Source Code Assessment of the CMI Intoxilyzer 9000 Instrument

Drafted For:
State of Georgia
Criminal Justice Coordinating Council

20 September 2013

Principal Investigator:
Andrew Howard
Senior Research Scientist
Georgia Tech Research Institute
andrew.howard@gtri.gatech.edu
### Executive Summary

The Intoxilyzer 9000

Georgia Tech

Source Code Review Objectives

### Testing Methods and Approach

Overview

Physical Evaluation

Source Code Assessment

Findings

### Appendix

Appendix A: Cobra Analysis

Appendix B: Bibliography

Appendix C: Source Code Reference
Executive Summary

The State of Georgia Criminal Justice Coordinating Council (CJCC) engaged the Georgia Tech Research Institute (GTRI) to evaluate the Intoxilyzer 9000 device, manufactured by CMI Incorporated. The objective of this study was to review the source code of the Intoxilyzer 9000 for potential security vulnerabilities, non-conformance with product requirements or best practices, and code segments that might lead to an inaccurate reading during a breath alcohol test. CMI hosted GTRI personnel for a source code review of the Intoxilyzer 9000 over three weeks between July 2013 and September 2013.

The findings regarding the Intoxilyzer 9000 and its related source code are:

- Device appears to operate as expected and GTRI identified no practical flaws that inhibit its functionality
- Device appears to appropriately handle breath alcohol values throughout the source code
- Device has minor semantic flaws in source code that should be corrected to adhere to industry best practices for software development but these flaws do not appear to impact the results the device produces
- Device is susceptible to compromise from attacks that require physical access to the device

Herein we provide a more detailed evaluation of the Intoxilyzer 9000 source code produced by CMI. Only the Intoxilyzer 9000 source code identified in Appendix C was evaluated. All evaluations were conducted within CMI facilities and Georgia Tech does not possess any source code. Any previous versions, including but not limited to, the Intoxilyzer 5000 and Intoxilyzer 8000, were not evaluated. Future versions or deviations from examined source code are not inherently covered by this report. The assumptions made herein are valid only for the Intoxilyzer 9000 source code identified in Appendix C.

This study only focused on Intoxilyzer 9000 source code related to transferring calculated breath alcohol data from the device to the user interface. As this was a source code review, this study assumed that breath alcohol values returned by device infrared spectrometry were correct. Additionally, the method and accuracy of the mathematical calculations regarding breath alcohol testing contained in the source code were not within the scope of this study. These calculations require review by experts in infrared spectrometry and breath alcohol. This study does not represent a guarantee that the reviewed Intoxilyzer 9000 source code is secure, will properly operate, or properly handle breath alcohol values, only a best effort attempt at a review.

Evaluation methods and standards were chosen according to the “Technical Guide to Information Security Testing and Assessment” published by the National Institute of Standards and Technology (NIST) and the “CERT C++ Secure Coding Standard” published by the United States Computer Emergency Readiness Team (US-CERT). The characteristics evaluated were chosen based on principles and guidelines set forth in “Software Testing: Fundamental Principles and Essential Knowledge.” This study is only a listing of Intoxilyzer 9000 source code findings not a risk assessment. The risk and potential impact of the findings identified by this report should each be evaluated independently by the State of Georgia and CMI Inc. for potential remediation.

The Intoxilyzer 9000

The Intoxilyzer 9000 provides breath alcohol testing through the use of infrared spectrometry technology. The device operates through an on screen user interface (UI). During subject testing the device takes additional input provided by the subject breathing into a tube. The device, using infrared spectrometry, then analyzes the collected breath sample. The UI guides the user and subject through the testing processing using on-screen prompts.

During this study, the Intoxilyzer 9000, serial number 90-000623, manufactured on 08/2013 was used for testing. Source code review was conducted on three software modules: a user interface module written in the C-sharp (C#) language, a scientific analysis written coded in the C-plus-plus (C++) language, and a device control module written in the C language.
The device contains two processors which are governed by the Windows CE operating system (version 6, 2006). The primary processor runs the user-interface and library modules. The secondary processor runs the proprietary device control module. The two processors communicate via a serial peripheral interface (SPI). CMI did not allow the source code or detailed notes pertaining to the source code to leave their facility. Given this limitation, this report only contains high-level observations from the source code review. Appendix C contains additional information necessary to identify the reviewed source code.

Georgia Tech

The Georgia Institute of Technology (Georgia Tech) is one of the nation’s preeminent public technological universities, ranking first in the nation for Research and Development at a public university. Georgia Tech maintains a robust presence in the information security research and development field and is designated as a National Security Agency Center for Academic Excellence in Research. Georgia Tech’s multi-dimensional organizational structure allows for agile research in both theoretical and applied spaces.

The Georgia Tech Research institute (GTRI) is the applied research unit of Georgia Tech. For the past 75 years, GTRI has been the non-profit engineering and applied research and development arm of Georgia Tech. GTRI has extensive capabilities and experience in the design, testing, and sustained engineering for technology systems. GTRI brings a key attribute to the management and development of programs, namely independence. Because of GTRI’s not-for-profit status and because GTRI is not a manufacturer, GTRI solutions are innovative, affordable, and in the best interest of the customer.

Source Code Review Objectives

The objective of this project was to review the source code of the Intoxilyzer 9000 for potential security vulnerabilities, non-conformance with product requirements or best practices, and code segments that might lead to an inaccurate reading during a breath alcohol test. Only the Intoxilyzer 9000 was reviewed.

The assessment examined the source code for the characteristics listed in Table 1 in addition to any obvious source code errors. The characteristics were chosen based on principles and guidelines set forth in “Software Testing: Fundamental Principles and Essential Knowledge.”

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casting and conversion errors</td>
<td>Statements that inappropriately convert the result between types such as converting floating point numbers to integers.</td>
</tr>
<tr>
<td>Mathematical precision</td>
<td>Calculations that cause precision loss due to operations on large numbers.</td>
</tr>
<tr>
<td>Mathematical range errors</td>
<td>Statements which inappropriately use math functions with defined ranges (e.g. square root function).</td>
</tr>
<tr>
<td>Mathematical exceptions</td>
<td>Division by zero or operations on NaNs (not-a-number).</td>
</tr>
<tr>
<td>Input validation</td>
<td>Validating untrusted inputs are properly checked and formatted for use within code.</td>
</tr>
<tr>
<td>Sanity checks</td>
<td>Examining code pathways and functions to ensure that pre-conditions are met.</td>
</tr>
<tr>
<td>Bounds checks</td>
<td>Examining loop constructs, arrays, and other collections for conditions that may operate outside the boundaries of the specific resource.</td>
</tr>
<tr>
<td>Password handling</td>
<td>Checking that passwords were properly stored and handled.</td>
</tr>
<tr>
<td>Encryption algorithms</td>
<td>Checking for insecure (or not proven secure) encryption, checking for</td>
</tr>
</tbody>
</table>
proven secure encryption (AES, 3DES, etc.) being used in accordance with secure guidelines.

Testing Methods and Approach

Overview

The overall assessment was conducted according to guidelines set forth by the following agencies: the National Institute of Standards and Technology (NIST), the United States Computer Emergency Readiness Team (US-CERT), Mitre (MITRE), and the National Vulnerability Database (NVD). During the evaluation the standards set forth by these institutions were the guiding factors in determining the severity of potential weakness within the code. The referenced publications can be found in Appendix B: Bibliography.

Throughout this review the reader will encounter Common Weakness Enumeration (CWE) references. These references were chosen by researchers to represent the weakness or flaw that the code exhibited during that phase of evaluation. CWEs "provide a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design" [1].

Assessment activities were conducted using a three-step approach:

Step 1: Source Code Familiarization

GTRI first familiarized themselves with the source code and the architecture of the Intoxilyzer 9000. During this process, GTRI attempted to understand the organization of the source code and how different pieces of the source code interoperate. GTRI identified relevant source code files and segments for further analysis.

Step 2: Automated Review

GTRI then conducted an automated review of the source code. Automated review is a method of reviewing source code through the use of automated tools. GTRI utilized commercial-off-the-shelf (COTS) tools to identify potential software vulnerabilities in an automated fashion. The tools which were selected are noted in the Source Code Assessment section. The advantage of automated tools is that they can review a large amount of source code quickly; however, false-positives are common. After automated tools identified potential issues, a human review was conducted.

Step 3: Manual Review

In the final review step, GTRI reviewed relevant aspects of the source code manually. During this step, GTRI researchers walked through each line of code that handles breath alcohol values. This review did not include a validation of the methods or calculations used to obtain breath alcohol values.

Physical Evaluation

The device was tested using both standard usage and abusive usage.

Under standard use, the device was operated in accordance with its operating parameters, including using approved peripherals such as USB keyboards, USB network, and Ethernet connections. During this testing, on-screen instructions were followed when operating the device. Primarily, instructions presented for using the breathing apparatus were followed to get a valid result.
Under abusive use, the software was subject to any (non physical) means necessary to coerce an incorrect result. This includes overflow attacks on input fields, improper data formats, attempting to inject SQL commands through various input fields, connecting both approved and unapproved peripherals, and using the breathing apparatus in a way that was inconsistent with normal breathing (or using external air sources).

Under both testing scenarios, connectivity tests were limited to plug-and-play (PnP) functionality common to many operating systems.

**Source Code Assessment**

Software vulnerability assessment consists of examining code, especially data input pathways, where that data can be used in a malicious way to coerce the code to perform incorrectly. Per the methodology, automated software tools including CppLint and FxCop were used to generate an analysis of the code. CppLint is a C/C++ static analysis tool which analyzes code for both style errors (poorly named variables, poorly indented code, poorly scoped variables, etc.) and for typical coding mistakes (variables used before initialization, buffer overflows, memory leaks, etc.). FxCop is a similar tool for the C# language; it performs both static and dynamic analysis of source code and will report similar errors to that of CppLint. The results of the investigation included herein contain results obtained from these tools and verified by GTRI researchers.

Semantic assessment consisted of a general overview of language constructs to determine if subtle flaws existed that are commonly overlooked. These flaws exist because the code may express the intent or idea correctly but the implementation cannot work due to physical limitations of the hardware, inaccurate representation of the semantic construct, or an overlooked flaw. For example absolute value functions fail when converting maximum negative integers to positive integers; the negative range contains one more number that cannot be properly reflected on the positive range.

Security assessment consisted of examining protocols surrounding access to critical device functions. This included passwords and access to administrative functions, encryption schemes used for data handling, and network protocols used for updating firmware or download information from the device. This assessment is to ensure that proper handling of sensitive information and proper protection of critical functions are employed.

**Findings**

During the source code review, six potential vulnerabilities were identified. The findings indicate a potential only—no finding led to a successful exploitation (adverse manipulation of the device or its output). This does not indicate the device is free from all exploitation. These findings included possibilities for SQL injection, floating point exception, and buffer overflows.

**Finding 1) SQL Injection:** More than ten statements had improper handling of SQL variables and as such could possibly be allowed to contain data that could be misinterpreted by an SQL interpreter thus causing an exploit. In practice, all attempts to perform actual SQL injection were unsuccessful due to input field constraints such as limiting the number of characters allowed in the input data or validating that the data was in a specific format (i.e. date fields were entered in a MM/DD/YYYY format). (CWE-20, CWE-95)

† CWE-20: Improper Input Validation (The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program.)
† CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code (The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes code syntax before using the input in a dynamic evaluation call.)

**Finding 2) Buffer Overflow:** At least two buffer overflow conditions were identified within the code. The specific flaws were deemed non-critical and were not accessible from the user interface. The non-criticality of these flaws were determined based on the type of resources utilized during the overflow (constant strings), the static memory layout of
the device code, and that no adverse runtime affects were observed as a result of these overflows. (CWE-787, CWE-805)

† CWE-787: Out-of-bounds Write (The software writes data past the end, or before the beginning, of the intended buffer.)
† CWE-805: Buffer Access with Incorrect Length Value (The software uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.)

(Finding 3) Precision Loss: There were at least ten statements that could potentially lead to a round-off error during floating-point calculations. Numerous statements were identified within the analysis routines that used floating-point calculations that could result in a loss of precision. These calculations relied on the implicit assumption of valid numbers and/or numbers that would result in a valid number after the calculation is complete. Based on review, calculations throughout the source code preserve at least 8 decimals of precision under normal operating conditions. It is highly unlikely that this precision loss will impact any breath alcohol calculations. (CWE-349, CWE-681, CWE-739, CWE-885)

† CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data (The software, when processing trusted data, accepts any untrusted data that is also included with the trusted data, treating the untrusted data as if it were trusted.)
† CWE-681: Incorrect Conversion between Numeric Types (When converting from one data type to another, such as long to integer, data can be omitted or translated in a way that produces unexpected values. If the resulting values are used in a sensitive context, then dangerous behaviors may occur.)
† CWE-739: CERT C Secure Coding Section 05 - Floating Point (Weaknesses in this category are related to rules in the floating point section of the CERT C Secure Coding Standard.)
† CWE-885: SFP Cluster: Risky Values (This category identifies Software Fault Patterns (SFPs) within the Risky Values cluster.)

(Finding 4) Integer Overflow: There were statements that could potentially lead to an overflow error. More than ten calculations were witnessed which multiplied two or more integers and stored the result in a similar sized integer. Depending on the input of the two integers the resulting calculation may produce a number larger than what the result variable can store properly. During testing there were no numbers witnessed that would be this large to cause such an overflow; however, such conditions could not be eliminated as a possibility. Additionally, it is likely that an integer overflow will cause the device to error. (CWE-190)

† CWE-190: Integer Overflow or Wraparound (The software performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value. This can introduce other weaknesses when the calculation is used for resource management or execution control.)

(Finding 5) Sign/Unsigned Conversion: There was a superfluous use of casting signed integers to unsigned integers. The code made assumptions about the signed-ness of the integers without actually verifying that the variable was positive. During review, there were no negative numbers witnessed; however, such conditions could not be eliminated as a possibility. (CWE-195)

† CWE-195: Signed to Unsigned Conversion Error (A signed-to-unsigned conversion error takes place when a signed primitive is used as an unsigned value, usually as a size variable.)

(Finding 6) Weak Authentication: Users are authenticated by a user identifier and a short password. While passwords appear to be stored securely, the device uses a relatively weak authentication scheme that can be brute-forced or guessed. A privileged user account, who has access to all of the configuration pages of the device and can alter many of the fine aspects of the test, is not required to pass any additional authentication challenges, and can be compromised in the same manner as an unprivileged user.

By pressing a specific character sequence an operator can attempt to escape "Kiosk Mode" and break out of the Intoxilyzer 9000 application and into the Windows CE operating system. This requires a short password that is computed in a predictable fashion. Not only is the password predictable, but also a short password can be brute-forced. After escaping "Kiosk Mode", the operator has privileged access to the software executables and database contents. (CWE-521)
† CWE-521: Weak Password Requirements (The product does not require that users should have strong passwords, which makes it easier for attackers to compromise user accounts.)
Appendix

Appendix A: Cobra Analysis

The State of Georgia requested that GTRI review the Intoxilyzer 9000 network communication source code. The network communication capability of the Intoxilyzer 9000 is referred to as Cobra. The reviewed Intoxilyzer 9000 configuration for the State of Georgia did not have the Cobra functionality in use; however, the source code was available for review. As such, the following findings do not specifically apply to this study but are provided as requested.

(Cobra-1) Firmware Update: The device was found to have network based weakness when the device is used to update its firmware. During the update an encrypted zip file is transferred to the device and its contents are written, unverified, to the flash memory device. An attacker could decipher the proper filename and location of the executable and craft an updated zip file which replaces the current application. If an attack were properly executed, the application on the device could be overwritten with a malicious application. (CWE-327, CWE-347)

† CWE-327: Use of a Broken or Risky Cryptographic Algorithm (The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the exposure of sensitive information.)
† CWE-347: Improper Verification of Cryptographic Signature (The software does not verify, or incorrectly verifies, the cryptographic signature for data.)

(Cobra-2) Encryption: The device uses an unproven, breakable, encryption scheme to encrypt all network communication to and from the device. (CWE-326, CWE-327)

† CWE-326: Inadequate Encryption Strength (The software stores or transmits sensitive data using an encryption scheme that is theoretically sound, but is not strong enough for the level of protection required.)
† CWE-327: Use of a Broken or Risky Cryptographic Algorithm (The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the exposure of sensitive information.)

Appendix B: Bibliography


**Appendix C: Source Code Reference**

The secure hash algorithm (SHA) was chosen based on NIST recommendations and version 1 (SHA-1) was chosen based on the available tool set and installed components on the CMI-provided testing machines. Hashes are used to identify whether text has been altered—the inherent property of hash calculations are that the same identical text will produce the same hash value, alterations of the text (even minor) will produce significantly different hashes. The odds of a duplicate hash value being produced by two different texts is about 1 in 10 quadrillion (20 decimal digits of precision). The following hashes were taken from the source code and can be used to identify that such source has not been modified since this review. CMI did not allow source code or source code filenames to leave their facility.

<table>
<thead>
<tr>
<th>SHA-1 Hash (File Contents)</th>
<th>SHA-1 Hash (File Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d0323b00a6c868a6a27a584cae0e033a3fe04c</td>
<td>0ee4b97ae6b3a580ed08884c490214b7b27164</td>
</tr>
<tr>
<td>23284b57b49badab1cd10eae79f3c9a595549793a</td>
<td>1cf3f3db41c351d95dev66c8e9812b570b78741a3</td>
</tr>
<tr>
<td>6768b11194845674eb5cb60859bb251b83e630cf9</td>
<td>1f06dd04e60e7328c8821159ef43079f3f40a</td>
</tr>
<tr>
<td>5f0a111ae35990d1baa1159bd303ed6385f5de4</td>
<td>f21b0df03de4b1bb853077ba876cf3289dd95</td>
</tr>
<tr>
<td>b4d689fe726694d9871b5838f0f1cc778af3b1d86</td>
<td>13304e53a700a903ef0af28e1e1f674a57214</td>
</tr>
<tr>
<td>7e53553a56f4785b3ef4ade472bd8be37303878</td>
<td>75142544d1d47b68be96c104662827d05afe3b8</td>
</tr>
<tr>
<td>5886f712d38ae6a3dc343bb69fa0f0f31f8acc1</td>
<td>eee4238c7fff184aadd02b194679688eb560e0d3</td>
</tr>
<tr>
<td>1a9bb407a975cfb46e43d3c6aadd6bffdde1aa</td>
<td>15ed2a43f8ddc37f9f21e1ee651b96b73680c</td>
</tr>
<tr>
<td>86a6c2a4d54d0a5664a11f44d42bf87786e6e688</td>
<td>dcd6c3bb4227b1dabb8ef100d7e3ee69162b7631</td>
</tr>
<tr>
<td>54dbf32a1ab054f7db8e6c801b5f858d4c</td>
<td>bdd2b72939370f4fbed7074a508118f7e699656c</td>
</tr>
<tr>
<td>0fbd54fb3d1819138b8fe1892ccc9a3d082ba0</td>
<td>b4dc4514f6f29969bb57a9bba599e3185843dc</td>
</tr>
<tr>
<td>5de92994653818dfeff1779c310a494a0b52cd</td>
<td>bf6ee22b28c99e3068f7f77be502530d0dbca6</td>
</tr>
<tr>
<td>64cd1443430f29e119b327737f2326dd4079ea</td>
<td>98c8063a8e0ac580678290f807363975e094</td>
</tr>
<tr>
<td>5c0720d2a666f2b899d6029d43af75e85e6d179</td>
<td>d53e40d2c34ebbb034f8b28c0f3b725898ce3ce</td>
</tr>
<tr>
<td>1b51a4a8839c17473e17170eef150b6eeb3b4084</td>
<td>562bef3d112f4b8ce1f90c984628b8dca1b240</td>
</tr>
<tr>
<td>77b30484b2494b85d33af7f08825ff7251782ac</td>
<td>d913ccf3c2a1bd9029c92e66710ff8f4493d9b</td>
</tr>
</tbody>
</table>