ES&S Unity 3.0.1.1
Source Code Review

for

California Secretary of State
Office of Voting Systems Technology Assessment

February 15th, 2008

The source code review for the ES&S Unity 3.0.1.1 voting system was conducted by:

atsec information security corporation
9130 Jollyville Road, Suite 260
Austin, TX 78759
www.atsec.com

for California Secretary of State Debra Bowen under contract with Freeman, Craft, & McGregor Group (FCMG). atsec is accredited as a Common Criteria Evaluation Lab, a Cryptographic Module Test Lab (FIPS 140-2), and provides other computer security testing services for commercial companies.

General description of Equipment Under Test (EUT)

The Unity 3.0.1.1 (Unity) system, marketed by Election Systems & Software (ES&S), consists of the following components:

- Model 100 optical scan precinct ballot counter (M100)
- Model 650 optical scan central ballot counter (M650)
- AutoMARK Voter Assist Terminal (VAT)
- Election Data Manager (EDM)
- ES&S Image Manager (ESSIM)
- Hardware Program Manager (HPM)
- Election Reporting Manager (ERM)
- Audit Manager (AM)
- AutoMARK Information Management System (AIMS)

The Election Systems and Software (ES&S) Unity 3.0.1.1 voting system is a complete electronic voting system designed and implemented to create and manage elections. The Unity system programs run on Microsoft Windows platforms and are used with ES&S ballot marking and tabulating devices. Ballot marking for voters with disabilities is performed by the AutoMARK VAT. Tabulation is performed by the M100 and M650, which are based on embedded operating systems.

The Election Data Manager (EDM) is used to build a database of all jurisdiction and election data. Contests, candidates, precinct data and other election information can be entered or imported into the database. This information is then used to create election-specific files.

The ES&S Image Manager (ESSIM) is used to design and typeset the ballots. Absentee and early voting ballots can also be printed through the ESSIM.

Once the ballots are prepared, the Hardware Program Manager (HPM) is used to program the various memory transfer devices that are used to configure and program the ballot tabulating devices (M100 and M650).

After the ballots have been typeset and printed and the tabulators have been programmed, voters mark the printed ballots, which are then tabulated. To enter a vote, the ballot card is marked in the appropriate oval to record a vote for a candidate or Yes/No choice in each contest.

Voters with disabilities may use the AutoMARK Voter Assist Terminal (VAT). In preparation for use, the AutoMARK VAT is set up and configured using the AutoMARK Information Management System (AIMS) system. Once configured, the AutoMARK VAT will provide ballot choices and instructions for the voter. Selections are displayed in large text print on the touch screen monitor, as well as read by the audio system in the language choice selected by the voter. Voters enter their selections by touching buttons on the screen, pressing keys on the keypad, or using an ADA device (a sip/puff switch or foot pedal). Once the voter's selections are made and confirmed, the ballot is marked for the voter.

Local ballot tabulation is handled by the M100 Ballot Tabulator, which is a precinct level ballot verification and tabulation system. The ballot is read by the M100, and any irregularities are immediately reported on an LCD screen. At the end of voting, a paper tape can be printed with the election accounting. On shutdown, the memory card with all of the voting information is removed from the M100 and is returned to election central.

The M650 tabulator is used for central tabulation and vote-by-mail ballots. Results can be printed, as on the M100, and can be stored on a zip-disk.

Once the election is closed, the Election Reporting Manager (ERM) is used to consolidate and report the system-wide tabulation.

The Audit Manager (AM) is a user management tool that allows an election administrator to add users and manage access to the Unity programs. A log of activities performed in Election Data Manager (EDM) and ES&S Image Manager (ESSIM) is created by the Audit Manager. Each action performed in EDM and ESSIM is logged in Audit Manager.

Scope Limitations

The purpose of this testing was to identify and document vulnerabilities and potential vulnerabilities in the Unity voting system by examination of the product documentation and the product source code. The emphasis was on assessing the security and integrity of the system, and in particular, on identifying any security vulnerabilities that could be exploited to alter vote recording, vote results, or critical election data such as audit logs, or to conduct a "denial of service" attack on the voting system.

For the purpose of the test, the test team was asked to consider the following classes of attackers:

- **Voter**: Usually has low knowledge of the voting system machine design and configuration. Some may have more advanced knowledge. May carry out attacks designed by others. They have access to the machine for less than one day.
- **Poll worker**: Usually has a low knowledge of the voting machine design and configuration. Some may have more advanced knowledge. May carry out attacks designed by others. They have access to the machine for less than one day.
- **Election official insider**: Has a wide range of knowledge of the voting machine design and configuration. They may have restricted access for long periods of time. Their designated activities include:
    - Set up and pre-election procedures.
    - Election operation.
    - Post election processing of results
- **Storage Personnel**: Election or vendor employees that perform pre- and post-election maintenance and have access to the stored machines. Activities also include archiving and storage of the physical machines.
- **Vendor insider**: Has a great knowledge of the voting system design and configuration. They have unlimited access to the machine before it is delivered to the purchaser and, thereafter, may have unrestricted access when performing warranty and maintenance service and when providing election administration services.

The team was not limited to these attackers, and their direction included direction from Resolution # 17-05 of the Technical Guidelines Development Committee (hereafter "TGDC") of the U.S. Election Assistance Commission, adopted at the TGDC plenary meeting on January 18 and 19, 2005, which calls for:

> ". . . testing of voting systems that includes a significant amount of open-ended research for vulnerabilities by an analysis team supplied with complete source code and system documentation and operational voting system hardware. The vulnerabilities sought should not exclude those involving collusion between multiple parties (including vendor insiders) and should not exclude those involving adversaries with significant financial and technical resources."

The specific task, as presented in the Statement of Work for the Source Code Review Team, was:

- *Focus on potential vulnerabilities and related issues (code quality and standards compliance), considering that an exploitable issue in a component that is not in itself security relevant could be used to subvert more critical data. This is an issue whenever the architecture of the system does not provide strong separation of the components.*
- *Adherence to the applicable standards in sections: 4 of Volume I, 7 of Volume I, and 5 of Volume II of the 2002 Voluntary Voting System Standards.*
- *Adherence to other applicable coding format conventions and standards including best practices for the coding language used, and any IEEE, NIST, ISO or NSA standards or guidelines which the reviewers find reasonably applicable.*
- *Analysis of the program logic and branching structure.*
- *Search for exposures to commonly exploited vulnerabilities, such as buffer overflows, integer overflow, inappropriate casting or arithmetic.*
- *Evaluation of the use and correct implementation of cryptography and key management.*
- *Analysis of error and exception handling.*
- *Evaluation of the likelihood of security failures being detected.*
  - *Are audit mechanisms reliable and tamper resistant?*
  - *Is data that might be subject to tampering properly validated and authenticated?*
- *Evaluation of the risk that a user can escalate his or her capabilities beyond those which are authorized.*
- *Evaluation of whether the design and implementation follow sound, generally accepted engineering practices. Is code defensively written against:*
  - *bad data,*
  - *errors in other modules,*
  - *changes in environment,*
  - *user errors,*
  - *and other adverse conditions?*
- *Evaluation of whether the system is designed in a way that allows meaningful analysis.*
  - *Is the architecture and code amenable to an external review (such as this one)?*
  - *Could code analysis tools be usefully applied?*
  - *Is the code complexity at a level that it obfuscates its logic?*
- *Search for embedded, exploitable code (such as "Easter eggs") that can be triggered to affect the system.*

- *Search for dynamic memory access features which would permit the replacement of certificated executable code or control data or insertion of exploitable code or data.*
- *Search for use of runtime scripts, instructions, or other control data that can affect the operation of security relevant functions or the integrity of the data.*

…

The review is to provide a "Vulnerability Assessment", based upon the model provided in ISO/IEC WD 18045:2006(E) Information Technology-Security Techniques-Methodology for IT Security Evaluation, App B documenting and categorizing vulnerabilities, if any, to any tampering or errors that could cause incorrect recording, tabulation, tallying or reporting of votes or that could alter critical election data such as election definition or system audit data."

Operation of the Review

The review was conducted from 6 December 2007 to 12 February 2008 at the atsec offices in Austin, TX. The team consisted of multiple experts from atsec (Klaus Weidner, Yi Mao, Lou Losee, Steve Weingart, and Scott Chapman) and was supported by meetings with FCMG (Steve Freeman).

The review (consisting of documentation review and source review) examined the Technical Data Package (TDP) and the source code supplied by ES&S. The TDP and source code used were verified copies of the TDP and source code, which were sent from the National Association of Election State Election Directors (NASED) Independent Test Authority (ITA) lab. The chain of custody followed the files from the lab, to the Secretary of State, to the Source Code Review team at atsec. The integrity of the delivered documents was verified from electronic file signature hashes provided by FCMG from the trusted sources original disks.

The source code review (based on the TDP and source code) used a combination of manual review and automated data collection and analysis methodologies to identify potential areas for exploitation. The source code review was divided into the following categories for reporting:

- Potential Security Critical Vulnerabilities (subdivided by component and type of vulnerability
- Documentation
- Coding Style Analysis, including review of:
    - Control Constructs [VSS Volume II, 5.4.1]
    - Coding Conventions [VSS Volume II, 5.4.2]
    - Adherence to other coding format conventions and standards
    - Program logic and branching structure
    - Commonly exploited vulnerabilities
    - Cryptography and key management

- Likelihood of security failures being detected
- Privilege escalation
- System amenability to analysis
- Exploitable code / Easter eggs
- Dynamic memory access features
- Runtime scripts / instructions / control data
- Best practices / Defensive coding (subdivided by component and type of issue)

Because of the complexity and volume of the material to be reviewed, limited time available and broad scope (assessment of documents and quality of the code, along with source code review), the team concentrated on surveying a breadth of categories of vulnerabilities that they could identify, and only reviewed in depth enough samples of each of the categories to determine how that vulnerability was being handled. For all the categories, no attempt was made to enumerate how many instances existed. Other source code review projects would be likely to find more instances, but those findings should be within the listed categories.

Test tools used included the commercial Fortify source code analysis tool, open source search and analysis tools, and in-house developed scripts. Details specifying tools and sources, as well as the scripts used for the tools are provided in the confidential reports.

Report overview

Full details will be found in the confidential source code review reports (main report plus addendum report), including the detailed work papers. A vulnerability summary table is found at the end of this report, as well as description of the rating system used. The vulnerability rating assessment is based on the Common Methodology for Information Technology Security Evaluation (CEM v3.1) Rev 1 and Rev 2, App B. The use of this terminology is for convenience in characterizing the potential vulnerability of the system to the identified attack, but is not necessarily compliant with and should not be taken as representing a full, formal finding under Common Criteria evaluation methodology.

The summary table includes items where insufficient information was available for a vulnerability assessment. These are marked with “n/a” (not applicable).

Unless otherwise indicated, the potential vulnerabilities found in the source code review have not been confirmed to be exploitable in the fully-deployed environment due to time constraints, and it is possible that technical measures outside of the specific module being examined may prevent an exploit. For the purposes of the vulnerability rating, only assumptions, checks, and protective measures that are clearly identified in the relevant code comments or documentation are considered to be in place. For example, if a function implicitly assumes that parameters are checked or sanitized in a different code location, but no documentation exists for this assumption, the reviewer did not attempt to trace code paths to check if this implicit assumption is appropriate. For security critical sections, the reviewer’s expectation was that either explicit checks or clear and verifiable documentation about assumptions should exist.

Document Assessment

The provided documentation referred to other documents that were not included in the TDP and therefore not available for review. Among the unavailable documents were a system security specification and the ES&S coding standards and development practices. Some security relevant aspects were not covered by the provided documentation

The initial data package was missing multiple files, including source code for encryption libraries, SQL stored procedures, and other security relevant items. An additional data package covering these items arrived too late to be considered in the initial report. The reviewers prepared an addendum report to cover the additional items. This public report covers information from both confidential reports.

The developer did not provide detailed build instructions that would explain how the system is constructed from the source code. Among the missing aspects were details about versions of compilers, build environment and preconditions, and ordering requirements.

Design documentation

The M100 ballot counter is designed to load and dynamically execute binary files that are stored on the PCMCIA card containing the election definition (A.12) in cleartext without effective integrity protection (A.1). The design documentation does not clearly explain this important security relevant functionality; for example, it refers to the corresponding source module as a “resource manager” and does not mention storage of executable files in the description of the storage device characteristics, in the description of inputs, or in the data structure layout specification section. The document claims that “no filesystem is required on the card”, but the card actually does contain an embedded filesystem storing the executable files.

The M100 design documentation contains a 68-page section titled “Programming Specification Details”, but this section actually is completely redundant. Each page consists only of a very brief module description (copied verbatim from a table elsewhere in the document) followed by general statements that are identically repeated on every page. Surprisingly, *“Inputs, Outputs and other Data Elements”* were marked as “not applicable” for all modules. (Filenames such as *comms.c*, *printer.c*, and *scan.c* imply that these modules would contain code related to input and output operations, and most code would normally be expected to act on data elements.) Also, the developer considered *"Design Decisions"* to be not applicable to any of the modules. The stated *“Logic Used”* is always that *“Module logic is based on preserving vote data integrity and system maintainability”*. The section, which is almost half of the document, only serves the purpose of filling pages that at first glance appear to be a specification based on [VSS] requirements, but in actuality, the section does not contribute any useful information.

The M100 design documentation contains a specification of the data structure layout for information stored on the PCMCIA card. The reviewer compared the actual structures as defined in the source code to the documentation, and none of the actual structures matched the specification. Each one showed significant differences to or omissions from the specification.

User guidance

The user's guide for the Election Reporting Manager describes how a password is constructed from publicly-available data. This password cannot be changed, and anyone reading the documentation can use this information to deduce the password. This is not an effective authentication mechanism.

The user's guide for the Hardware Programming Manager does not distinguish between steps done by an administrative user and regular operation, and it is not clear if a non-administrative user can use the system.

Source Code Assessment

Potential or actual vulnerabilities

The source review identified potential or actual vulnerabilities as listed in the appendix of this report and detailed in the confidential report.

This report uses identifiers in the format "A.x" to refer to the specific potential or actual vulnerabilities, where the number "x" corresponds to the section number in appendix A of the confidential report. Each vulnerability is described in the following sections, and cross referenced from other locations in this document where applicable. Please refer to the table in the appendix of this report for the full list.

User authentication and passwords

In the area of passwords and user authentication, the reviewers noted several potential vulnerabilities. Note that the impact of vulnerabilities in this area strongly depends on the authorization required to successfully perform the attack as compared to the additional authorization gained through exploiting the vulnerability. For example, an attack that requires administrative system access to exploit may be uninteresting to an attacker, but one that allows elevating a low-privileged status to higher privileges would be of interest. The point of listing these potential vulnerabilities is to emphasize that password protection in itself should not be counted on as a barrier against unauthorized access, and that a combination of measures also including physical and logical access controls combined with monitoring is generally required. The documentation available to the reviewers did not provide sufficient detail to determine the necessary conditions and possible results of exploiting these issues.

- A.2 "AM Passwords": The password of the database used by the Audit Manager is hardcoded in the source code, and the passwords of all user accounts including the admin account maintained by the Audit Manager are stored as cleartext in the database. An attacker who has access to the software installation media or the installed program can retrieve the password from the binary code.
- A.4 "EDM iVotronic Password Scramble Key and Algorithm": A hardcoded key is used to obfuscate passwords before storing them in a database. The scrambling algorithm is very weak and reversible, allowing an attacker with access to the

scrambled password to retrieve the actual password. The iVotronic is supported by the Unity software but is not being used for California elections.

- A.5 “AIMS Passwords”: The AIMS application source code contains the hardcoded SQL Server Admin password. This password and the user password are protected by very weak encoding/decoding algorithms.
- A.7 “HPM Logon Bypass” and A.11 “ERM Logon Bypass”: The Election Reporting Manager and Hardware Programming Manager applications do not have an authentication mechanism, and the identification mechanism can be bypassed. At the login screen, the user enters a three-letter user ID as identification and is not prompted for a password. The COBOL applications use direct file access with no privilege separation, implying that a user privileged to run the application is likely to have write permission for the data files also, including election definitions for the scanners and election result reports.
- A.8 “HPM/ERM Admin Password”: The administrator password for the Hardware Programming Manager and the Election Reporting Manager can be deduced from the content of its data files. Changes in the election information in the HPM can propagate to all voting terminals and scanners, and changes in the ERM can affect the reported election results.

Audit trail integrity

- A.1 “M100 and M650 Media and Communication”: The audit trail on the storage media does not have an effective integrity protection or tamper detection mechanism.
- A.3 “AM Log Records”: There is no tamper detection mechanism in place to detect the deletion/modification of log data saved in the Audit Manager database. (cf. A.2, which can help enable an attacker gain access to the database.)
- A.10 “HPM/ERM Audit Log”: The Hardware Programming Manager and the Election Reporting Manager do not provide a tamper detection mechanism to detect the deletion or modification of audit log data generated to ensure accountability of user actions.

Cryptography and key management

In the area of cryptography and key management, multiple potential and actual vulnerabilities were identified, including inappropriate use of symmetric cryptography for authenticity checking (A.9) and several different very weak homebrewed ciphers (A.4, A.7, A.8, A.11). In addition, the code and comments indicated that a checksum algorithm that is suitable only for detecting accidental corruption is used inappropriately with the claimed intent of detecting malicious tampering (A.1).

- A.1 “M100 and M650 Media and Communication”: The storage card containing the election definition is unencrypted, allowing an attacker to obtain sensitive information in clear text (such as the dial-in information for contacting the host receiving election data; and the reopen password used to add additional ballots to a closed election). The data also lacks effective integrity protection, enabling an attacker to modify data such as the election definition, executable files (cf. A.12), and the vote totals stored on the card or being transmitted. The modem data transmission feature is not being used for California elections, but the issue is still applicable for the data transfer via storage cards.

- A.6 “AIMS File Encryption”: The file encryption/decryption mechanism to protect the data transfer between AIMS and other components (such as the Election Data Manager) is very weak.
- A.9 “HPM/EDM/ESSIM Encryption Key”: The encryption key used for sending encrypted files between the Election Data Manager and the Hardware Programming Manager is hardcoded in the source code. Note that the application also supports use of cleartext files, and this may be appropriate depending on the deployed environment, but users should not assume strong protection when enabling the encryption feature. The same symmetric key is used for encryption and decryption, allowing an attacker to decrypt, modify, and re-encrypt data.

Code and data integrity issues

The developers generally assume that input data will be supplied in the correct expected format. Many modules that process input data do not perform data validation such as range checks for input numbers or checking validity of internal cross references in interlinked data, leading to potentially exploitable vulnerabilities when those assumptions turn out to be incorrect. For example, the following issues are directly related to insufficient checking of input data:

- A.12 “M100 executables on PCMCIA card”: The storage card used in the M100 for the election definition also contains binary executable files that are loaded from the card and dynamically executed. An attacker could replace the programs with malicious code (cf. A.1).
- A.13 “M100 unchecked pointer arithmetic”: When loading the election definition from the PCMCIA card into RAM, offsets stored on the card are converted to memory pointers with no data validation, which can potentially access arbitrary application memory and cause execution of arbitrary code.
- A.14 “M100-M650 memory allocation integer overflow”: When loading the election definition from storage media into RAM, size fields stored on the media are used in calculations, with no data validation performed. This can result in integer overflow and allocation of insufficiently-sized storage buffers, which can potentially be exploited to execute arbitrary code supplied on storage media.
- A.15 “M650 command injection”: The program builds instructions to the command interpreter at runtime and passes these instructions to the operating system for execution. The generated command lines contain unsanitized data from external sources, which could be exploited to execute malicious commands.
- The Audit Manager application constructs SQL statements at runtime using string concatenation including unchecked user supplied values, leading to potentially exploitable SQL injection vulnerabilities.

Easter eggs / back doors

The reviewers found no instances of deliberately inserted back doors or Easter eggs. However, many of the vulnerabilities could potentially be exploited by a knowledgeable insider, such as A.12/A.13/A.14, which may enable an attacker to insert executable code into the system. The issues related to hardcoded passwords, missing authentication and weak encryption could be used to bypass expected security features.

Adherence to applicable standards

The Voting Systems Performance and Test Standards [VSS (2002)] provide standards related to coding conventions and best practices. The reviewer examined the source code for applicable items as specified in the work plan and reported the result of this examination.

The confidential report lists specific instances where code constructs do not appear to match the requirements of this standard. Note that these instances are not automatically equivalent to actual or potential vulnerabilities. This summary report omits instances that do not appear noteworthy. The following items are considered significant because they are an obstacle for effective code analysis, potentially hiding other problems, or because they could contribute to introducing problems in later changes to the source code.

- Missing validation of input parameters or otherwise inadequate specification of expected range values, including instances of mismatches between the documented and actual semantics of functions, and potential vulnerabilities related to corrupting memory allocations due to integer overflow (cf. A.14).
- Missing array range checks and use of unchecked pointer arithmetic when accessing data in C/C++ code, leading to undocumented dependencies between distant code segments and to potential vulnerabilities (cf. A.13).
- Potential vote counter/integer overflow in the Election Reporting Manager due to instances of non-range-checked arithmetic involving vote counters. There is a range check for the results counter number, but this is insufficiently documented and it is unclear from review if this provides sufficient assurance in cases where input data may have unexpected values.
- Uses of numeric constants other than 0 and 1 should be explained by expressive variable names or code comments. Multiple cases were found where constants were used without adequate explanation. Most examples occur where the constant value is named as a variable but the name does not indicate why that value is significant, for example defining a constant FIVE with the value "5" with no further explanation.
- Several instances of nested use of the conditional "?:" operator in complex multi-line expressions.
- Use of the confusingly similar hardcoded usernames "AIMSMgr" and "AIMS Manager" for two distinct database users.
- An apparent mixup where an encryption function is used where a call to the corresponding decrypt function was intended.

- Instances of misleading code comments that do not match the implementation. For example, a function initializing the filesystem providing executable files on the M100 (A.12) has comments explaining that an integrity check is done before activating the filesystem. Actually, the filesystem is already activated by a startup file when the system boots, and the call to this function and the integrity check happen afterwards. (Note that the integrity check would not offer effective protection against intentional modification in any case, cf. A.1.) Change history comments elsewhere in the file indicate that the comments document the way a previous version of the software worked. The programming design specification for this module is limited to the explanation that “Module logic is based on preserving vote data integrity and system maintainability”.

Overall secure design and implementation

Many of the applications run at a privilege level that provides full read/write access to all of their security-critical application data. In these cases, the ‘least privilege’ principle is not exercised. Lack of privilege separation in the design makes it difficult to ensure reliable detection of security failures.

The AIMS source code does make a distinction between the three different database users “sa” (server admin), “AIMSMgr” (some administrative rights) and “AIMS Manager” (normal user), and this role split combined with a separate database engine appears suitable to provide privilege separation. Unfortunately, the documentation does not explain the different privileges assigned to these users and the security aspects of the role separation. Also, the application uses a built-in database engine that runs with the same rights and privileges as the application process, with the effect that the application is potentially able to bypass the official data access interfaces and their permission checks and access raw data directly. This limits the actual separation that can be achieved in this environment.

Design documents and code comments do not provide assurance that audit logs are protected from tampering. The code segments doing logging generally have sufficient privileges to modify or delete logs due to lack of privilege separation. The design documents do not mention use of operating system features or other measures to support the integrity of the logs.

The applications generally fully trust data received from external interfaces, including doing unchecked pointer arithmetic and memory allocation based on values loaded from storage media (A.13, A.14) and executing binaries provided on removable media (A.12). Secure software should always perform data validation on external inputs.

System amenability to analysis

The reviewer noted the following items as impediments to an effective security analysis of the system:

- Design documentation that is insufficiently detailed and in some cases incorrect.
- Incomplete information about external resources, such as registry contents and permission settings.

- The system design does not consistently use privilege separation, leading to large amounts of code being potentially security-critical due to having privileges to modify data.
- Unhelpful or misleading comments in the code.
- Subjectively, large amount of source code compared to the functionality implemented.

The code constructs used were generally straightforward and easy to follow on a local level. However, the overall complexity of components and their interfaces, combined with insufficient design documentation, made it difficult to globally analyze the system.

Results summary

The security issues summarized in this document and detailed in the confidential reports raise serious concerns about the assurance level of claimed security features of the Unity system.

The system is designed to execute code supplied on the election definition memory cards on the precinct ballot counters, with no effective measures to ensure integrity and authenticity of this code. Due to this, there is little assurance that the systems will actually be running the reviewed code at election time.

The system fails to provide strong Identification and Authentication for access control. Some components have no access controls at all. For those components that do restrict access by requiring a User ID, a password, or a User ID/password pair, the login credentials are either hardcoded in the source code, stored in clear text in a database, or at best, scrambled with extremely weak algorithms that do not prevent credentials from being discovered. Thus, all components in the system are potentially exposed to unauthorized access.

The system fails to provide confidentiality and integrity of election data (including election definitions and election results). The election data is transferred among components of the Unity System via removable media devices. The data on the media devices is either in plain text, stored with simple checksum values, obfuscated with extremely weak homebrewed algorithms, or at best encrypted with symmetric algorithms. In cases where the data to be transferred is encrypted, the encryption keys are hardcoded either as a plain ASCII string or with a simple obfuscation that can be easily reversed. The election data can be maliciously modified by a component of the Unity System or during the transition of the media from one component to the other, but yet still be treated as valid by other Unity System components.

The system fails to provide reliable accountability for audit logs. Audit logs of the Unity system are kept either in databases or log files, none of which are protected by any kind of tamper-detection mechanism. The audit log of the system is susceptible to tampering without being detected.

The developers generally assumed that input data will be supplied in the correct expected format. There is little validation checking of the data, leading to potentially exploitable vulnerabilities when those assumptions turn out to be incorrect, for example, due to malicious manipulation of the election definition leading to execution of attacker-supplied code.

The security of the Unity System depends on its secure use, which assumes that all parties involved in developing, maintaining, distributing, deploying and using the Unity system must be trustworthy. This assumption is equivalent to saying that there are no threats to the Unity system.

A mitigating factor is that the system is based on optical scanning of human readable paper ballots, which provides the possibility of a manual recount to verify accuracy of the automatically tabulated results. Such recounts depend on a secure chain of custody of the marked ballots, but that is beyond the scope of this review. Note that the ballot counter's feature to automatically sort ballots into destination containers cannot be considered fully trustworthy unless the software vulnerabilities are addressed.

SUMMARY TABLE OF SECURITY TESTING FINDINGS

			Attacker						Vulnerability Assessment					Total
Ref	Title	Component	Voter	Poll worker	Election official	Storage Personnel	Vendor	Scalability	Time	Expertise	Knowledge	Window of Opportunity	Equipment	Attack Resistance
A.1	M100, M650	Media and communication		X	X	X	X	Low-High*	0	6	3	4	0	Enh-Basic
A.2	AM	Passwords			X		X	High	0	6	3	1	0	Enh-Basic
A.3	AM	Log records			X		X	High	0	6	3	1	0	Enh-Basic
A.4	EDM	iVotronic password scramble key and algorithm			X		X	High	0	6	3	1	0	Enh-Basic
A.5	AIMS	Passwords			X		X	High	0	6	3	1	0	Enh-Basic
A.6	AIMS	File encryption			X		X	High	0	6	3	1	0	Enh-Basic
A.7	HPM	Logon bypass			X		X	High	0	0	0	1	0	None
A.8	HPM, ERM	Admin password			X		X	High	0	3	3	1	0	None
A.9	HPM, EDM, ESSIM	Encryption key			X		X	High	0	3	3	1	0	None
A.10	HPM, ERM	Audit log			X		X	Low	0	0	0	1	0	None
A.11	ERM	Logon bypass			X		X	High	0	0	0	1	0	None
A.12	M100	Executables on PCMCIA card		X	X	X	X	Low-High*	0	6	3	4	0	Enh-Basic
A.13	M100	Unchecked pointer arithmetic		X	X	X	X	Low-High*	0	6	3	4	0	Enh-Basic
A.14	M100, M650	Memory allocation integer overflow		X	X	X	X	Low-High*	0	6	3	4	0	Enh-Basic
A.15	M650	Command injection			X	X	X	Low-High*	not rated					n/a

* A range given for scalability such as "Low-High" refers to attacks where the scalability depends on the type of attacker, for example comparing a poll worker to a vendor insider.

Legend for the Summary Table of Security Testing Findings

Vulnerability Assessment Coding:

1. Time to Exploit. "…total amount of time taken by an attacker to identify that a particular potential vulnerability may exist in the TOE, to develop an attack method and to sustain effort required to mount the attack against the TOE. "[CEM v3.1, App B] "TOE" is the target of evaluation.

2. Expertise. "…the level of generic knowledge of the underlying principles, product type or attack methods "[ibid]

3. Knowledge of Target of Evaluation (TOE). "…specific expertise in relation to the TOE."[ibid]

4. Window of Opportunity. "…equate to the number of samples of the TOE that the attacker can obtain. This is particularly relevant where attempts to penetrate the TOE and undermine the SFR may result in the destruction of the TOE preventing use of that TOE sample for further testing, e.g. hardware devices"[ibid]. For this test, the Window of Opportunity includes limitations on accessing a specific feature which has significance to the security of the system. "SFR" is "Security Functional Requirement" which is a member of set of formally, predefined security requirement in the Common Criteria standards that are used as a basis for interpreting and testing security requirements for a TOE.

5. Equipment, hardware/software or other. "…the equipment required to identify or exploit a vulnerability " [ibid]

"Table 3, Calculation of Attack Factor" [ibid]

Factor	Value
Elapsed Time	
≤ one day	0
≤ one week	1
≤ two weeks	2
≤ one month	4
≤ two months	7
≤ three months	10
≤ four months	13
≤ five months	15
≤ six months	17
> six months	19
Expertise	
Layman	0
Proficient	3*(1)
Expert	6
Multiple experts	8
Knowledge of TOE	
Public	0
Restricted	3
Sensitive	7
Critical	11
Window of Opportunity	
Unnecessary / unlimited access	0
Easy	1
Moderate	4
Difficult	10
None	**(2)
Equipment	
Standard	0
Specialized	4(3)
Bespoke	7
Multiple bespoke	9

(1) When several proficient persons are required to complete the attack path, the resulting level of expertise still remains "proficient" (which leads to a 3 rating).

(2) Indicates that the attack path is not exploitable due to other measures in the intended operational environment of the TOE.

(3) If clearly different test benches consisting of specialized equipment are required for distinct steps of an attack, this should be rated as bespoke"

"bespoke" is specified when "…clearly different test benches consisting of specialised equipment are required for distinct steps of an attack"[ibid].

"Table 4, Ratings of vulnerabilities and TOE resistance" [ibid]

Values	Attack potential required to exploit scenario:	TOE resistant to attackers with attack potential of:
0-9	Basic	No rating
10-13	Enhanced-Basic	Basic
14-19	Moderate	Enhanced-Basic
20-24	High	Moderate
≥ 25	Beyond High	

As an example, consider an attack on a poorly designed bicycle lock. Assume the attack can be done by anyone with access to the bicycle in less than 20 minutes (≤ one day). The attack requires some skill with locks (Proficient) and knowledge of how the lock works (Restricted) to be effective. Windows of Opportunity are somewhat limited (Moderate) because other observers would be expected to respond, but the tools used are commonly available (Standard). The resulting vulnerability to access (total of the factors=10) barely qualifies as Enhanced-Basic, which implies that the attack would require more than a casual event but would not deter a moderately capable attacker.

In contrast, consider an attacker attempting to retrieve a password contained within the data on an unencrypted backup CD. The attack requires information gained through experience with the system and system documentation (Knowledge of system=3) and some common software to examine the file contents, but does not require additional time (≤ one day) or special tools (Standard). Some knowledge of the system to recognize the files and clear text contents are needed (Restricted), but the clear text may be read by a layman (Layman). The Window of Opportunity requires getting a copy of the CD (Easy). This gives a total vulnerability risk of Basic (Total of factors = 4).

These two examples are both foundation attacks that support other attacks by opening accesses and acquiring knowledge of the system that may be used in other attacks.

Public Report prepared by:

Klaus Weidner, Principal Consultant, atsec information security

atsec reviewers:

Scott Chapman, Principal Consultant, atsec information security [Reviewer]

Helmut Kurth, Vice President, atsec information security [Final Approver of Report]

Lou Losee, Senior Consultant, atsec information security [Reviewer]

Yi Mao, Consultant, atsec information security [Reviewer]

Klaus Weidner, Principal Consultant, atsec information security [Lead Reviewer]

Steve Weingart, Consultant, atsec information security [Reviewer]