January 16, 2009

Mr. Brian Hancock **Sent via E-mail**
U.S. Election Assistance Commission
Voting System Testing and Certification Program
1225 New York Avenue, NW, Suite 110
Washington, DC 20005

Mr. Hancock,

The purpose of this letter is to document the 3% review of the Unity 3.2.0.0 source code in accordance with your 21 November 2008 email providing instruction on the reuse of testing for the ES&S certification effort. This letter also provides the iBeta recommendation to the EAC regarding the reuse of the source code review conducted by SysTest.

Documentation of the Review Process

To conduct the review, iBeta used our PCA Source Code Review Procedure. The source code was delivered from SysTest Labs and configuration managed in the iBeta Source Code Repository. With the exception of Cobol, the coding languages submitted for review had been previously reviewed on other certification test efforts therefore the previously used interpretation of the generic VSS requirements to the language specific review criteria were utilized unmodified. For the Cobol review, iBeta provided the interpretation of each VSS requirement to ES&S prior to initiating the source code review task. The language specific review criteria for each of the five languages is not attached to this letter and will be provided if deemed necessary for the EAC review. The VSS requirements applicable to the source code review task are:

VSS	
Vol. #	**Section(s) #**
1	4.2.2
1	4.2.3
1	4.2.4
1	4.2.5
1	4.2.6
1	4.2.7
1	6.2
1	6.4.2
2	2.5.4d
2	5.4.2

To select the 3% for review, iBeta conducted an analysis by first using a library of static analysis tools to parse each application source code base and provide a list of the files and functions as well as the Lines of Code (LOC) count. iBeta uses executable LOCs only and does not include comment, blank, or continued lines in our metrics. An exception to this process was the Cobol applications as our library of static analysis tools do not address Cobol source code. For those two applications, the number of files and files sizes were used to determine the volume of code in order to select 3%.

Once the spreadsheets were populated for each application, a selection of files/functions was made based on the file header information documenting the file purpose. iBeta focused the review by selecting source code files and functions that process vote data, audit logs, and reporting.

The ES&S AutoMARK source code submitted was compared against the source code submitted with the Premier certification effort as the code is similar. The differences between those two source code bases were then reviewed as part of the ES&S 3% source code review. The unique as well as the shared application discrepancies are reported herein.

The peer review of each Source Code Review was conducted by experienced reviewers who had reviewed source code to the VSS requirements on a minimum of two VSTL test efforts. Based on the instruction in your 21 November 2008 email "*This review will focus on important functional sections of the code in order to determine the depth and focus of source review conducted by SysTest*", the peer review analyzed each instance of non-compliance with the VSS requirements and assessed if the issue impacted source code logic. Discrepancies that dealt with comments, headers, formatting, and style were accepted as non-logic issues and color coded as green. Potential logic issues were flagged as needing an EAC decision and color coded as yellow. Confirmed logic issues were to be flagged as red (no confirmed logic issues were identified).

The matrix of the source code reviewed is provided as Attachment 1 and each individual discrepancy spreadsheet is provided as a separate confidential compressed file delivered on CD subsequent to the email delivery of this letter.

Summary of 3% Source Code Review Results

A total of 330 discrepancies were identified with the majority, 307 or 93%, categorized as non-logic issues. The summary of discrepancies categorized as EAC Decision Discrepancies as well as the vendor responses are provided as Attachment 2 to this letter.

Of the 21 of 23 potential logic discrepancies, ES&S has provided in their response their justification for non complying with the requirement or their disagreement of the iBeta interpretation of the VSS requirements. Precedence for the iBeta interpretation has been established with testing for other clients and these established interpretations must be applied consistently to all manufacturers under test with iBeta.   We do acknowledge that in some instances another interpretation may be possible and that alternative interpretation may be acceptable to the EAC reviewers.

The remaining 2 potential logic discrepancies remain under investigation by both Premier and ES&S and are expected to be addressed within the Premier source code review letter.

Recommendation Regarding the Reuse of the SysTest Source Code Review

In order to provide a recommendation, iBeta evaluated the results of the 3% source code review whereas the results would be recommended as accepted if no significant discrepancies were found, this includes the less critical requirement which were not addressed, not recorded or interpretations are inconsistent with documenting industry accepted practices. As there were discrepancies written that potentially impact the source code, two other analyses were conducted:

1. Confirmed that the results of the iBeta review of the 3% of code are consistent with the previous results (not identical but consistent): This confirmation was reached by reviewing the types of discrepancies generated by SysTest in the 100% review against those generated by iBeta.

2. Reviewed the severity of the discrepancies discovered: The number of discrepancies potentially impacting the source code is considered very low versus the overall number of discrepancies (as is consistent with a 100% review). The severity of the discrepancies and the vendor responses do indicate that the majority of those 21 potential logic discrepancies would be resolved without source code modifications.

Based on the limited impact (or perhaps no impact) on the source code as a result of these discrepancies, iBeta recommends reuse of the results of the SysTest source code.

Sincerely,

Gail Audette

Gail Audette
iBeta Quality Manager

Attachment 1: Matrix of Source Code Reviewed
Attachment 2: Summary of Discrepancies

Enclosure: CONFIDENTIAL CD Source Code Review Discrepancies 1-16-09.zip

cc: Steve Pearson, ES&S
Sue Munguia, ES&S

CONFIDENTIAL - Attachment 1: Matrix of Source Code Reviewed - CONFIDENTIAL

Product	Source Code Language	Version of Code Submitted to VSTL	Date Code Submitted to VSTL	Spreadsheet	Lines Reviewed	Total Lines	Total Issues	EAC Issues
Unity 3.2.0.0 Software								
AutoMARK Information System (AIMS)	Various	1.3.57	08/16/07	Shared application	887	26539	9	2
	SQL			SQL AIMS 1.3.54 08062007			2	2
	CS			to few lines to review	0	38	0	0
	C++			CPP AIMSCrypt 1.0.0.1 10152008	16	400	2	0
Audit Manager	VB	7.5.0.0g	07/31/07	VB AuditManager 7.5.0.0g 07312007	138	3556	0	0
EDM	C++	7.8.0.0j	07/31/07	CPP EDM 7.8.0.0j 073107	2539	72879	6	1
ESSXML.DLL	C++	2.1.0.0b	06/04/07	CPP EDM ESSXML 2.1.0.0b MFC Shared 1.1.0.0a 06042007	111	2870	1	0
MFC Shared Source	C++	1.1.0.0a	06/04/07	CPP EDM ESSXML 2.1.0.0b MFC Shared 1.1.0.0a 06042007				
ESSIM	C++	7.7.0.0f	07/18/07	CPP ESSIM 7.7.0.0f 07182007	1196	30546	26	1
HPM	Cobol	5.7.0.0f	05/14/08	Cobol HPM 5.7.0.0f 05182008			178	0
HPMDLL	C++	1.0.0.0a	06/11/07	CPP HPM-ERM DLLs 1.0.0.0a 06112007	0	108	0	0
ERM	Cobol	7.5.2.0c	10/24/08	Cobol ERM 7.5.2.0c			53	4
ERMDLL	C++	1.0.0.0a	06/11/07	CPP HPM-ERM DLLs 1.0.0.0a 06112007	0	0	0	0
Shared Utilities								
MAKEIBIN.EXE	C++	9.2.0.0t	08/07/07	CPP Shared Utilities 9.2.2.0 05142008	642	20804	7	2
UNDRVOTE.EXE	C++	9.2.1.0b	05/31/07	CPP Shared Utilities 9.2.2.0 05142008				
VIOWIN.EXE	C/C++	9.2.0.0b	05/07/07	CPP Shared Utilities vol3 05072007	28	554	3	0
VIODIALOG.EXE	C/C++	9.2.1.0c	05/14/08	CPP Shared Utilities 9.2.2.0 05142008				
EVENTS.EXE	C/C++	9.2.0.0h	06/19/07					

3131 South Vaughn Way • Suite 650 • Aurora, CO 80014 • Phone (303) 627-1110 • Fax (303) 627-1221

IMAGES.EXE	C/C++	9.2.0.0f	05/16/07					
CF_Utility.EXE	VB	9.2.0.0i	05/07/07	VB CF_Utility 9.2.0.0 05072007	261	8004	0	0
GetAuditData.EXE	VB	9.2.0.0b	05/07/07	VB GetAuditData 9.2.0.0b 05072007	46	1264	1	0
ESSPEB.DLL	C++	1.0.1.0c	05/15/08	CPP Shared Utilities vol2 1.0.1.0 05142008	478	24872	16	7
CB_PEB.DLL	C++	1.0.1.0b	05/14/08	CPP Shared Utilities vol2 1.0.1.0 05142008				
CRCDLL.DLL	C++	1.4.1.0b	05/07/07	CPP Shared Utilities vol3 05072007				
ESSM100.DLL	C/C++	1.7.1.0c	05/06/08	CPP Shared Utilities vol2 1.0.1.0 05142008				
ESSPCMIO.DLL	C++	1.1.0.0a	08/07/07					
CB_M100.DLL	C++	1.4.0.0a	08/07/07					
ESSEAGL.DLL	C++	1.3.1.0e	07/20/07					
CB_EAGL.DLL	C++	1.3.1.0c	05/31/07					
CB_RAND.DLL	C++	1.1.0.0a	08/07/07					
MYDLL.DLL	C	1.1.0.0a	08/07/07	C ESS all Unity 3.2 04282008	538	17750	12	1
MPRBOOT.HEX	Assembler	2.6.1.0b	05/16/07	ASM MPRBOOT 2.6.1.0b 05162007.xls	56	1340	0	0
ESSCRYPT.DLL	C/C++	1.9.0.0a	07/31/07	CPP Shared Utilities vol2 1.0.1.0 05142008				
ESSDECPT.EXE	C++	1.9.0.0a	07/31/07					
ESSCRPT1.DLL	C++	1.1.0.0b	05/16/07					
ElectionPackager	C++	1.0.0.0e	07/06/07					
ESSZIP	C++	2.0.0.0f	07/06/07					
PCCARD30.EXE	C++	3.5.0.0h	05/14/08					
PBMtoBMP	C++	1.1.0.0c	04/18/08					
WIN650	C++	2.2.1.0.4	05/31/07					
INIT650.EXE	C/C++	2.2.1.0.4	05/31/07					
SERVE650.EXE (Newserve650)	C++	2.2.1.0.4	05/31/07					
CB_650.DLL	C	1.2.0.0a	08/07/07	C ESS all Unity 3.2 04282008				
REGUTIL.DLL	C++	1.1.0.0d	05/31/07	CPP Shared Utilities vol2 1.0.1.0 05142008				
SHELLSETUP.EXE	C++	1.1.0.0a	04/12/07					
SHELL.EXE	C++	1.1.0.0b	05/07/07	CPP Shared Utilities vol3 05072007				
EXITWIN.EXE	VB	1.1.0.0a	04/12/07	VB ExitWin 1.1.0.0a 04122007	33	469	0	0
Firmware								
****Model 200****								
TOS /wo JVM		N/A	N/A					
DS200	C/C++	1.3.7.0g	04/23/08	CPP DS200 1.3.7.0g 04282008	386	12552	2	1

Power Management_MSP430	C	1.2.0.0a	04/28/08	C DS200 all 1.2.0.0a 04282008	741	20930	3	0
Scanner_C8051	C	2.11.0.0a	04/28/08	C DS200 all 1.2.0.0a 04282008				
****Model 650****								
M-650	C	2.2.1.0.5	06/20/07	C ESS all Unity 3.2 04282008				
****AutoMARK****								
AutoMARK-Voter Assist Terminal (VAT)	Various	1.3.2816	09/18/07	CPP VAT (ESS ScannerPrinterLibrary 1.8.31-GetMarks 1.4.9) 10152008	679	21026	9	2
Totals					8775	266501	330	23
Percentages						3.3%		7%

Compiled Discrepancy Report for *ES&S*

Lang-uage	Compo-nent	Version	Spread-sheet	Disc #	File	Func-tion	Discrepancy Description	VSS Refer-ence	iBeta Classification	ES&S Vendor Response
C	WIN650: folder 07-0531 Shared Utilities\WIN650 2.2.1.0.4\Source	2.2.1.0.4	Discrepancies C ESS all Unity 3.2 04282008.xls	10	msInit\etp_rev.c	Permute8Bytes	line 329 hard-coded key.	v1: 6.4.2	Hard-coded key	The hard coded table cited is used in an old scheme to "scramble" or obfuscate the M650 audit log file before it is written to the M650 internal file on the M650 internal RAM drive. The audit log file is printed in real-time on a continuous form matrix printer and becomes the audit log of record. This table and its contents are well commented so it passes the test for hard constants. This function is not used in any way to validate or protect the firmware.
Cobol	HPM	5.7.0.0f	Discrepancies Cobol HPM 5.7.0.0f 05182008.xls	23	PE001AL L.PRC	910-SET-EQUIP-TYPE	Series of ELSE IF clauses is missing the final ELSE clause	v.1: 4.2.4.a	iBeta interpretation for the control contructs requirement is violated.	Volume I, Section 4.2.4.a specifies the acceptable control constructs to be used. One of the listed acceptable control constructs is If-Then-Else. This section does not elaborate any further on the acceptable different forms of syntax for If-Then-Else statements. It is our belief that the sections of code cited in this discrepancy are structured, sound, easily understood and accepted syntax forms of IF-Then-Else statements.
Cobol	HPM	5.7.0.0f	DiscrepanciesCobol HPM 5.7.0.0f 05182008.xls	24	PE001AL L.PRC	911-GET-EQUIP-TYPE	Procedure header contains ONLY description no other required info for procedure over 10 lines of code Series of ELSE IF clauses is missing the final ELSE clause Lines 399,402 and 405 contain non-enumerated constants	v.1: 4.2.3.b 4.2.7 (a, a.1-a.6) v.1: 4.2.4.a v.2: 5.4.2.u	1. iBeta interpretation for the Exit Point requirement is violated. 2. iBeta interpretation for the control constructs requirement is violated. 3. Non-enum constants is acceptable per discrepancy 20 explanation.	Volume I, Section 4.2.4.a specifies the acceptable control constructs to be used. One of the listed acceptable control constructs is If-Then-Else. This section does not elaborate any further on the acceptable different forms of syntax for If-Then-Else statements. It is our belief that the sections of code cited in this discrepancy are structured, sound, easily understood and accepted syntax forms of IF-Then-Else statements.

Cobol	HPM	5.7.0.0f	DiscrepanciesCobol HPM 5.7.0.0f 05182008.xls	25	PE001ALL.PRC	912-SET-COUNT-TYPE	Procedure header contains ONLY description no other required info for procedure over 10 lines of code Series of ELSE IF clauses is missing the final ELSE clause Lines 415, 417, 422, 425, 428, 431, 436, 439, 442, 445,449, 452, 455 and 458 contain non-enumerated constants	v.1: 4.2.3.b 4.2.7 (a, a.1-a.6) v.1: 4.2.4.a v.2: 5.4.2.u	1. iBeta interpretation for the control contructs requirement is violated. 2. Non-enum constants is acceptable per discrepancy 20 explanation.	Volume I, Section 4.2.4.a specifies the acceptable control constructs to be used. One of the listed acceptable control constructs is If-Then-Else. This section does not elaborate any further on the acceptable different forms of syntax for If-Then-Else statements. It is our belief that the sections of code cited in this discrepancy are structured, sound, easily understood and accepted syntax forms of IF-Then-Else statements.
Cobol	HPM	5.7.0.0f	Discrepancies Cobol HPM 5.7.0.0f 05182008.xls	26	PE001ALL.PRC	914-SET-VOTE-FOR	Procedure header contains ONLY description no other required info for procedure over 10 lines of code Series of ELSE IF clauses is missing the final ELSE clause Lines 467, 470 and 473 contain non-enumerated constants	v.1: 4.2.3.b 4.2.7 (a, a.1-a.6) v.1: 4.2.4.a v.2: 5.4.2.u	1.iBeta interpretation for the control contructs requirement is violated.. 2. Non-enum constants is acceptable per discrepancy 20 explanation.	Volume I, Section 4.2.4.a specifies the acceptable control constructs to be used. One of the listed acceptable control constructs is If-Then-Else. This section does not elaborate any further on the acceptable different forms of syntax for If-Then-Else statements. It is our belief that the sections of code cited in this discrepancy are structured, sound, easily understood and accepted syntax forms of IF-Then-Else statements.
CPP	EDM	7.8.0.0j	CPP EDM 7.8.0.0j 073107.xls	5	geodlg.cpp	CGeoDlg::OnClickedDelete	1) multiple embedded calls in logical statement at lines 856, 871 2) Illegal breaks at lines 847, 859, 874, line 880 changes the state of the system and therefore break statements are not allowed. If code deletes one it must delete all in order to complete unit operation described.	v.1: 4.2.3.ev.2: 5.4.2.m	Multiple exits	This noted discrepancy is an IF statement that tests the result of several Boolean returning functions. ES&S does not consider these to be embedded statements, the functions aren't doing processing in the sense that they change the state of the system or change any value. Rather they are functions that fetch or otherwise determine a value and return the value. This may be something difficult for a reviewer to discern so they would just flag it because it is a function within a conditional expression. As for the second part of item #5 ES&S would disagree with the reviewer. No state changes (precinct deleted) are made until after the conditions that can trigger those breaks are passed. It is not necessary that all precincts be deleted from the list in this code.

CPP	DS200	1.3.7.0g	Discrepancies CPP DS200 1.3.7.0g 04232008.xls	2	MSP430_BootLoader\src\command.cpp	CBSLCommand::Apply_1_10_Patch	Line 540, 587 illegal write of executable binary not created through a trusted build (or must be proven COTS) (If MSP430's version is equal to or less than version 1.10)	v.1: 4.2.2, 6.2	Self modifying code	The source code for the Texas Instruments MSP-430 micro-controller used in the Power Management Board of the DS200 contains the application of a patch mandated by the microcontroller manufacturer to fix a defect in the 1.10 and earlier versions of the Texas Instruments boot loader. This is a COTS part and the patch is unmodified and obtained directly from the manufacturer. ES&S does not believe that a mandatory, factory supplied COTS patch is subject to the VVSG and regardless of jurisdiction the patch does not in any way compromise the integrity of the power board or the DS200 system. The patch as applied by the ES&S code is unmodified from the patch offered on the Texas Instruments web site. The patch does not change and does not modify anything. Rather, it fixes a flaw in the original boot loader that allows the microcontroller to function correctly. The power management board itself has no connection whatsoever to the accumulation, storage or tabulation of the election data. It is an ancillary device that is completely separate from the main board. ES&S does not believe the application of a COTS patch from a microcontroller manufacturer violates the intent of the cited section of the VVSG.
CPP	ESSIM	7.7.0.0f	Discrepancies CPP ESSIM 7.7.0.0f 07182007.xls	26	IFCUtil.cpp	CBalStyleNum::Compare	Line 3693: validate input argument (no validation, two else clauses would return 0 (equal) if one or both of the values are out of bounds for some reason, which makes no sense to this reviewer)	v.2: 5.4.2.a	Input validation	The logic discrepancy is from the addition of empty mandatory 'else' clauses. The function compares two values and will make a comparison of two other values nested within one of the first comparison tests. The comparisons in this function can only have one of three outcomes. The code explicitly tests for all three possibilities and therefore requires a mandatory 'else' clause which will be empty. Those two empty clauses are unreachable, regardless of argument values, and the logic will not fall through them. The function will produce the exact same results, had the last 'else if' clauses been coded as 'else' clauses. That would make them the mandatory else clauses and eliminate the empty 'else' clauses
CPP	VIODIALOG	9.2.1.0c	Discrepancies CPP Shared Utilities 9.2.0.0 05142008.xls	6	Vio Shared Files\VioDialog\NewILJWSC32.cpp	SioReset_CB	1) Parameter list is incomplete in header section. 2) "DCBptr" is not validated. 3) Constants other than "0" or "1" enumerated or defined:0x11,0x13,512,4,128.	v.1: 4.2.3.b 4.2.7 (a, a.1-a.6) v.2: 5.4.2.e v.1: 4.2.7.b	2. Pointer validation	The function that calls SioReset_CB() is called SioReset(). The DCBptr is assigned in SioReset(), and it is set to point at a persistent global variable in the file. Therefore, the pointer DCBptr can never be NULL. This global is also properly initialized, and the "Port" parameter is verified. This pointer does not need to be checked in SioReset_CB() because it will always be pointing to this global variable. If for some reason the pointer is passed to SioReset_CB() and it is NULL, that indicates the microprocessor is not executing the program code properly, and any checking we would do would be pointless.

CPP	VIODIALOG	9.2.1.0c	Discrepancies CPP Shared Utilities 9.2.0.0 05142008.xls	7	Vio Shared Files\Vio Dialog\NewILJWSC32.cpp	SioFlow	1) Need default case for switch() statement . 2) Line 1262,1266: Need exlicit comparison in if() statement.	v.2: 5.4.2.f v.2: 5.4.2.t	1.Default case	There is no default case in the given switch statement because we do not want the DCBptr settings to change if we are given a "Cmd" value that we do not recognize. We would only have an empty (no code) default case at the end of the switch statement.
CPP	ESSCRPT1	1.1.0.0b	Discrepancies CPP Shared Utilities vol2 1.0.1.0 05142008.xls	6	Source\BlowFish.cpp	PermuteFunc	"ct" is not validated.	v.2: 5.4.2.e	Pointer validation	The parameter “ct” is defined as a pointer to a structure of the type “SBlowfishCipherTables”. If the functions are called with a pointer to any other data structure type, the compiler will generate an error. The “ct” parameter is implicitly verified by the compiler.
CPP	ESSCRPT1	1.1.0.0b	Discrepancies CPP Shared Utilities vol2 1.0.1.0 05142008.xls	7	Source\BlowFish.cpp	BlowfishInit	"ct" & "key" are not validated.	v.2: 5.4.2.e	Pointer validation	The parameter “ct” is defined as a pointer to a structure of the type “SBlowfishCipherTables”. If the functions are called with a pointer to any other data structure type, the compiler will generate an error. The “ct” parameter is implicitly verified by the compiler. The parameter “key” is protected from overflow by the parameter “keyLen” which is defined in the comments as the length of the “key”. It is the programmer’s responsibility to define the length of “key” when the function is called. The internal data structures are protected from “keyLen” being too large by clamping the parameter to a maximum value of “MAX_KEY_BYTES”.
CPP	ESSCRPT1	1.1.0.0b	Discrepancies CPP Shared Utilities vol2 1.0.1.0 05142008.xls	8	Source\BlowFish.cpp	BlowfishEncryptBlock	"ct", "leftBlock" & "rightBlock" are not validated.	v.2: 5.4.2.e	Pointer validation	The parameter “ct” is defined as a pointer to a structure of the type “SBlowfishCipherTables”. If the functions are called with a pointer to any other data structure type, the compiler will generate an error. The “ct” parameter is implicitly verified by the compiler. The parameters “leftBlock” and “rightBlock” are commented as “32 bit blocks of code to be encrypted/decrypted. The programmer will insure that these will only point to data that is to be encrypted or decrypted.
CPP	ESSCRPT1	1.1.0.0b	Discrepancies CPP Shared Utilities vol2 1.0.1.0 05142008.xls	9	Source\BlowFish.cpp	BlowfishDecryptBlock	"ct", "leftBlock" & "rightBlock" are not validated.	v.2: 5.4.2.e	Pointer validation	The parameter “ct” is defined as a pointer to a structure of the type “SBlowfishCipherTables”. If the functions are called with a pointer to any other data structure type, the compiler will generate an error. The “ct” parameter is implicitly verified by the compiler. The parameters “leftBlock” and “rightBlock” are commented as “32 bit blocks of code to be encrypted/decrypted. The programmer will insure that these will only point to data that is to be encrypted or decrypted.

CPP	ESSCRPT1	1.1.0.0b	Discrepancies CPP Shared Utilities vol2 1.0.1.0 05142008.xls	10	Source\BlowFish.cpp	BlowfishEncrypt	kgw -- this method is the local method for the DLL exported function "EncryptData" which was not reviewed in the 3% review, but which also does not validate any pointers or state in its header that the input array is expected to be of size that is a multiple of the block size or else it will overflow. This method also assumes that input block length is a multiple of block size. Line 615 overflow occurs.	v.2: 5.4.2.d	Overflow	Blowfish is a “block encryption” algorithm. All block encryption algorithms work in terms of some block size. It is standard operating procedure to allocate buffer space in terms of the block size of whatever algorithm you are using. (Just the same as creating a “char array” one byte longer that the max number of characters so that there is room for the terminating NULL character.) Therefore the arrays will not overflow.
CPP	ESSCRPT1	1.1.0.0b	Discrepancies CPP Shared Utilities vol2 1.0.1.0 05142008.xls	11	Source\BlowFish.cpp	BlowfishDecrypt	kgw-although one would expect that a buffer for decryption would be a multiple of the size of a block, line 664 overflows the buffer because it was not checked in advance.	v.2: 5.4.2.d	Overflow	Blowfish is a “block encryption” algorithm. All block encryption algorithms work in terms of some block size. It is standard operating procedure to allocate buffer space in terms of the block size of whatever algorithm you are using. (Just the same as creating a “char array” one byte longer that the max number of characters so that there is room for the terminating NULL character.) Therefore the arrays will not overflow.
CPP	Election Packager	1.0.0.0e	Discrepancies CPP Shared Utilities vol2 1.0.1.0 05142008.xls	12	ElectionPackagerDlg.cpp	(file)	line 142 hard-coded password (used at lines 1974, 1987, 1999)	v1: 6.4.2	Hard coded password	The data structure being cited is the encryption key used when encrypting/decrypting files in the Unity system. This encryption key (note a password) CANNOT change because both the creation and receiving sizes of this feature must know the key to be used to encrypt and then decrypt the packaged data. This function is used when ES&S needs to acquire election definition data from clients that code their own elections and the respective election data is Zipped up with an encryption key to protect it while in transit for the client site to the ES&S support center. This is a copy of the election data and the 'master' copy of the election data remains resident on the client system.

CPP	VAT - GetMarks	1.4.9	Discrepancies CPP VAT (ESS ScannerPrinterLibrary 1.8.31-GetMarks 1.4.9) 10152008.xls	3	ESSNY.cpp	ipfEssNYGetXCompactedPixel	Line 223: validate input arguments and pointers	v.2: 5.4.2.a v.2: 5.4.2.e	Pointer validation	We need to keep this routine streamlined and optimized because it is a major bottleneck for the scanning algorithms. This routine is often called over a million times during a regular scan. If we add extra code to validate input arguments then this will greatly detract from the user's experience (it would take too long to perform a scan). The controls we provide to prevent the pointer from overwriting out of bounds locations are as follows: To be in bounds, vert must be between 0 and 5000 and horz must be between 0 and 2047. This is because the input buffer is always g_image which is a buffer of MAXSCANLINE (=5000) pointers pointing into FM->image_buffer (defined in GETMARKS.H) where each buffer is BYTES_PER_LINE=256 bytes (2048 pixels which allows horz to range from 0-2047). We ensure this is the case by simple code inspection of all the calls to ipfEssNYGetCompactedPixel. (Greater detail available upon request).
CPP	VAT - GetMarks	1.4.9	Discrepancies CPP VAT (ESS ScannerPrinterLibrary 1.8.31-GetMarks 1.4.9) 10152008.xls	4	ESSNY.cpp	ipfEssNYListToArray	Line 483: Provide control for assigned array indexed pointer	v.2: 5.4.2.e	Pointer validation	Justification: the control is built into the way the loops work in that routine that utilize that array index. Every pass through the loop, which increments the array index (which starts at 0), the next entry in the linked list pEssNyList is retrieved. If that next entry is NULL then the array index will not increase any further. Therefore, the array index is constrained by the number of entries in pEssNyList. The number of entries in pEssNyList is definitely equal to the number of calls to ipfEssNyListAdd following a call to ipfEssNyListClear. Also, the size of the array is equal to numTm in all the calls to ipfESSNYListToArray. So we simply need to trace all calls of ipfEssNyListAdd and confirm the number of calls after an ipfEssNyListClear is <= the value of numTm in the call to ipfEssNYListToArray. (Greater detail available upon request).

SQL	AIMS	1.3.54	Discrepancies SQL AIMS 1.3.54 08062007.xls	1	dbo.AuditLogAddItem.PRC	CREATE procedure AuditLogAddItem (@strEventName varchar(100),	line 42 audit logging optional. Numerous places in code (4) turn off audit logging and do not transact the unit operation that would include turning it back on. The specific procedures are IFCImport, IGImport, IUnityImport & spXMLImport. Additional information: no related trigger found.	v.1: 4.2.3.e	Single exit	The audit logging is turned off explicitly and only at the start of a massive election automated import (from a standard import file set of one sort or another -- be it XML or AIS or AccuVote MDB or whatever). Logging is turned back on after that import is completed. If the import process fails, the audit log will contain entries of this failure and guide the user to a resolution of the issue.
SQL	AIMS	1.3.54	Discrepancies SQL AIMS 1.3.54 08062007.xls	2	dbo.BallotRaceAdd.PRC	CREATE procedure BallotRaceAdd (@intBallotID int,	line 49 multiple database inserts not transacted	v.1: 4.2.3.e	Single exit	We do not believe Volume 1, Section 4.2.3.e requries the implementation of database start / end transaction processing when applications are performing database updates. All database updates, other than mass updating of tables from the import of election data from an external resouce (as described in the issue above) are captured in the AIMS audit log. Any data I/O that fails to complete successfully is so noted back to the AIMS user and is recorded in the audit log for corrective action.
CPP	AutomarkEncoder	1.0.105	Shared Discrepancies:	5	AutomarkEncoder 1.0.105\AutomarkEncoder.cpp	makekey	1) Parameters "realkeyforward", and "realkeyfinal" are not validated. 2)Line 1425: Need explicit comparison in if() statement. 3) Constants other than "0" or "1" enumerated or defined: 7	v.2: 5.4.2.e v.2: 5.4.2.t v.2: 5.4.2.u	Pointer validation	
VBA	AIMS	1.3.552	Shared Discrepancies:	3	Election.cls	Election.RefreshProperties	Line 802: Need explicit return in "Failure:"	v.2: 5.4.2.b	Explicit return	