X		X
VCG		X		
RATS for PERL		X	X	
Perl::Critic		X	X	

Figure 3.7: SA Tools' Optimization Options. *Each of the static tools provides different optimization options, though many of these were not feasible.*

As explained above, tool optimization options varied significantly across the SA toolboxes with open source tools having fewer options to modify the scans or final reports. Commercially available tools generally allowed for custom rules, filtering of files, changes to scan intensity, and suppression/auditing of results. However, the research team discovered that optimization beyond scan intensity and suppression/auditing generally required expert-level knowledge likely beyond the level of most users. Although the researchers originally planned to use all optimization options available for each tool, the team only used scan intensity, following training by the tool manufacturer. Figure 3.8 documents issues with tool optimization and the team's actions for resolution.

SA Tool	Issue
App Detective	• No optimization issues.
HP WebInspect	• No optimization issues.
HP Fortify	• Allows for custom rules, however development of rules requires expert-level knowledge.
Coverity	• No optimization issues.
Parasoft	• No optimization issues.
VCG	• No suppression/auditing function for reports.
RATS for PERL	• No suppression/auditing function for reports.
Perl::Critic	• No suppression/auditing function for reports.

Figure 3.8: Optimization Issues by SA Tool. *Open source tools required greater attention for optimization, while the other tools generally did not run into issues, with the exception of HP Fortify.*

3.5 Installation and User Manuals

All actions taken during the project were documented, including all changes made to the configuration of the scanning tools for each of the three vendors. The research team's documentation was used to create a customized Installation & User's Manual for each SA tool for use with each of the three internet voting systems. The manuals are provided on a separate disc accompanying this report.

Each user manual describes how to install the SA tool, how to conduct a scan, and how to generate reports. These manuals were provided to, and validated by, Pro V&V.

3.6 Pro V&V Validation

Pro V&V, Inc., is a test laboratory located in Huntsville, Alabama. Pro V&V received its Voting System Testing Accreditation from the NIST National Voluntary Laboratory Accreditation Program (NVLAP) in 2012, and has been audited by the EAC, successfully meeting all requirements for the EAC VSTL accreditation.

For this project, Pro V&V acted as an independent third party evaluator, taking all of the installation and user documentation, scanning tools, source code and applications, and testing results from the CALIBRE research team. The purpose of this evaluation was a validation of findings and a verification of user manual documentation.

Pro V&V utilized documentation created by the CALIBRE research team to install and configure each SA tool in a step-by-step manner. They then followed CALIBRE procedures for identification and adjudication of False Positives, and validated the CALIBRE findings. At the conclusion of this engagement, Pro V&V provided a test report detailing their findings and recommending any additional configuration changes that would further optimize each software assurance tool, as well as changes to the user manuals.

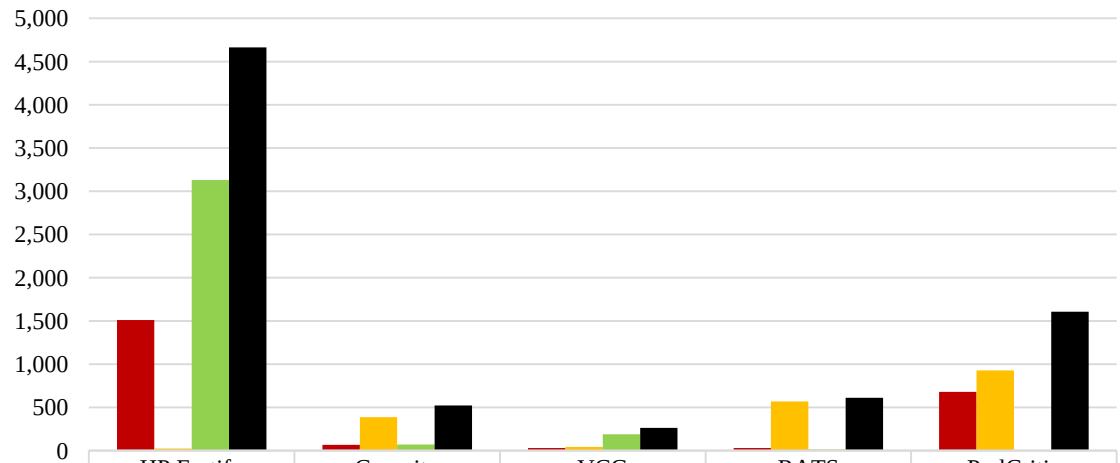
4 CALIBRE Results

Using the methodology described in Chapter 3, CALIBRE performed both static and dynamic testing on the software code provided by the three participating vendors.

In order to secure participation of the voting system vendors for this effort, CALIBRE agreed to not report the results based on the individual vendors. Therefore the test results are reported *by tool for all three vendors' code*. While this is an unusual method of reporting the results, the major conclusions drawn from the testing would not change had the results been reported by vendor. CALIBRE performed, but did not report on, a comparative analysis between each vendor's code with the three static tools, and the high level results are identical. As a result of differing levels of attention and effort provided by vendors to adjudicate False Positives, the analysis does produce differing results in terms of the True Positive rates, as described below by tool.

4.1 Static Analysis Tool Results

Figure 4.1 presents the testing results of all five static analysis tools. For each tool the count of the number of *Low*, *Medium* and *High* defects, along with the total, are displayed.



	HP Fortify	Coverity	VCG	RATS	PerlCritic
High	1,511	66	30	30	680
Med	24	386	43	568	927
Low	3,129	70	189	13	0
Total	4,664	522	262	611	1,607

Figure 4.1: Test Results for All Static Analysis Tools. *HP Fortify identified more defects than any other static analysis tool.*

HP Fortify, a commercial product which was used against all three voting system vendors, identified more *High* defects, as well as more overall defects, than any of the other tools. Coverity, also a commercial product, and VCG, an open source product, were used against two voting system vendors, while, RATS and Perl::Critic, both open source tools, were used against only one vendor. While Perl::Critic found the second most overall and *High* defects, the detail,

or lack of detail, provided in the Perl::Critic reports made the information of little value. Additionally, 99% of the defects found were potential best coding practices defects.

Even though HP Fortify was the only tool used against all three vendors, when the test results are viewed from a vendor perspective, HP Fortify found more total defects and more high defects than the other two tools tested against each voting system vendor's code. (Recall the test protocol was to use three static analysis tools against each voting system vendor's code.)

The test results for each tool are discussed in more detail in the following sections.

4.1.1 HP Fortify Source Code Analyzer

HP Fortify was used against the source code for all three internet voting system vendors. The rating scale used by HP Fortify is: *Critical*, *High*, *Medium*, and *Low*. The *Critical* and *High* defects were combined for comparison purposes, as discussed in Section 2.2.2.

Of the 4,664 defects identified by HP Fortify, 1,511 were classified as *High*, 24 as *Medium*, and 3,129 as *Low*.

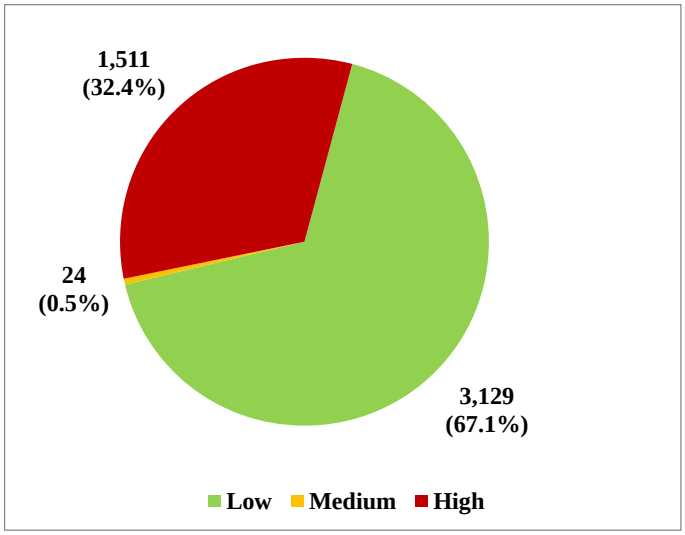


Figure 4.2: Defects Detected by HP Fortify by Severity Rating. *One third of defects detected by HP Fortify were rated High, while two thirds of were rated Low.*

After adjudication with the vendors, of the 1,511 defects classified as *High*, 68% (1,029) were judged to be True Positives (TP), while 32% (482) were considered False Positives (FP), as shown in Figure 4.3.

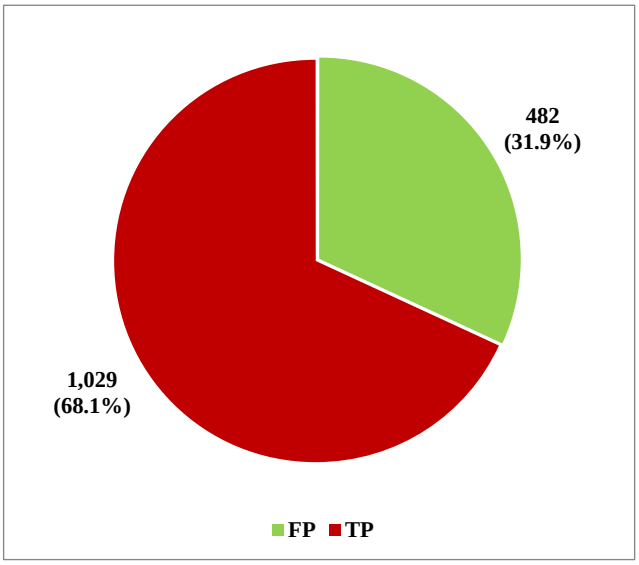


Figure 4.3: HP Fortify False Positive Adjudication Results. *68% of defects identified by HP Fortify were adjudicated as True Positives by vendors.*

As mentioned above, HP Fortify distinguishes between *Critical* and *High* defects in its reporting. Figure 4.4 and Figure 4.5 provide a breakdown by *Critical* and *High* and the adjudicated True Positive/False Positive findings.

Severity & Defect Category	TP	FP
Critical	**82**	**48**
Path Manipulation	41	2
Cross-Site Scripting: Persistent	21	7
Cross Site Scripting: DOM	8	
Dynamic Code Evaluation: Code Injection	4	
SQL Injection	4	
Privacy Violation	2	
Open Redirect	1	1
Password Management: Hardcoded Password	1	2
Privacy Violation (Privacy Violation)		28
Privacy Violation (Shared Sink)		8

Figure 4.4: *Critical*-Rated Defects Detected by HP Fortify. *About two-thirds of the Critical-rated defects HP Fortify detected were True Positives.*

Among the *Critical*-rated defects detected by HP Fortify, 63% were assessed to be True Positives. Of these 82 defects, 85% were associated with two categories, Path Manipulation and Cross-Site Scripting. For these two categories, 89% of the detected instances were assessed to be True Positives (70 of 79).

Severity & Defect Category	TP	FP
High	**947**	**434**
Null Dereference	268	8
Privacy Violation: Heap Inspection	229	
Unreleased Resource: Unmanaged Object	139	1
Log Forging	58	
Insecure Randomness	42	
Unreleased Resource: Streams	40	7
Path Manipulation	35	9
Privacy Violation: Heap Inspection(Shared Sink)	26	
Code Correctness: Regular Expressions Denial of Service	20	
Unsafe Native Invoke	16	
Access Control: Database	14	
Portability Flaw: File Separator	13	6
Password Management: Hardcoded Password	10	25
Header Manipulation: Cookies	6	
Unreleased Resource: Database	6	22
Denial of Service	5	
Weak Security Manager Check: Overridable Method	5	
Value Shadowing	4	
Command Injection	2	
Missing XML Validation	2	1
Password Management: Password in Configuration File	2	38
Weak XML Schema: Unbounded Occurrences	2	
ASP.NET Bad Practices: Non-Serializable Object Stored in Session	1	
Privacy Violation	1	4
XPath Injection	1	
Log Forging (Shared Sink)		234
Log Forging(Log Forging)		15
Often Misused: Authentication		2
Path Manipulation (Shared Sink)		2
Privacy Violation (Shared Sink)		60

Figure 4.5: *High*-Rated Defects Detected by HP Fortify. *69% of the High defects found by HP Fortify were assessed as True Positives, more than half of which were within two defect categories.*

Among the *High*-rated defects detected by HP Fortify, 69% were assessed to be True Positives. Of these 947 defects, 52% were associated with two defect categories, Null Reference and Privacy Violation: Heap Inspection. For these two categories, 98% of the detected instances were assessed to be True Positives (497 of 505). The 10 defect categories with the largest number of True Positives account for 92% of all the True Positives assessed. For these 10 categories, 97% of the detected instances were assessed to True Positives (873 of 898).

While the True Positive discussion and static analysis results stated above are accurate, the reporting of the HP Fortify results in aggregate for the three vendors is distorted by the

[redacted]

assessments done by the vendors. As discussed in section 3.1.2, the vendors all took different approaches, and levels of effort, to the assessment and adjudication process. The True Positive/False Positive analysis for HP Fortify when viewed by vendor is much different. One vendor assessed 10% of the HP Fortify *High* defects as True Positive; a second assessed 97% of the defects as True Positive; and a third had 100% adjudicated as True Positive.

The top 10 defect categories, by number of detected instances, account for 89% of all True Positive *High*-rated defects detected by HP Fortify (includes *Critical*-rated defects, as previously discussed), as illustrated in Figure 4.7.

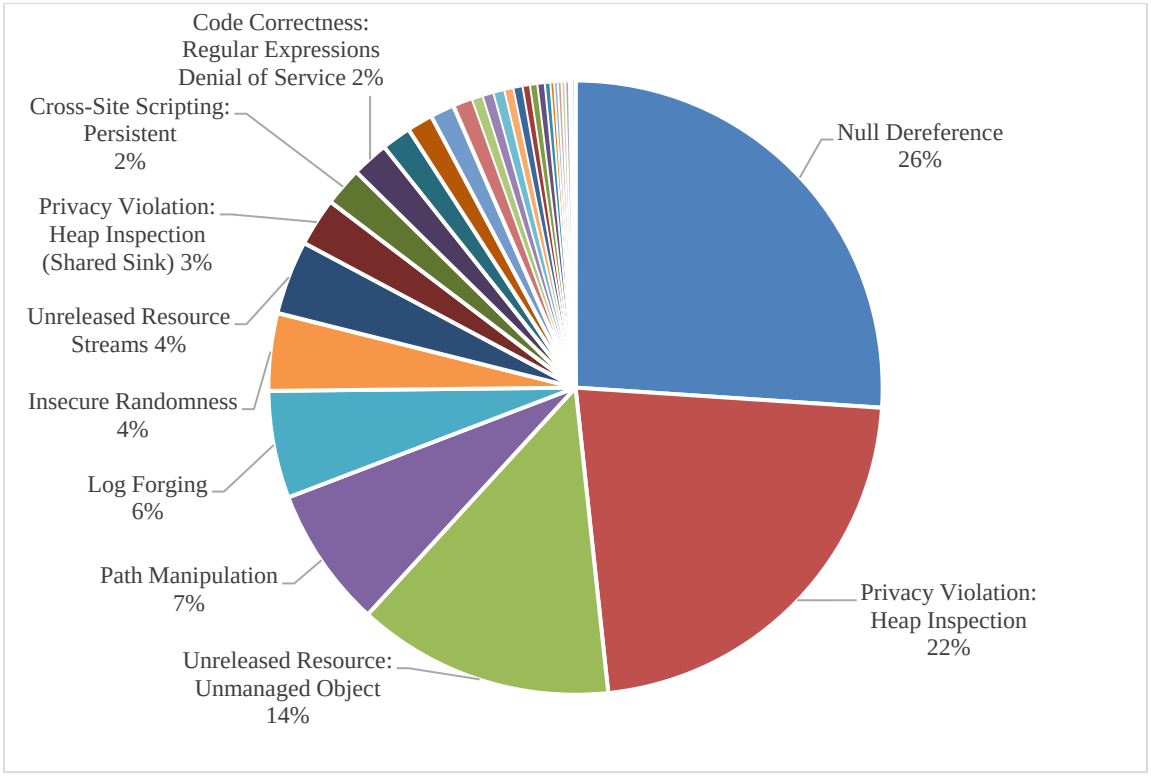


Figure 4.7: *High*-Rated True Positives Defects Detected by HP Fortify. *The top five defect categories – null dereference, privacy violation, unreleased resource, path manipulation, and log forcing – account for 75% of the defects identified as True Positives by HP Fortify.*

The Figure 4.6 identifies the defect categories and instances of the defects assessed to be True Positives (both *Critical*- and *High*-rated), and a mapping of the defect categories to the associated CWE, SANS Top 25, and OWASP Top 10.

Instances	Defect Category	CWE #	SANS	OWASP
268	Null Dereference	476		
229	Privacy Violation: Heap Inspection	226		
139	Unreleased Resource: Unmanaged Object	404		
76	Path Manipulation	22, 73	6, 13	4
58	Log Forging	117		

Instances	Defect Category	CWE #	SANS	OWASP
42	Insecure Randomness	330		
40	Unreleased Resource: Streams	404		
26	Privacy Violation: Heap Inspection(Shared Sink)	226		
21	Cross-Site Scripting: Persistent	79, 80	4	3
20	Code Correctness: Regular Expressions Denial of Service	185, 730		
16	Unsafe Native Invoke	111		
14	Access Control: Database	566	1, 6	4
13	Portability Flaw: File Separator	474		
11	Password Management: Hardcoded Password	259, 798	7, 25	2
6	Cross Site Scripting: DOM	79, 80	4	3
6	Header Manipulation: Cookies	113		
6	Unreleased Resource: Database	404		
5	Denial of Service	730		
5	Weak Security Manager Check: Overridable Method	358		
4	Dynamic Code Evaluation: Code Injection	95		
4	SQL Injection	89	1	1
4	Value Shadowing			
3	Privacy Violation	359		
2	Command Injection	77, 78	2	1
2	Missing XML Validation	112		
2	Password Management: Password in Configuration File	13, 260, 555		2, 5
2	Weak XML Schema: Unbounded Occurrences	400, 770		
1	ASP.NET Bad Practices: Non-Serializable Object Stored in Session	579		
1	Open Redirect	601	22	10
1	XPath Injection	643		

Figure 4.6: *High*-Rated True Positive Defects Detected by HP Fortify Sorted by Category and Mapped to CWE, SANS Top 25, and OWASP Top 10. *The top 10 most frequent True Positive defects detected by HP Fortify and rated High represent 93.9% of all defects detected by the tool.*

4.1.2 Coverity

Coverity was used against the source code for two of the three internet voting system vendors. The rating scale used by Coverity is: *High*, *Medium*, and *Low*.

Of the 522 defects detected by Coverity, 66 were classified as *High*, 386 as *Medium*, and 70 as *Low*, as illustrated in Figure 4.8.

[redacted]

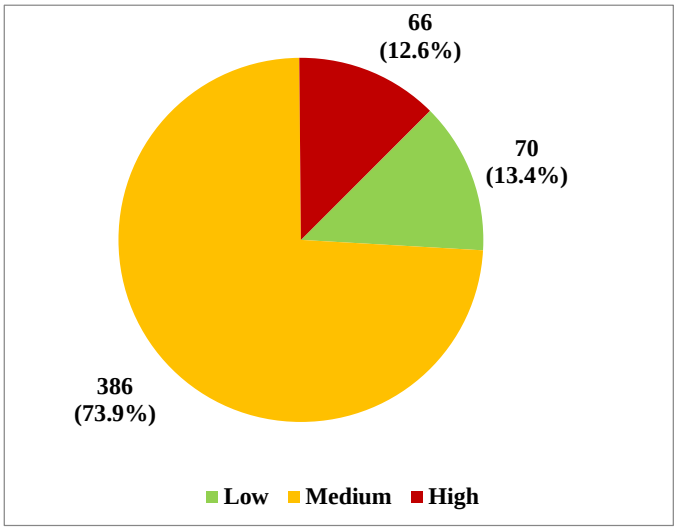


Figure 4.8: Defects Detected by Coverity by Severity Rating. *One eighth of all defects detected by Coverity were rated High, while three quarters of those were rated Medium.*

After adjudication with the vendors, of the 66 defects classified as *High*, 10.6% were judged to be True Positives, as illustrated in Figure 4.9.

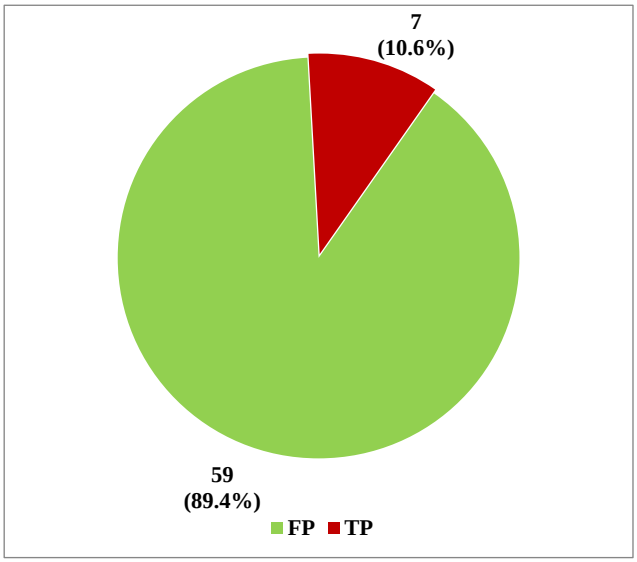


Figure 4.9: Coverity False Positive Adjudication Results. *10.6% of defects identified by Coverity were assessed as True Positives.*

While the True Positive discussion and static analysis results stated above are accurate, the reporting of the Coverity results, like the HP Fortify results, in aggregate for the vendors is distorted by the assessments done by the vendors. As discussed in section 3.1.2, the vendors all took different approaches, and levels of effort, to the assessment and adjudication process. The True Positive/False Positive analysis for Coverity when viewed by vendor is much different. One vendor assessed 85% of the Coverity *High* defects as True Positives, while a second assessed none of the defects as True Positives.

All *High*-rated True Positive defects detected by Coverity were Resource Leaks and this defect category is mapped to the associated CWE, SANS Top 25 and OWASP Top 10 in Figure 4.10.

Instances	Defect Category	CWE #	SANS	OWASP
7	Resource Leak	402		6

Figure 4.10 *High*-Rated True Positive Defects Detected by Coverity Sorted by Category and Mapped to CWE, SANS Top 25, and OWASP Top 10. *All True Positive defects detected by Coverity were resource leaks.*

4.1.3 VisualCodeGrepper – VCG

VCG, an open source static analysis tool, was used against the source code for two of the three internet voting system vendors. The rating scale used by VCG is: *High*, *Medium*, and *Low*.

Of the 262 defects detected by VCG, 30 were classified as *High*, 43 as *Medium*, and 189 as *Low*, as illustrated in Figure 4.11.

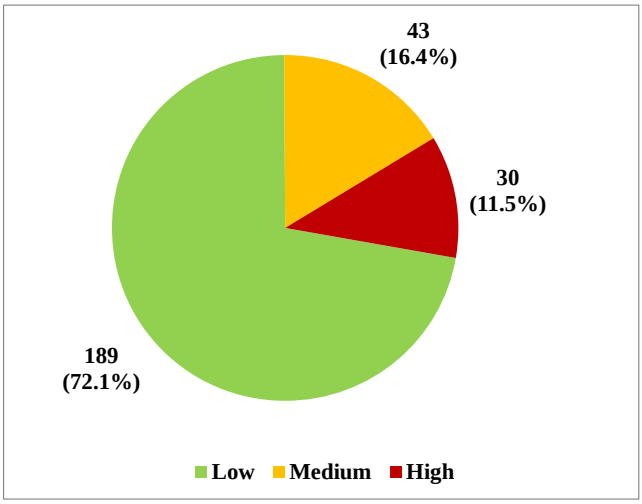


Figure 4.11: Defects Detected by VCG by Severity Rating. *One eighth of all defects detected by VCG were rated High, while nearly three quarters were rated Low.*

After adjudication with the vendors, of the 30 defects identified as *High*, 6.7% (2) were judged to be True Positives, as illustrated in Figure 4.12.

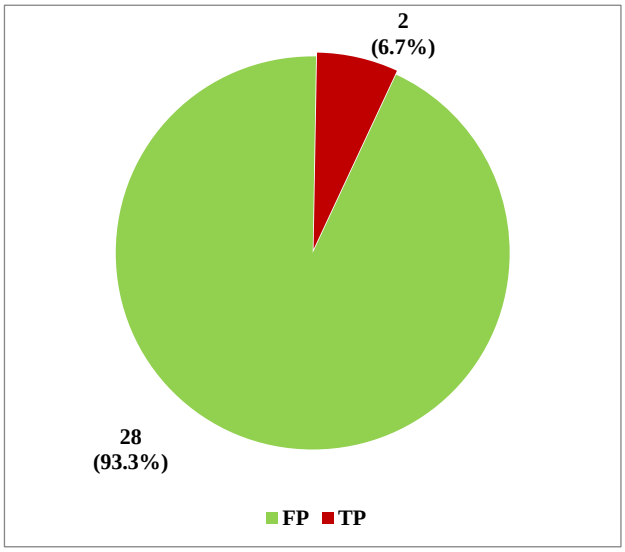


Figure 4.12: VCG False Positive Adjudication Results. *6.7% of defects identified by VCG were True Positives.*

Once again, while the True Positive discussion and static analysis results stated above are true, the reporting of the VGC results in aggregate for the vendors is distorted by the assessments done by the vendors. The True Positive/False Positive analysis for VCG when viewed by vendor is much different. One vendor assessed 100% of the VCG *High* defects as True Positives; a second assessed none of the defects as True Positives.

The 30 *High*-rated defects were classified in two categories, as shown in Figure 4.13.

Defect Category	TP	FP
SQL Injection	2	
Poor Input Validation		28

Figure 4.13: *High*-Rated Defects Detected by VCG. *The majority of all High-rated defects detected by VCG were True Positives.*

All *High*-rated True Positive defects detected by VCG were SQL Injections and this defect category is mapped to the associated CWE, SANS Top 25, and OWASP Top 10 in Figure 4.14.

Instances	Defect Category	CWE #	SANS	OWASP
2	SQL Injection	89	1	1

Figure 4.14: *High*-Rated True Positive Defects Detected by VCG Sorted by Category and Mapped to CWE, SANS Top 25, and OWASP Top 10. *SQL Injection represents all True Positive defects detected by VCG.*

4.1.4 RATS - Rough Auditing Tool for Security

RATS, an open source SA tool, is a rough auditing tool for security, originally developed by Secure Software Inc. It is a tool for scanning source code in multiple languages and flagging common security related programming errors such as buffer overflows and TOCTOU (Time Of

Check, Time Of Use) race conditions. As its name implies, the tool performs only a rough analysis of source code, and will not find every error and will also find things that are not errors.[17]

This tool was used against the source code for one internet voting vendor and its rating scale is: *High*, *Medium*, and *Low*.

Of the 611 defects detected by RATS, 30 were classified as *High*, 568 as *Medium*, and 13 as *Low*, as illustrated in Figure 4.15.

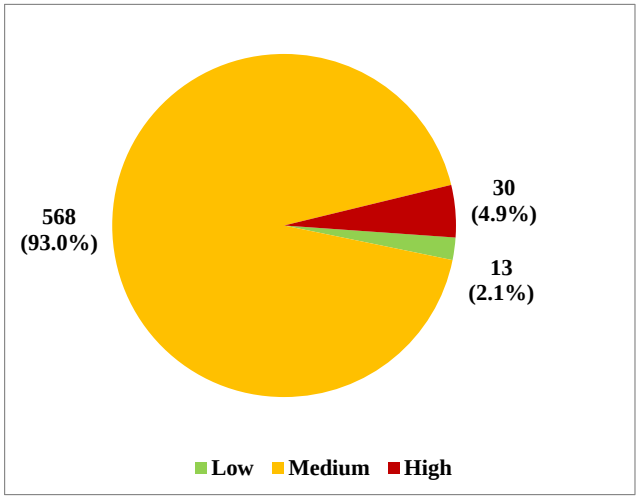


Figure 4.15: Defects Detected by RATS by Severity Rating. *The majority of the defects detected by RATS were rated Medium, with only 4.9% rated as High.*

After adjudication with the vendors, of the 30 *High*-rated defects detected by RATS, all were judged to be True Positives. This high rate of True Positives should not be taken as a reflection on the quality of the tool, but is likely a result of the vendor assessment effort. (Refer to section 3.3 Assessment and Adjudication of True/False Positives.)

[17] For more information: https://code.google.com/p/rough-auditing-tool-for-security/

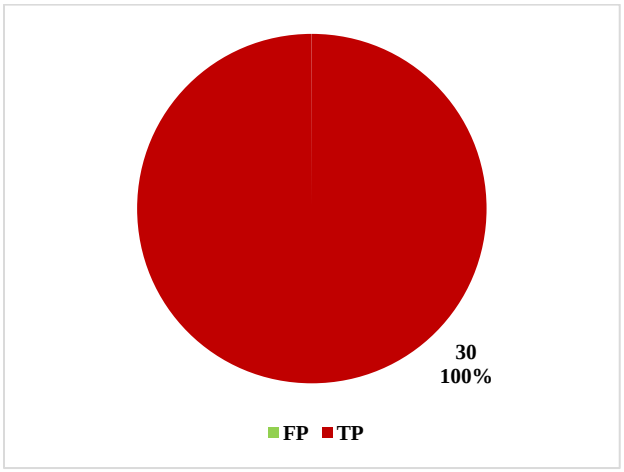


Figure 4.16: RATS False Positive Adjudication Results. *100% of defects identified by RATS were adjudicated as True Positives.*

All *High*-rated True Positive defects detected by RATS were classified in three categories mapped to the associated CWE, SANS Top 25, and OWASP Top 10 in Figure 4.17.

Instances	Defect Category	CWE #	SANS	OWASP
15	Improper Control of Generation of Code	CWE 94		1
10	Execute Code	CWE 78		6
5	Session Hijacking	CWE 384		3

Figure 4.17: *High*-Rated True Positive Defects Detected by RATS Sorted by Category and Mapped to CWE, SANS Top 25, and OWASP Top 10. *All High-rated True Positive defects detected by RATS are classified in three categories: improper control of generation of code, execute code, and session hijacking.*

4.1.5 Perl::Critic

Perl::Critic is an open source tool for the Perl programming language. Because of the lack of detail in the Perl::Critic defect report, the research team was not able to translate the Perl::Critic findings to either defect categories similar to the other static analysis tools nor to the CWEs, OWASP, or SANS taxonomy.

This tool was used against the source code for one internet voting vendor and its rating scale is: *High*, *Medium*, and *Low*.

Of the 1,607 defects found by Perl::Critic, 680 were rated as *High* and 927 as *Medium*, as illustrated in Figure 4.18.

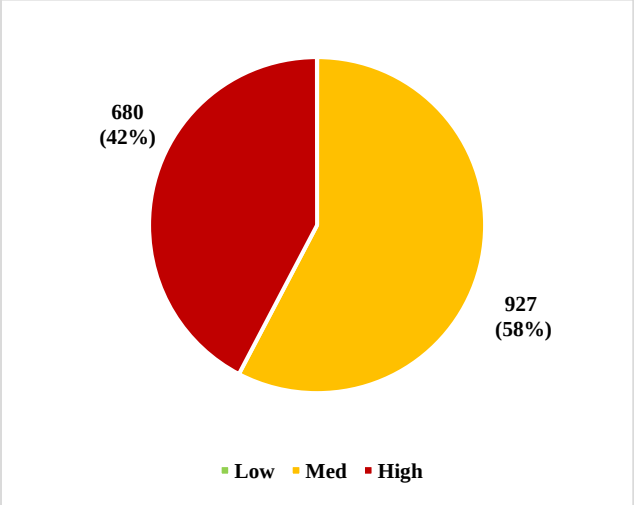


Figure 4.18: Defects Detected by Perl::Critic by Severity Rating. *42.3% of the defects detected by Perl::Critic were rated High.*

All of the 680 *High*-rated defects detected by Perl::Critic were adjudicated to be True Positives. Again, this high rate of True Positives should not be taken as a reflection on the quality of the tool, but is likely a result of the vendor assessment effort. (Refer to section 3.3 Assessment and Adjudication of True/False Positives.)

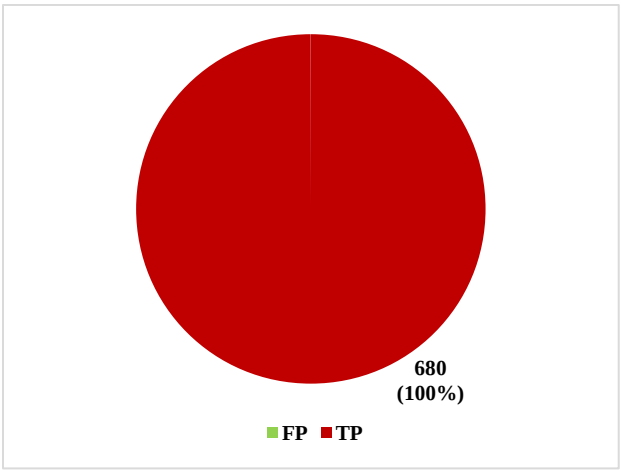


Figure 4.19: RATS False Positive Adjudication Results. *100% of defects identified by Perl::Critic were adjudicated as True Positives.*

4.2 Dynamic Test Results

The results of the dynamic scanners used in the testing are shown in Figure 4.20. The WebInspect tool tested the internet voting system vendors' source code in operation, and App Detective tested the associated database.

As previously noted, assessing a defect produced by dynamic scanning as a False Positive requires a high level of expertise of the code. For this reason, and because the primary focus of

this effort was on static analysis tools, the test results were provided to each vendor but no adjudication of the results was performed.

Severity	WebInspect	App Detective
Low	103	13
Medium	41	21
High	4	40
Total	148	74

Figure 4.20: Dynamic Testing Results. *HP WebInspect found four defects classified as High risk, and AppDetective found 40 High-risk defects.*

An important note regarding the dynamic test findings: the test environment does not reflect any additional security protocols used by the vendors in the hosting/production environment which may, or may not, neutralize or mitigate any of the defects found in CALIBRE's testing.

4.2.1 WebInspect Test Results

Across all the voting system vendors, four defects (3%) identified by WebInspect were rated as *High*, as illustrated in Figure 4.21.

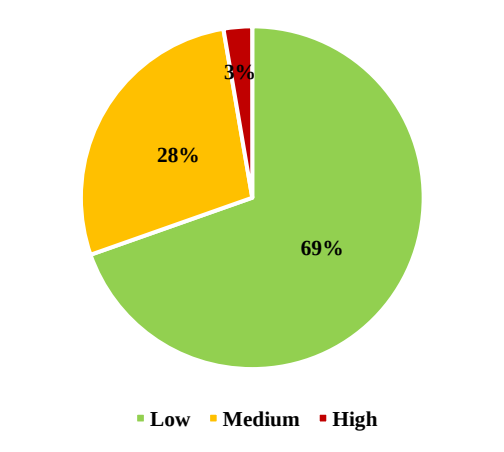


Figure 4.21: Dynamic Testing Results. *More than two thirds of all defects detected by WebInspect were rated Low with only 3% rated High.*

Figure 4.22 shows the identified defect categories, number of instances and a mapping to CWEs, SANS Top 25, and OWASP Top 10.

Instances	Defect Category	CWE #	SANS	OWASP
2	Cross-Frame Scripting	352	12	8
1	Web Server Cross-Site Scripting	79, 80, 116, 811	4	3
1	Session Fixation	384		2

Figure 4.22: *High*-Rated True Positive Defects Detected by WebInspect Sorted by Category and Mapped to CWE, SANS Top 25, and OWASP Top 10. *All High-rated True Positive defects detected by WebInspect are classified in three categories, cross-frame scripting, web server cross-site scripting, and session fixation.*

4.2.2 App Detective Test Results

54% of the defects identified by App Detective were rated as *High*, as illustrated in Figure 4.23.

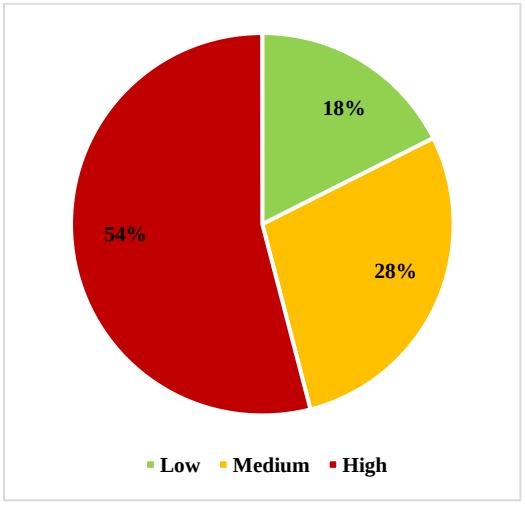


Figure 4.23: AppDet Testing Results. *54% of all defects detected by App Detective were rated High.*

Figure 4.24 lists the defect categories found by AppDet.

Defect Categories
Easily guessed root/account password
Blank root/account passwords
Password for user same as username
Latest release not installed
Anonymous user exists
Permissions on user table
MySQL Authentication bypass vulnerability
Critical Patch Update Missing

Figure 4.24: *High*-Rated Defect Categories Identified by AppDet.

As the AppDet defects relate to database issues, it is not possible to map these weaknesses to CWEs.

4.3 Analysis of Results

As the results in section 4.1 illustrate, the five static analysis tools showed great variability in detecting defects. One difficulty in reporting test results for the tools in the aggregate, across all vendors, is that not all the tools were used against all the vendors. Thus the numbers presented do not truly constitute an "apples-to-apples" comparison. Nonetheless, meaningful statements about the tools that can be made:

HP Fortify, which was used against all three vendors' source code, consistently found more defects, and more *High*-rated defects, than the other tools used against the same vendor's code. The only tool that found more potential weaknesses was Perl::Critic, a tool primarily used to test Perl coding best practices; however, the usefulness of the Perl::Critic is limited based on the detail, or lack of detail, provided in the Perl::Critic reports.

This finding is consistent with the fact that HP Fortify has a larger library of defects it tests against than the other tools. Based on the testing and analysis conducted for this report, if only one static analysis tool could be chosen, from the tools in the toolbox utilized, it would be HP Fortify.

An objective of this effort was to determine the usefulness of using multiple static SA tools on the same vendor software. In the aggregate (and starting with HP Fortify as the first tool used), as additional tools were utilized, more defects were found. This finding is consistent with the NSA CAS report. However, if only defects rated *High* were considered, the additional tools *did not* identify additional defects. The defects assessed as True Positives found by Coverity, SQL Injection (CWE 89), and VCG, Resource Leak (CWE 404), were also identified by HP Fortify.

The research team attempted to determine whether the defects found by the two tools, HP Fortify and Coverity, and HP Fortify and VCG, were identifying the same piece of suspect code (defects) in their scans. The team conducted extensive analysis on the test results as reported by the different tools. However, the team was not able to correlate defects identified by HP Fortify with either the Coverity- or VCG-identified defects due to the different levels of detail supplied by the tools' reports.

As previously stated, vendors' adjudication approaches differed drastically, resulting in the distortions of the three tools' True/False Positives rates. No conclusions about the tools True Positive/False Positive rate should be drawn from the data reported herein.

5 Pro V&V Validation Results

Pro V&V successfully utilized the Installation and User Manuals created by CALIBRE to install all SA tools and conduct the testing of the voting system vendors' code. Pro V&V replicated the CALIBRE testing results, and concurred with CALIBRE's adjudication of the False Positive analysis, thus verifying the validity of CALIBRE's methodology and protocols used.

Pro V&V utilized the documentation created by CALIBRE to install and configure each SA tool for use against the voting system vendors' software without incident. They then performed the testing according to the documentation developed by CALIBRE and replicated the test results found by CALIBRE. In one instance, using the WebInspect tool, Pro V&V's results differed from those of the CALIBRE testing. Investigation revealed that the version of WebInspect differed from the version used by Pro V&V (CALIBRE had conducted its testing several months prior to Pro V&V and an update to the software had occurred in the interim). CALIBRE received an updated version of the software and re-ran the test, producing results that matched those of Pro V&V.

Pro V&V also conducted a True Positive/False Positive review of the assessments submitted by the vendors. In all cases that the vendor assessed a defect, Pro V&V concurred with the vendor finding. For the vendor that performed a random sampling of the defects in their assessment process, Pro V&V reviewed additional defects that found additional False Positives.

Pro V&V's complete report is found in Appendix B.

6 Conclusions and Recommendations

The results presented in this report document that existing software assurance tools provide a viable means of identifying potential security and coding best practices weaknesses of existing internet voting system vendors' software. This report documents a successful, and verified, methodology for conducting SA tool testing of voting system vendor software. The report also identifies challenges encountered and identifies resolutions that led to successful testing.

This report examined the effectiveness of a tailored suite of software assurance tools, both static and dynamic, against the code of three EAC-registered internet voting systems, and the ability to optimize the tools in order to reduce the number of False Positives detected during the scans. While this effort was not intended to assess, nor draw any opinions or conclusions on the security posture of the voting systems, all vendors were identified to possess defects in their code. This report describes:

- The project's background, including the development, and subsequent modification of the toolbox of analysis tools;
- The protocol and methodology used to test the vendor software by the analysis tools and a discussion of tool optimization;
- The results of the testing conducted against the three vendor's source code by tool and an analysis of the results; and
- The validation of the protocol, methodology, and results by Pro V&V.

The following are conclusions based on the testing effort and the analysis of the test results for this project:

- If commercial (non-open source) tools are being utilized (e.g., Parasoft), it is imperative to understand the programming language(s) and other technical details utilized by the voting system vendors prior to tool acquisition.
- Using multiple static code analysis tools increased the number of potential defects identified in the source code for all severity ratings (*High, Medium, and Low*).
- However, if one considers the use of HP Fortify as the primary static analysis tool, the additional tools utilized for this analysis did not increase the number of True Positive *High* severity defects identified.
- For the C# and Java coding languages, HP Fortify identified the vast majority of potential defects. The open source tools used were of marginal value.
- Of the tools that were utilized for this analysis, the commercial source tools (HP Fortify and Coverity) provided varying levels of customization/optimization that the open source tools often did not.
- Customizing static analysis tools to reduce/eliminate False Positives can be done in the development phase of coding (in the IDE); however when the tools are used for the current type of analysis, post development, all defects must be examined to determine a True/False Positive finding.

In the broader scope, this project provides FVAP with a methodology to assess the software assurance posture of potential vendors for future FVAP pilot/demonstration projects, and identifies several useful SA tools for such an assessment. This project can serve as a prototype for the EAC to integrate automated SA tools into the EAC certification process and highlights, for voting system vendors, the need for SA tool integration into their development process. Additionally, this project can serve as a building block for NIST as they continue assurance tool testing in the voting environment and act as a first step in developing a library of defects (CWEs) specifically affecting voting systems.

Going forward, the following are recommendations for either additional research with regard to the use of software assurance tools, other areas of research and analysis that may be useful regarding testing tools, and/or additional security analysis of voting system vendors:

- Investigate the use of two or more robust static analysis tools (e.g., HP Fortify and Parasoft) in True/False Positive identification; using two or more tools with similar defect coverage could help with the False Positive adjudication by rapidly identifying defects found by multiple tools.
- Examine the use of multiple open source tools to provide the same level of analysis as a single commercial tool.
- Research that defines the criteria that SA tools use to map their findings to the CWEs.
- Conduct analysis on the defects found by SA tools compared to a list of potentially applicable CWEs, to include:
    - Identifying CWEs found by SA tools
    - Identify non-CWE enumerated defects and the associated criteria
    - Development of definitions that tool manufacturers can use to enhance their rules for mapping to the CWEs
    - Identify CWEs that need to be refined to facilitate the mapping process
    - Recommended changes to the CWEs to facilitate a higher percent of the tool findings mapping to the CWEs
- Conduct analysis of additional SA tools on vendor code, identifying critical and high defects found for each tool.
- Creation of knowledge base of typical defects that would cover:
    - How to manually test a defect
    - What makes the defect a True or False Positive (i.e., what to look for)
- Perform an attack analysis on each vendor site to provide more details on how a potential attacker may try to access the system.
- Compare and contrast dynamic scanners on operational systems with known defects and scan with dynamic toolkit.
- Conduct a cost/benefit analysis of the use of automated tools versus manual code review in the testing and certification process.

- Analyze incorporating the requirement for a toolkit of assurance tools as part of the voting system testing and certification process in lieu of a complete line-by-line review of code.
- Conduct analysis on the potential impact of identified defects in vendor code on voting systems and related voting results.
- Develop an overall, comprehensive toolkit balanced with commercial and open source static and dynamic tools that can meet a wide variety of operational environments and software coding languages.
- Examine strategies to make the aforementioned toolkit a cost-effective package that could be used by voting system vendors, as a part of the standards NIST certifies and the EAC incorporates into its internet voting standards, to provide election officials and voters a greater level of confidence.

While existing SA tools provide a viable means of identifying potential security and coding best practice weaknesses of existing internet voting system vendor's software, the development of a cost effective suite of tools for use by multiple constituencies, from software developers to testing laboratories, holds the potential for delivering voting systems that have been designed from the ground up, and verified through testing, to be safe and secure.

Appendix A: Vendor Questionnaire

Questions regarding your voting system:

- Which operating system do you use? (e.g., Windows 2008 R2, Linux)
- What language(s) is your source code in? (e.g., Java, C#, C\C++)
- Which integrated development environment do you use? (e.g., Visual Studio, Eclipse)
- Does your system use any third party compilers? (e.g., Maven, Apache Ant)
- Do you currently use any static/dynamic code analyzers outside the CALIBRE's suite of tools? (e.g., Redgate ANTS, Checkmarx)
- Which web server/servlet do you use? (e.g., Apache Tomcat)

The following items are necessary to adequately test the voting system:

Description	**Point of Contact**	**Date Received**
Replica of your development system with any IDEs and third party compilers		
Replica of your operational system with database and an account with admin privileges		
Documentation for both the setup and compilation of your source code		
A user's guide on how to navigate your website		
Documentation on database (e.g. account credentials)		
A list of voters with credentials to use with your voting application		

Additional Comments:

Appendix B: Pro V&V Report

PRO V&V

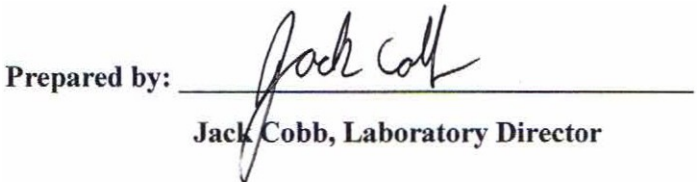
TEST REPORT

FOR

PERFORMING VERIFICATION AND VALIDATION

ON THE

OPTIMIZATION OF SOFTWARE ASSURANCE TOOLS



May 12, 2014

Pro V&V, Inc.
700 Boulevard South, Suite 102
Huntsville, AL 35803
256-713-1111

TABLE OF CONTENTS

[redacted]

1.0 INTRODUCTION

This document presents the test processes and procedures followed by Pro V&V, Inc., (hereafter referred to as Pro V&V) when performing verification and validation on the optimization of software assurance tools. Pro V&V performed this effort with the intent of providing a third party independent opinion for the optimization and documentation created and performed by CALIBRE for each of the submitted internet voting software applications. Pro V&V carried out all activities performed as part of this test engagement in such a way as to meet the requirements of National Institute of Standards and Technology (NIST) handbooks 150-2006 and 150:22-2008 and to satisfy the needs of CALIBRE, the regulatory authorities, or organizations providing recognition.

1.1 Pro V&V

Pro V&V is accredited as a National Voluntary Laboratory Accreditation Program (NVLAP) testing laboratory by the National Institute of Standards and Technology (NIST). This NVLAP accreditation incorporates the requirements of ISO/IEC 17025 and incorporates testing to the Voting System Standards (VSS), including the Help America Vote Act (HAVA) requirements and the Voluntary Voting Systems Guidelines (VVSG), for core test methods involving: Voting System Testing, Technical Data Package review, Physical Configuration Audit, Source Code Review, Witnessed Build and System Installation Testing, Functional Configuration Audit, System Integration testing, and Telecommunication and Security testing.

Additionally, Pro V&V has been audited by the Election Assistance Commission (EAC) and successfully met all requirements for the EAC Voting System Test Laboratory (VSTL) accreditation.

1.2 Background

Software assurance (SA) is generally defined as the level of confidence that software is free from vulnerabilities, weaknesses, or flaws and that the deployed software functions as intended. The Federal Voting Assistance Program (FVAP) commissioned CALIBRE to optimize a software assurance toolkit for each of the registered internet voting manufacturers. CALIBRE selected several tools that are approved by the Defense Information Systems Agency (DISA) and the National Information Assurance Partnership (NIAP).

During the software assurance research project, CALIBRE recommended three static code analyzers that were identified in the National Security Agency's Center for Assured Software report issued in December 2011 *(NIST SAMATE (Software Assurance Metrics and Tool Evaluation: Static Analysis Tool Study Methodology)).* This report stated that that the use of up to three different static source code analysis tools may provide up to a 61% true positive rate. The goal of the CALIBRE optimization project was to meet or exceed this rate.

SA tools can be automated, semi-automated, or manual software tools which test for flaws within the software coding environment. SA tools can be categorized and evaluated based on their analysis, methodology, and purpose. Previous research determined that a software assurance toolkit customized to each voting system manufacturer could possibly decrease software vulnerabilities by identifying critical vulnerabilities.

The table below presents a list of the static source code analysis tools, dynamic web application scanner, and dynamic database scanner that were selected as part of the software assurance tool kit.

Table I: Software Assurance Toolkit

Dynamic Software Assurance Tools		Static Software Assurance Tools				
Database Scanner	Web Application Scanner	Source Code Security Analyzer				
App Detective	HP WebInspect	RATS	Perl Critic	HP Fortify	Coverity	VisualCode Grepper

1.3 Scope

The "Target of Evaluation" (TOE) for the test engagement was the optimization performed by CALIBRE for each of the SA tools by manufacturer and the documentation used to perform the optimization. Pro V&V performed verification on the documentation provided. Pro V&V also provided verification that the false positives reported by the EAC registered internet voting manufacturers were false positives.

This test engagement included source code in a development environment and the software application in an operating environment for the three EAC registered internet voting systems manufacturers. This test engagement also included a selected toolkit for each of the EAC registered internet voting systems manufacturers.

The tools recommended by CALIBRE include all needed scanning tools to allow the use of three different static analysis tools. The documented tools in Table I were researched and selected based on a research project performed by CALIBRE. According to the technical information provided by the three EAC registered internet voting systems manufacturers, these tools were selected because of languages used, system architectures and third party software.

1.4 References

- National Institute of Standards and Technology (NIST) Handbook 150, 2006 Edition, National Voluntary Laboratory Accreditation Program procedures and General Requirements
- NIST Handbook 150-22, 2008 Edition, National Voluntary Laboratory Accreditation Program (NVLAP) Voting Systems Testing

1.5 Terms and Abbreviations

CM – Configuration Management

COTS – Commercial off-the-Shelf Software/Hardware

DISA – Defense Information Systems Agency

EAC – Election Assistance Commission

FVAP – Federal Voting Assistance Program

NIAP – National Information Assurance Partnership

NIST – National Institute of Standards and Technology

NVLAP – National Volunteer Laboratory Accreditation Program

SA – Software Assurance

SAMATE – Software Assurance Metrics and Tool Evaluation

VLAN – Virtual Local Area Network

VM – Virtual Machine

VSTL – Voting Systems Test Laboratory

1.6 Testing Responsibilities

All testing was conducted under the guidance of Pro V&V by personnel verified by Pro V&V to be qualified to perform the testing. No other personnel or subcontractors were used in this test engagement.

1.7 Quality Assurance

Pro V&V assumed full responsibility for quality control throughout the test engagement. The Quality Assurance Manager was responsible for employing quality control methods in order to provide effective, high quality services that achieved or exceeded the acceptable quality level performance standards set by the NVLAP. To accomplish this, Pro V&V has developed a Quality Management System (QMS) that meets the guidelines for NIST and EAC accreditation.

2.0 MATERIALS REQUIRED FOR TESTING

This section contains the detailed descriptions of items utilized during this test engagement including all software, hardware, peripherals, both proprietary and COTS, and any test support equipment or materials necessary for test performance.

This test engagement was conducted remotely using virtual operating environments. The virtual operating environment was documented to include both logical environment and the physical environment.

2.1 Software

The virtual environment was explicitly documented before test execution to detail the virtual hardware and software for each environment. This documentation included the logical PC, laptop, or server and also defined the manufacturer, processor, memory, hard drive capacity, operating system, and any COTS supporting applications such as Microsoft Office or Adobe Acrobat Reader.

For each internet voting system, both a development environment and an operational environment was required. Before test execution, a detailed inventory of all software required to support the test engagement was documented. This included versions, and other identifying components. The software components used during testing are detailed in Table 2.1, below:

Table 2.1 Software

Component Name	Version	Description
Real VNC Viewer	4.1.3	Linux virtual viewer
DigitalVolanco Hash Tool	1.1.0.0	Digital signature tool
Oracle VM VirtualBox Manager	4.3.10r93012	Virtual machine software

In addition to the software components for test execution, the software assurance toolkits were documented in a manner as to ensure the traceability and reproducibility of the test execution. Before test execution, a detailed inventory of all SA tools was documented. The SA toolkits tested are detailed in Table 2.2, below:

Table 2.2 Software Assurance Toolkits

Tool Name	Version
HP Fortify Audit Workbench	3.80.0060
HP Fortify SCA	5.15.0.0060
Coverity	6.6.1
VisualCodeGrepper	1.5.1.1
HP RATS (Rough Audit Tool	2.3
Perl::Critic	1.121
HP WebInspect	10.1.177.0
AppDetectivePro	8.2

2.2 Equipment

The physical hardware that hosts the virtual environments was documented before test execution. This documentation included manufacturer, processor, memory, hard drive capacity, operating system, and any COTS supporting applications such as Microsoft Office or Adobe Acrobat Reader.

The details of the VM Host Physical Server/Server Array hardware are listed below:

Table 2.3 VM Host Physical Server/Server Array Hardware

Make	*Various*
Model	*Various*

Processor	Intel Xeon®
Memory	192 GB
Operating System	VMWare ESXi 5
Software and Version	VMWare ESXi 5

The details of the VM Client Machine hardware are listed below:

Table 2.4 VM Client Machine Hardware

Computer Name	CALFVAPSCAN2.calibresys.com
Domain	calibresys.com
IP Address	10.200.200.9
Services	SNMP, SNMP WMI Provider, Telnet Client
Processor	Intel Xeon® X560 @ 2.67GHz
Memory	8.00 GB
Storage	300GB
Operating System	Windows Server 2008 R2 Datacenter (64-bit product ID55041-242-0667907-84014)
Software and Version	VMware Tools for Windows Version 9.0.5 build-1065307

Computer Name	CALFVAPSCAN3.calibresys.com
Domain	calibresys.com
IP Address	10.200.200.10
Services	SNMP, SNMP WMI Provider, Telnet Client
Processor	Intel Xeon® X560 @ 2.67GHz
Memory	8.00 GB
Storage	300GB
Operating System	Windows Server 2008 R2 Datacenter (64-bit product ID55041-242-0667907-84014)
Software and Version	VMware Tools for Windows Version 9.0.5 build-1065307

2.3 Optimization Documentation

CALIBRE provided documentation in the form of an installation guide and users guide for each of the three registered EAC internet voting manufacturers. Pro V&V has the specific name and revision numbers on file. Because the document specific name will identify the voting systems manufacturer they will not be released with this report.

3.0 TEST SPECIFICATIONS

3.1 Static Code Analyzer

For the static code analyzers, an installation for each software assurance tool was performed. Once each tool was in a known default state, the build environment was created by installing all compilers, supporting software and source code for a manufacturer. After the creation of the build environment, a scan was performed using each of the selected tools. This scan was considered the baseline scan for each tool.

An analysis was then performed comparing the baseline scan reported by CALIBRE and the baseline scan performed by Pro V&V. In all scans the findings were able to be reproduced. CALIBRE submitted finding of false positives for each tool per manufacturer. These findings were then analyzed by Pro V&V for correctness. The analysis for two of the manufacturers did not contain enough detail to make a determination on the degree to which the tools reported false positives. For these two manufacturers Pro V&V performed an analysis of the source code against selected findings and determined that a more detailed analysis would have to be performed to create a statistical measurement.

3.2 Dynamic Analysis Tools

For dynamic analysis tools, each scanner was installed on a single virtual machine. The virtual machine had access to the operating environment provided by the EAC internet voting systems manufacturer in order to perform the scan remotely. From this virtual machine, a scan was performed of the internet voting system that was considered the baseline scan.

An analysis was then performed comparing the baseline scan reported by CALIBRE and the baseline scan performed by Pro V&V. The results of these two scans were not able to be reproduced for the web application scanner. It was determined that the scanner had been updated in the time period between scans. CALIBRE updated WebInspect and performed another scan and the results were reproduced.

3.3 Test Environment Architecture

The test environment architecture used for this test engagement consisted of four separate virtual machines. A build machine for each manufacture and an operational scanning machine that can be pointed to each operational environment submitted by each manufacturer were required. The diagram below provides the architectural overview of this configuration.

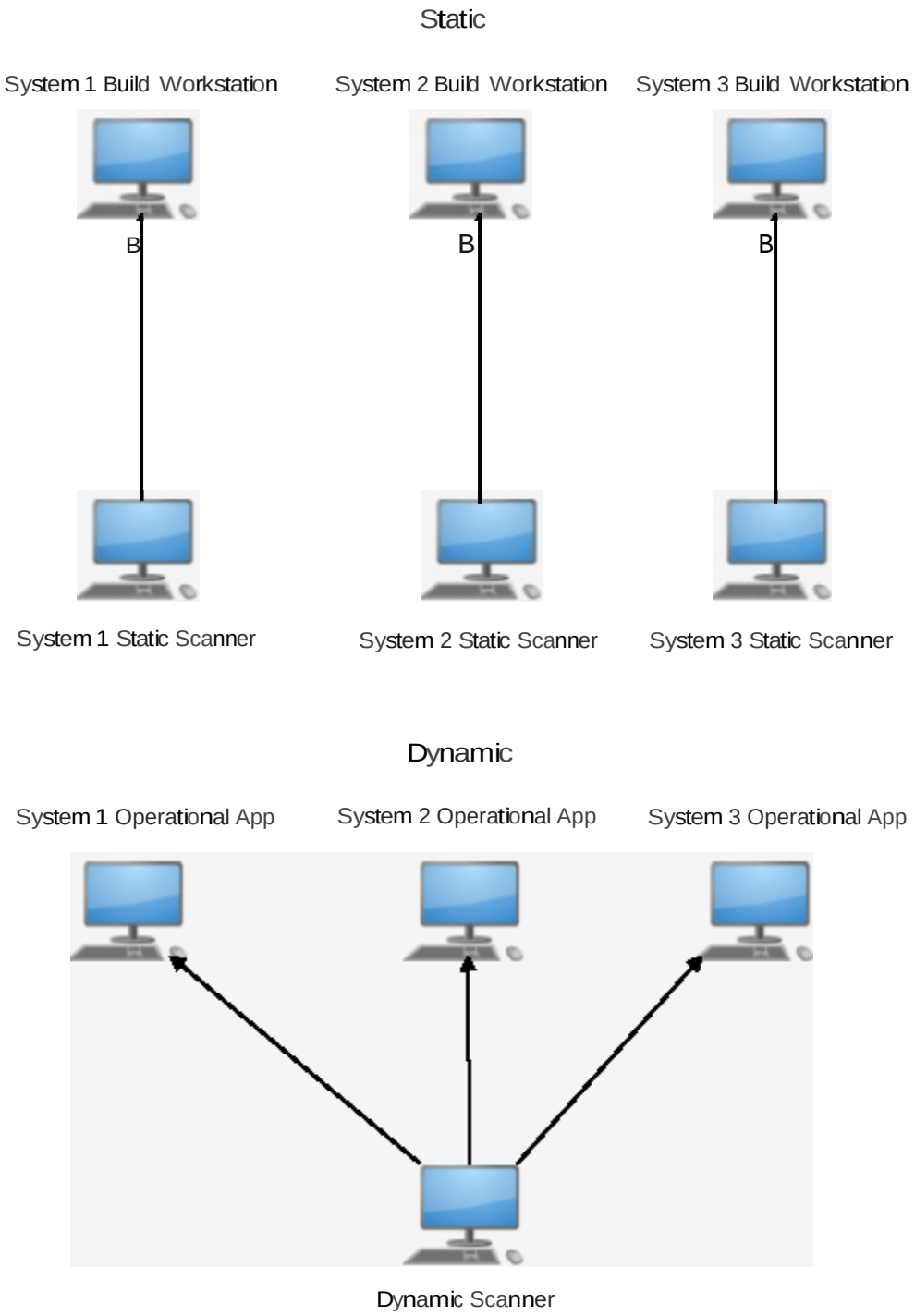


Figure 1: Test Environment Diagram

4.0 TEST DATA

4.1 Test Data Recording

All test data produced during this project is the property of CALIBRE. The output test data from the software assurance tool scanners was collected and stored in an appropriate manner as to allow for data analysis.

4.2 Test Data Criteria and Reduction

All test results were evaluated against the expected results set forth in the test cases. For static code analyzers, the comparative analysis investigated the data output differences to ensure the optimization removed false positives and did not alter the ability of the SA tools to identify true negatives.

5.0 TEST PROCEDURE AND CONDITIONS

5.1 Facility Requirements

The test engagement was performed remotely utilizing virtualized hardware resources. The physical hardware was located at the CALIBRE facility in Alexandria, Virginia. During the test engagement, control of the test items and the test environment was maintained at all times.

5.2 Test Setup

A baseline configuration was established for the virtual environment for both the development environment and the operational environment. The creation of the baseline environment performed by CALIBRE was documented. Each baseline configuration included a virtual machine, the operating system, supporting software applications and the SA tools for that environment. For the development environment, all static code analyzers were installed in an out-of-the-box default configuration. For the operational environment, all dynamic scanners were also installed in an out-of-the-box default configuration.

The baseline configurations for each of the three EAC registered internet voting systems were used. The virtual environments were hosted on the hardware documented in Section 2 *Materials Required for Testing* of this Test Report. Once the developmental baseline configuration was successfully hosted, source code and build environment were installed on the development baseline configuration. The operational environment was then used to examine the operational application using the dynamic scanners.

For test setup, the physical hardware used to host the virtual environments was configured. This hardware consisted of the VM Host Physical Server/Server Array identified in Section 2 *Equipment* of this Test Report. The logical test environment was then established through a VLAN containing a set of virtual machines. The VLAN was used for hosting voting systems, development environments, and source code scanners.

There are thirteen boxes on the VLAN that are identified as follows:

Table 5.1 VLAN Box Identification

Box	Identification
10.200.200.1	*NA*
10.200.200.2	*NA*
10.200.200.3	*NA*
10.200.200.5	CALFVAPSCAN1
10.200.200.6	Manufacturer Development
10.200.200.7	Manufacturer Operational
10.200.200.8	Manufacturer Development
10.200.200.9	CALFVAPSCAN2
10.200.200.10	CALFVAPSCAN3
10.200.200.11	Manufacturer Operational
10.200.200.12	Manufacturer Operational
10.200.200.13	*NA*
10.200.200.14	Pro V&V's Manufacturer Development

5.3 Test Sequence

Scanning of systems occurred in two phases. During Phase 1, the static code analyzer toolkits were used to perform all scanners. After the baseline was established in Phase 1, Phase 2 was accomplished by performing scans using dynamic toolkits.

CALIBRE identified distinct steps for each of the test phases. These steps are presented in the documentation provided to Pro V&V by CALIBRE. The steps to install and configure each tool per EAC registered manufacturer were provided.

Verification and validation was performed, on each document. The purpose of the verification and validation was to provide a third party independent opinion for the optimization of SA tools performed by CALIBRE. This engagement included COTS software assurance tools, custom configurations of the SA tools per internet voting system manufacturer, and documentation created by CALIBRE for each custom configuration. The target of evaluation for engagement was not the source code for internet voting, but the optimization and documentation created and performed by CALIBRE.

6.0 RESULTS

6.1 Static Code Analyzer

HP Fortify SCA/Audit Workbench

The optimization documentation submitted for HP Fortify Audit Workbench by CALIBRE consisted of an installation guide and user's guide to be utilized by users possessing limited technical expertise to install and use the submitted product. This documentation was evaluated and was determined to be sufficient to meet the requirements. After Pro V&V installed HP Fortify Audit Workbench according to the installation guide and performed scans per the

submitted user's guide, any discrepancies noted in the evaluation were reported to CALIBRE and were successfully remediated prior to the end of the test campaign.

Coverity

The optimization documentation submitted for Coverity by CALIBRE consisted of an installation guide and user's guide to be utilized by users possessing limited technical expertise to install and use the submitted product. This documentation was evaluated and was determined to be sufficient to meet the requirements. After Pro V&V installed Coverity according to the installation guide and performed scans per the submitted user's guide, any discrepancies noted in the evaluation were reported to CALIBRE and were successfully remediated prior to the end of the test campaign.

VisualCodeGrepper (VCG)

The optimization documentation submitted for VCG by CALIBRE consisted of an installation guide and user's guide to be utilized by users possessing limited technical expertise to install and use the submitted product. This documentation was evaluated and was determined to be sufficient to meet the requirements. After Pro V&V installed VCG according to the installation guide and performed scans per the submitted user's guide, any discrepancies noted in the evaluation were reported to CALIBRE and were successfully remediated prior to the end of the test campaign.

RATS

The optimization documentation submitted for RATS by CALIBRE consisted of an installation guide and user's guide to be utilized by users possessing limited technical expertise to install and use the submitted product. This documentation was evaluated and was determined to be sufficient to meet the requirements. After Pro V&V installed RATS according to the installation guide and performed scans per the submitted user's guide, any discrepancies noted in the evaluation were reported to CALIBRE and were successfully remediated prior to the end of the test campaign.

Perl::Critic

The optimization documentation submitted for Perl::Critic by CALIBRE consisted of an installation guide and user's guide to be utilized by users possessing limited technical expertise to install and use the submitted product. This documentation was evaluated and was determined to be sufficient to meet the requirements. After Pro V&V installed Perl::Critic according to the installation guide and performed scans per the submitted user's guide, any discrepancies noted in the evaluation were reported to CALIBRE and were successfully remediated prior to the end of the test campaign.

6.2 Dynamic Analysis Tools

HP WebInspect

The optimization documentation submitted for WebInspect by CALIBRE consisted of an installation guide and user's guide to be utilized by users possessing limited technical expertise to install and use the submitted product. This documentation was evaluated and was determined to be sufficient to meet the requirements. After Pro V&V installed WebInspect according to the installation guide and performed scans per the submitted user's guide, any discrepancies noted in the evaluation were reported to CALIBRE and were successfully remediated prior to the end of the test campaign.

AppDetectivePro

The optimization documentation submitted for AppDetectivePro by CALIBRE consisted of an installation guide and user's guide to be utilized by users possessing limited technical expertise to install and use the submitted product. This documentation was evaluated and was determined to be sufficient to meet the requirements. Pro V&V could not install AppDetectivePro in the test environment. Pro V&V worked with technical support from the manufacturer and network engineers from CALIBRE for 2 months to try and resolve this issue. It was determined that a policy would not allow AppDectiveScan service to run. This caused the server to crash with a Kernel Power failure message. The duration of this project did not allow for the discrepancy to be resolved. Pro V&V verified that installation guide and reviewed the user's guide, but because this discrepancy could not be resolved the result obtained by the scanner could not be verified.

6.3 False Positive Analysis

CALIBRE submitted the results from the EAC registered internet voting systems manufacturers' analysis for false positives. Pro V&V performed an analysis of these results. The analysis performed by each manufacturer was to varying levels. One manufacturer performed a detailed analysis. For this manufacturer, Pro V&V verified each item identified by the manufacturer as and false positive. Pro V&V concurred on every item. Another manufacturer performed a sample analysis selected a few items for a given category. For this manufacturer, Pro V&V verified each item identified by the manufacturer as false positives and selected additional items. Pro V&V concurred on every item, but was able to identify more false positives than the manufacturer identified. The last manufacturer performed a very high level analysis. Because the results report were at such a high level Pro V&V could not definitively determine whether the findings were true or false positives.