How the CC Harmonizes with Secure Software Development Lifecycle

Sep. 11. 2013.

Jinseok Park (1st Author)
CI ST (Center for Information Security Technologies), Korea University
ilysoon7@korea.ac.kr

Seungjoo Kim (Corresponding Author)
CI ST (Center for Information Security Technologies), Korea University
skim71@korea.ac.kr
Overview

Problems
CC does not cover with a certified product’s zero-day attack after certifying it.
CC focuses on removing vulnerabilities.

Motivations
Removing weaknesses is very useful for time and cost for zero-day attack than removing vulnerability.
Secure software development lifecycle can minimize weaknesses for zero-day attack.

Conclusion
Harmonize the CC with the Secure software development lifecycle.
Definitions

- **(Software Security) Weakness**
  - A type of mistake in software
  - Bugs, Errors
  - Can be aggravated to (software security) vulnerabilities (i.e., Zero-day attacks)

- **(Software Security) Vulnerability**
  - An occurrence of a weakness (or multiple weaknesses) within software

- **Zero-day attack**
  - Weakness is exploited by hackers before the vendor becomes aware to fix it
Definitions

• $S$ : The set of all software in existence at some point in time
• $W$ : The set of all instance of software weaknesses in $S$
• $W_d$ : The set of discovered software weaknesses in $W$
• $W_{cwe}$ : The set of Identified with a CWE
• $V$ : The set of all vulnerabilities in $W$
• $V_d$ : The set of all discovered Vulnerabilities in $V$
• $V_{cve}$ : The set of Identified with a CVE
Motivations

- Software bugs or errors are so detrimental that they cost the U.S economy an estimated $59.5 billion annually. (GDP 0.6%)

- Errors requirements/design stage cost 1X to fix. But if it is not found until the post-product release stage, it costs 30 times more to fix.

<table>
<thead>
<tr>
<th>Requirements Gathering and Analysis/Architectural Design</th>
<th>Coding/Unit Test</th>
<th>Integration and Component/RAISE System Test</th>
<th>Early Customer Feedback/Beta Test Programs</th>
<th>Post-product Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>1X</td>
<td>5X</td>
<td>10X</td>
<td>15X</td>
<td>30X</td>
</tr>
</tbody>
</table>

[Reference : 1]
Motivations

- The top 10 software vendors have a patch remedy rate of just over 94% of all vulnerabilities disclosed.
- But, 47% of all vulnerabilities disclosed in 2012 remain without a remedy.
- A zero-day attack can still be thwarted by properly-patched software.
  - But they are not cost and time effective!
- Economically, many researchers have tried to remove the vulnerability in software
  - To remove weaknesses is very useful for time and cost.
Motivations

- If we can remove weaknesses, vulnerabilities and zero-day attack can also be removed.

- Thus, we are interested in removing design stage's and implementation stage's weaknesses.
  - It is very useful for time and cost to remove weaknesses
Problems

- The CC philosophy is that the threats to security and organisational security policy commitments should be clearly articulated and the proposed security measures be demonstrably sufficient for their intended purpose. [9]

- CC focuses on removing vulnerabilities.

- CC does not cover with a certified product’s zero-day attack after certifying it.
How to Fix It in a Nutshell

- **Software Assurance**
  - The level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software throughout the life cycle.

- **Secure Software Development Lifecycle**
  - Software Development Lifecycle + Software Assurance

- SSDLCs focus on removing weaknesses.

[Reference : 10]
CC and source code analysis tools are not rivals [11]

- They find different types of vulnerabilities
- If together, they can discover more common vulnerabilities types

**Design**

- Weak Audit
- I&A Vulnerabilities
- Inconsistent Access Control

**Security Mechanisms**

- TOCTTOU

**Implementation**

- XSS
- SQL Injection
- Buffer Overflow

**Other Areas**

- CC tools
How to Fix It in a Nutshell

- Based on CWE v2.4, CC v3.1, MS-SDL (one of the famous SSDLCs), static code analysis tools.
  - Dynamic analysis tools can remove limited weaknesses. [12]
Our Methods in Detail

- **MS-SDL (Microsoft Security Development Lifecycle)**
  - Software security assurance process
  - A mandatory policy since 2004

[Reference : 13-14]
Our Methods in Detail

- MS-SDL helps you build software, that's more secure by reducing the number and severity of vulnerabilities in your code.
Our Methods in Detail

- Consistent application of sound security practices during all phases of a development project will result in fewer vulnerabilities.

**Total Vulnerabilities Disclosed 12 Months After Release**

- **Windows® XP**
  - Before SDL: 119
  - After SDL: 66
  - 45% reduction in Vulnerabilities

- **OS I**
  - Before SDL: 400
  - After SDL: 242

- **OS II**
  - Before SDL: 157

- **OS III**

**Total Vulnerabilities Disclosed 36 Months After Release**

- **SQL Server® 2000**
  - Before SDL: 34
  - After SDL: 3
  - 91% reduction in Vulnerabilities

- **SQL Server 2005**
  - Before SDL: 187
  - After SDL: 3

**Competing commercial DB**

[Reference : 14]
Our Methods in Detail

- **Static code analysis tools**
  - Analyze source code and/or compiled version of code in order to help find security flaws (weaknesses)

- **Certificate of CWE compatibility (5 product) [15]**
  - CodeSonar, Covertiy Quality Advisor/Security Advisor, HP Fortify Static Code Analyzer, Klocwork Insight
Our Methods in Detail

- Four different areas:
  1. Design (CWE-701)
  2. Implementation (CWE-702)
  3. Security mechanisms (CWE-254)
  4. Other parts (non-security mechanisms)
Our Methods in Detail

- **Total weaknesses: 920 entries, 8 types**

  - View: 29 entries
  - Category: 176 entries
  - Weakness - Class: 88 entries
  - Weakness - Base: 330 entries
  - Weakness - Variant: 276 entries
  - Compound Element - Composite: 6 entries
  - Compound Element - Named Chain: 3 entries
  - Deprecated: 12 entries

Selected 703 entries
### Our Methods in Detail

- For example, CWE/ SANS TOP 25

<table>
<thead>
<tr>
<th>Rank</th>
<th>CWE Type</th>
<th>CWE-ID : Name</th>
<th>Design</th>
<th>Implementation</th>
<th>Security Mechanisms</th>
<th>Static Code Analysis Tools</th>
<th>CVE Entry</th>
<th>MS-SDL Design</th>
<th>IMplementation</th>
<th>Verification</th>
<th>SFR</th>
<th>SAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Base</td>
<td>CWE-89 : SQL Injection</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>7</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Base</td>
<td>CWE-78 : OS Command Injection</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>10</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Base</td>
<td>CWE-120 : Classic Buffer Overflow</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>5</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Base</td>
<td>CWE-79 : Cross-site Scripting</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>11</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Variant</td>
<td>CWE-306 : Missing Authentication for Critical Function</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>3</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Class</td>
<td>CWE-862 : Missing Authorization</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>19</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Base</td>
<td>CWE-798 : Use of Hard-coded Credentials</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>10</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Base</td>
<td>CWE-311 : Missing Encryption of Sensitive Data</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>20</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Base</td>
<td>CWE-434 : Unrestricted Upload of File with Dangerous Type</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>10</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Base</td>
<td>CWE-807 : Reliance on Untrusted Inputs in a Security Decision</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>5</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Reference : 16]
Our Methods in Detail

For example, CWE/ SANS TOP 25

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Class</td>
<td>CWE-250 : Execution with Unnecessary Privileges</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Composite</td>
<td>CWE-352 : Cross-Site Request Forgery(CSRF)</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Class</td>
<td>CWE-22 : Path Traversal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>Base</td>
<td>CWE-494 : Download of Code Without Integrity Check</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Class</td>
<td>CWE-863 : Incorrect Authorization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>Class</td>
<td>CWE-829 : Inclusion of Functionality from Untrusted Control Sphere</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>Class</td>
<td>CWE-732 : Incorrect Permission Assignment for Critical Resource</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>Base</td>
<td>CWE-676 : Use of Potentially Dangerous Function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>Base</td>
<td>CWE-327 : Use of a Broken or Risky Cryptographic Algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>20</td>
<td>Base</td>
<td>CWE-131 : Incorrect Calculation of Buffer Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>21</td>
<td>Base</td>
<td>CWE-307 : Improper Restriction of Excessive Authentication Attempts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>22</td>
<td>Variant</td>
<td>CWE-601 : Open Redirect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>Base</td>
<td>CWE-134 : Uncontrolled Format String</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>Base</td>
<td>CWE-190 : Integer Overflow or Wraparound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>Base</td>
<td>CWE-759 : Use of a One-Way Hash without a Salt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
Analyses

- Divided into four areas (Design, Implementation, Security mechanism, Non-Security mechanism)
- Distribution of weakness and vulnerabilities in each area
Analyses - CC -

-Design:
  - Non-Security Mechanism: 481<1458>
  - Security Mechanism: 73<241>

-Implementation:
  - Non-Security Mechanism: 240<857>
  - Security Mechanism: 110<387>

※ CWE<CVE>
Analyses - Static Code Analysis Tools -

[Diagram showing the analysis of static code analysis tools with axes for Design and Implementation, and triangles representing different categories such as Non-Security Mechanism, Security Mechanism, and Static Code Analysis Tools.]

※ CWE<CVE>
Analyses - CC & Static Code Analysis Tools -

Design

Non-Security Mechanism
- 139<447>
- 69<287>
- 295<894>
- 161<461>

Security Mechanism
- 19<40>
- 84<295>
- 14<26>
- 56<207>

Implementation

Static Code Analysis Tools
- MS-SDL
- CWE<CVE>
# How to Harmonize CC with SSDLC

- **Proposed Security Assurance Requirements (SAR)**
- **Now CC + SSDLC’s practice**

<table>
<thead>
<tr>
<th>SSDLC Process</th>
<th>Practice</th>
<th>CC Security Assurance Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Training</td>
<td>Core Security Training</td>
<td>ALC_DVS</td>
</tr>
<tr>
<td>2. Requirements</td>
<td>Establish Security and Privacy Requirements, Create Quality Gates/Bug Bars, Perform Security and Privacy Risk Assessments</td>
<td>ASE ALC_TAT AVA</td>
</tr>
<tr>
<td>3. Design</td>
<td>Establish Design Requirements, Attack Surface Analysis/Reduction, Use Threat Modeling</td>
<td>ADV AVA</td>
</tr>
<tr>
<td>4. Implementation</td>
<td>Use Approved Tools, Deprecate Unsafe Functions, Perform Static Analysis</td>
<td>ATE ADV_IMP</td>
</tr>
<tr>
<td>5. Verification</td>
<td>Perform Dynamic Analysis, Fuzz Testing, Attack Surface Review</td>
<td>ATE</td>
</tr>
</tbody>
</table>
Conclusion

- The CC and the SSDLC are similar methodologies for removing vulnerabilities.
  - But they find different types of vulnerabilities.

- Static code analysis tools can help removing weaknesses in CC

- The CC and the SSDLC are not competitors. Rather, they are complements.
References

Thank you

ilysoon7@korea.ac.kr
Jinseok Park
E-mail : ilysoon7@korea.ac.kr
Facebook : @ilysoon7

Jinseok Park received his B.S. (2010) in computer science from Soongsil University in Korea. Also, he served as electronic signature and authentication team of the Korea Information Security Agency (KISA) for 1 year. Now he is enrolled in the M.S. at Korea University. His research interests include security evaluation, information security and online social network service security. He has many certificates (CISSP, CISA, Security Product Evaluator, CPPG, Information Processing Engineer).
Seungjoo Kim (Corresponding Author)
E-mail: skim71@korea.ac.kr
Homepage: www.kimlab.net
Facebook, Twitter: @skim71

Prof. Seungjoo Kim received his B.S. (1994), M.S. (1996), and Ph.D. (1999) in information engineering from Sungkyunkwan University (SKKU) in Korea. Prior to joining the faculty at Korea University (KU) in 2011, He served as Assistant & Associate Professor of School of Information and Communication Engineering at SKKU for 7 years. Before that, He served as Director of the Cryptographic Technology Team and the (CC-based) IT Security Evaluation Team of the Korea Information Security Agency (KISA) for 5 years. Now he is Full Professor of Graduate School of Information Security at KU, and a member of KU's Center for Information Security Technologies (CIST). Also, He has served as an executive committee member of Korean E-Government, and advisory committee members of several public and private organizations such as National Intelligence Service of Korea, Digital Investigation Advisory Committee of Supreme Prosecutors' Office, Ministry of Justice, The Bank of Korea, ETRI(Electronic and Telecommunication Research Institute), and KISA, etc. His research interests include cryptography, information security and information assurance.
The corresponding author (Seungjoo Kim) acknowledges the support of the IT R&D program (10043959, Development of EAL 4 level military fusion security solution for protecting against unauthorized accesses and ensuring a trusted execution environment in mobile devices) of KEIT/MOTIE and MSIP(Ministry of Science, ICT & Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (NIPA-2013-H0301-13-1003) supervised by the NIPA(National IT Industry Promotion Agency)