The Role of Static Analysis in Secure Software Development

Tin Aung Win
President, (ISC)² Singapore Chapter
Agenda

• What is secure software?
  • What is static analysis?
  • Why static analysis?
  • Where does static analysis fit in a SDLC?
• Static analysis tools – Pros and Cons
• Conclusion
What is considered Secure Software?

My Definition:

Software that satisfies all of the following (non-exhaustive) criteria:

• It functions as intended
• Possess & performs only needed functionality
• Uses only needed facilities
• Behaves correctly in the presence of malicious attacks
• Built with incidence response in mind
• It fails safely
  • Not an easy feat...exceptions, race conditions, concurrency, ....
• It is resilient
• It can (probably) defend itself
Why do we need Secure Software?

- Humanity heavily relies on software (unfortunately)
- Insecure software can lead to any or all of the following:
  - From loss of personal information to state secrets
  - From e-robbery to destruction of financial operations
  - From inconveniences to infrastructure damages
  - From business disturbance to ruining the business ecosystem
  - From free speech advocates to hacktivism
  - From harming individuals to cyber war between nation states
Flame: A glimpse into the future of war

The most sophisticated cyberweapon yet unleashed.

Exploit:JS/Blacole

Description: Blacole, also known as the "Blackhole" exploit pack, is found on a compromised server and is installed there by an attacker by using one of many attack methods to gain access to the affected server.

Published Date: Mar 21, 2012

Alert level: Severe

Crippling Stuxnet virus infected Chevron's network too

Sophisticated virus was intended to knock offline an Iranian nuclear enrichment facility but strayed from its intended target about two years ago.

Duqu: New data-stealing Trojan could be Stuxnet version 2.0

Symantec says threat could be precursor to attacks on industrial control systems much like Stuxnet was.

• Three primary causes:
  • abuse, misuse
  • misconfiguration
  • bad Code
Bad Code – an example?

Spear Phishing – an example
Attacker sends emails to groups of people with specific common characteristics.
• Spear phishing emails appear to come from a trusted source
• Recipient clicked the malicious linked embedded in email.
• Exploit code downloaded and installed in the recipient’s system!

Sound really simple and easy – but is it?

Spear phishing led to DNS attack against the New York Times, others

Lucian Constantin, IDG News Service
Aug 26, 2013 7:55 AM

The cyberattack that resulted in nytimes.com and some other high-profile websites being inaccessible to a large number of users Tuesday started with a targeted phishing attack against a reseller for Melbourne IT, an Australian domain registrar and IT services company.

The attack resulted in hackers changing the DNS (Domain Name System) records for several domain names including nytimes.com, sharethis.com, huffingtonpost.co.uk, twitter.co.uk and twimg.com—a domain owned by Twitter—Jaime Blasco, director of the research lab at security firm AlienVault, said Tuesday in a blog post.

This resulted in traffic to those Websites being temporarily redirected to a server under the attackers’ control.
Software eco-system is Changing...

• So is the attack and threat landscape ...
• Internet of Things (IOT)
• Smart consumer and industrial devices
  • Mobile, Embedded, etc.
• Cloud
• Massively parallel processor systems (eg: GPGPU)
Industry Surveys
The most Serious types of Cyber Attacks

Source: Dark Intelligence and Cyber Security Report, Ponemon Institute LLC, Jul 2013
Industry Surveys
Most threatening Cyber attack

Source: Dark Intelligence and Cyber Security Report, Ponemon Institute LLC, Jul 2013
Surveyed 708 IT and IT security practitioners in the US with respondents have 11 years of relevant experience
Agenda

• What is secure software?
• What is static analysis?
  • Why static analysis?
  • Where does static analysis fit in a SDLC?
• Static analysis tools – Pros and Cons
• Conclusion
What is Static Analysis?

• The analysis of computer software source code, intermediate code or binary object code, that is performed without actually executing source programs

• Usually perform using automated tools and (and follows by) manual code review by experienced language & domain experts/reviewers

• Essential element of white Box/Glass box /Structural Testing

• Carry out by developer /quality engineer/Architect (Techs)
Backgrounder – Static Analysis Tools

• It first appeared as deep syntax checker - identify dead code, unused variables/registers and etc.
• 1970 - Researches in program analysis kicked off and funded by DARPA
• 1980 - Open source tools
  • Once upon in time, there was Lint.........
• 1990 - Commercial tools started to appear (with huge price tag....)
• 2000 - Byte code analysis and binary analysis tools showed up
• 2010 - Compilers follow suit
• Today - Increasingly become standard repertoire for developers as part of the SDLC and ALM environment
• And today, Lint is still alive & kicking with Android Java platform with more capability - potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.
How Static Analysis Works?

A “Generic” Static Analysis Process

Source Code → Build Model
- Data Flow
- Control Flow

Model → Perform Analysis
- Local
- Global

Control flow analysis
Data flow analysis
Coding standard compliance
Code metrics, etc.

Present Results → Review
Updatable Rules
+ Knowledge
- Language Specific
- Security patterns
Agenda

• What is secure software?
• What is static analysis?
• Why static analysis?
  • Where does static analysis fit in a SDLC?
  • Static analysis tools – Pros and Cons
• Conclusion
Why Static Analysis?
Cost of Fixing Critical Defects
Why Static Analysis?

• Enforce coding policies guidelines
  • Naming/Coding Conventions, Deprecated function calls, Version Numbering, User Interface Guidelines, Cryptographic standards, etc.

• Automatic generation of code metrics to assess code quality
  • Depth of Inheritance Tree, Number of derived child classes per base class, Lines of Code, Number of Input Parameters, Number of Return Points, Interface Complexity (Parameters + Returns), Cyclomatic Complexity, Coupling, Cohesion, and many more....

• Discover potential functional & security issues
  • Buffer overflow, Cross site scripting, SQL injection, etc.
Example: Coding Policies and Guidelines
Example: Code Metrics *(Parasoft)*

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Rule Id</th>
<th>Severity</th>
<th>Level</th>
<th>Acceptable ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coupling Between Objects</td>
<td>METRIC.CBO</td>
<td>4 - Low</td>
<td>type</td>
<td>lower than 30</td>
</tr>
<tr>
<td>Direct Types Used in Classes</td>
<td>METRIC.ELC</td>
<td>3 - Medium</td>
<td>type</td>
<td>lower than 20</td>
</tr>
<tr>
<td>Documentation Comment Ratio</td>
<td>METRIC.DCR</td>
<td>3 - Medium</td>
<td>type</td>
<td>lower than 40</td>
</tr>
<tr>
<td>Documentation Inherited Comment Ratio</td>
<td>METRIC.DICR</td>
<td>3 - Medium</td>
<td>type</td>
<td>greater than 40</td>
</tr>
<tr>
<td>Documentation True Comment Ratio</td>
<td>METRIC.DTCR</td>
<td>3 - Medium</td>
<td>type</td>
<td>greater than 40</td>
</tr>
<tr>
<td>Documentation True Inherited Comment Ratio</td>
<td>METRIC.DTICR</td>
<td>3 - Medium</td>
<td>type</td>
<td>greater than 40</td>
</tr>
<tr>
<td>Essential Complexity</td>
<td>METRIC.EC</td>
<td>2 - High</td>
<td>method</td>
<td>lower than 15</td>
</tr>
<tr>
<td>Fan Out</td>
<td>METRIC.DO</td>
<td>4 - Low</td>
<td>type</td>
<td>greater than 25</td>
</tr>
<tr>
<td>Implementation Comment Ratio in Classes</td>
<td>METRIC.DICR</td>
<td>4 - Low</td>
<td>type</td>
<td>greater than 25</td>
</tr>
<tr>
<td>Implementation True Comment Ratio in Classes</td>
<td>METRIC.DTCR</td>
<td>4 - Low</td>
<td>type</td>
<td>greater than 25</td>
</tr>
<tr>
<td>Inheritance Depth</td>
<td>METRIC.ID</td>
<td>5 - Lowest</td>
<td>type</td>
<td>greater than 7</td>
</tr>
<tr>
<td>Lack Of Cohesion</td>
<td>METRIC.LCOM</td>
<td>4 - Low</td>
<td>type</td>
<td>lower than 0.9</td>
</tr>
<tr>
<td>Maintainability Index</td>
<td>METRIC.NP</td>
<td>3 - Medium</td>
<td>type</td>
<td>greater than 65</td>
</tr>
<tr>
<td>Maximum Nested “if” Statement Depth in Methods</td>
<td>METRIC.MIF</td>
<td>3 - Medium</td>
<td>type</td>
<td>lower than 30</td>
</tr>
<tr>
<td>Maximum Nested “try” Statement Depth in Methods</td>
<td>METRIC.MITTRY</td>
<td>3 - Medium</td>
<td>type</td>
<td>lower than 30</td>
</tr>
<tr>
<td>Maximum Number Of Depth Levels</td>
<td>METRIC.MNOL</td>
<td>2 - High</td>
<td>type</td>
<td>greater than 25</td>
</tr>
<tr>
<td>Maximum Size Of Operation</td>
<td>METRIC.MSOO</td>
<td>2 - High</td>
<td>type</td>
<td>greater than 25</td>
</tr>
<tr>
<td>McCabe Cyclomatic Complexity</td>
<td>METRIC.VG</td>
<td>2 - High</td>
<td>type</td>
<td>greater than 25</td>
</tr>
<tr>
<td>Nejmeh NPath Complexity</td>
<td>METRIC.NP</td>
<td>2 - High</td>
<td>type</td>
<td>greater than 25</td>
</tr>
<tr>
<td>Nested Blocks Depth</td>
<td>METRIC.NBD</td>
<td>2 - High</td>
<td>type</td>
<td>greater than 25</td>
</tr>
<tr>
<td>Number Of “private” Fields</td>
<td>METRIC.NPRIF</td>
<td>2 - High</td>
<td>type</td>
<td>greater than 25</td>
</tr>
</tbody>
</table>

**Quality Tasks**

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Sum</th>
<th>Number of Items</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Maximum</th>
<th>Extreme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Essential Complexity</td>
<td>51</td>
<td>2.451</td>
<td>4.317</td>
<td>28</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>McCabe Cyclomatic Complexity</td>
<td>51</td>
<td>3.863</td>
<td>5.006</td>
<td>29</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>Number Of Children</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Number Of Code Lines in Methods</td>
<td>747</td>
<td>51</td>
<td>14.647</td>
<td>24.03</td>
<td>126</td>
<td>126</td>
</tr>
<tr>
<td>Number Of Parameters</td>
<td>4</td>
<td>0.882</td>
<td>1.078</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Robert Martin Afferent Couplings</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Robert Martin Efferent Couplings</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Example: Cyclomatic Complexity Metric

• Measures the amount of decision logic in a single software module.
• It gives the number of recommended tests for software.
• It is used during all phases of the software lifecycle.
• Based on the structure of software’s control flow graph.
## Some Open Source Applications Metrics

<table>
<thead>
<tr>
<th>Projects</th>
<th>No. of Modules</th>
<th>V(G)</th>
<th>V(G) &gt; 10</th>
<th>V(G) Avg</th>
<th>V(G) Max</th>
<th>EV(G)</th>
<th>EV(G) &gt;4</th>
<th>EV(G) Avg</th>
<th>EV(G) Max</th>
<th>Module #</th>
<th>% Code Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis2</td>
<td>8240</td>
<td>25,417</td>
<td>356</td>
<td>3.08</td>
<td>80</td>
<td>14,296</td>
<td>881</td>
<td>1.73</td>
<td>67</td>
<td>12,875</td>
<td>29.24%</td>
</tr>
<tr>
<td>Commons: beanutils</td>
<td>1194</td>
<td>2.943</td>
<td>25</td>
<td>2.46</td>
<td>44</td>
<td>1,795</td>
<td>38</td>
<td>1.50</td>
<td>22</td>
<td>176</td>
<td>35.62%</td>
</tr>
<tr>
<td>Commons: collections</td>
<td>3757</td>
<td>6,816</td>
<td>3165</td>
<td>1.81</td>
<td>29</td>
<td>4,887</td>
<td>70</td>
<td>1.30</td>
<td>23</td>
<td>3816</td>
<td>77.71%</td>
</tr>
<tr>
<td>Commons: digester</td>
<td>585</td>
<td>1893</td>
<td>4</td>
<td>2.07</td>
<td>35</td>
<td>1310</td>
<td>8</td>
<td>1.32</td>
<td>8</td>
<td>247</td>
<td>60.36%</td>
</tr>
<tr>
<td>Commons: discovery</td>
<td>269</td>
<td>554</td>
<td>3</td>
<td>2.06</td>
<td>13</td>
<td>371</td>
<td>10</td>
<td>1.38</td>
<td>1</td>
<td>380</td>
<td>37.78%</td>
</tr>
<tr>
<td>Commons: logging</td>
<td>280</td>
<td>703</td>
<td>9</td>
<td>2.51</td>
<td>41</td>
<td>377</td>
<td>11</td>
<td>1.35</td>
<td>7</td>
<td>292</td>
<td>49.38%</td>
</tr>
<tr>
<td>Commons: lang</td>
<td>2121</td>
<td>5624</td>
<td>43</td>
<td>2.65</td>
<td>55</td>
<td>3,645</td>
<td>159</td>
<td>1.72</td>
<td>41</td>
<td>1011</td>
<td>86.52%</td>
</tr>
<tr>
<td>Struts</td>
<td>4756</td>
<td>8653</td>
<td>81</td>
<td>1.82</td>
<td>54</td>
<td>5894</td>
<td>116</td>
<td>1.24</td>
<td>21</td>
<td>4063</td>
<td>29.52%</td>
</tr>
</tbody>
</table>
Example: Rules (Vstudio)

<table>
<thead>
<tr>
<th>Rules</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Globalization Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Interoperability Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Maintainability Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Mobility Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Naming Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Performance Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Portability Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Reliability Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Security Rules</td>
<td>Warning</td>
</tr>
<tr>
<td>Usage Rules</td>
<td>Warning</td>
</tr>
</tbody>
</table>
Example: A rule for Deprecation API

```c
PyObject *ctypes_callproc(....){
    Py_ssize_t i, n, argcount, argtype_count;
    struct argument *args, *pa;
    .......
    args = (struct argument *)alloca(sizeof (struct argument) * argcount);
    if (!args) {
        PyErr_NoMemory();
        return NULL;
    }
    .......
    memset(args, 0, sizeof(struct argument) * argcount);
}
```

Unprotected use of alloca _alloca indicates failure by raising a stack overflow exception. Use _malloc instead.

(part of python-3.3.2)
Static Analysis & Reporting: (Example) Fortify 360

![Fortify Report](image)

- **Summary**: Build ID: [Month 2014] Scanned: 149 files, 8,730 LOC (Executable)
- **Warnings**: 3 occurred during scan
- **Total Issues**: 128

### All issues by Folder

- **Critical (7)**
- **High (15)**
- **Medium (19)**
- **Low (80)**

**Abstract:**
The method `_RenderContentString` in `CheckError.aspx` sends unvalidated data to a web browser on line 12, which can result in the browser executing malicious code.

**Explanation:**
Cross-site scripting (XSS) vulnerabilities occur when:

1. Data enters a web application through an untrusted source. In the case of Reflected XSS, the untrusted source is typically a web request, while in the case of Persisted (also known as Stored) XSS it is typically a database or other back-end datastore.
Static Analysis & Reporting: (Example) Fortify 360

```
private string host_58 = "www.sandbox.paypal.com";
private const string SIGNATURE = "SIGNATURE";
private const string PWD = "PWD";
private const string ACTC = "ACTC";
//Replace <your API Username with your API Username
//Replace <your API Password with your API Password
//Replace <your Signature with your Signature
private string APIUsername = "";
private string APIPassword = ""
private string APISignature = ""
private string Subject = ""
private string BCCode = "PP-EChizard"

//HttpWebRequest Timeout specified in milliseconds
private static int Timeout = 5000;
private static readonly string[] SECURED_WOPS = new string[] { ACCT, CVV2, SIGNATURE, PWD }

public void SetCredentials(string Userid, string Pwd, string Signature)
{
    APIUsername = Userid;
    APIPassword = Pwd;
    APISignature = Signature;
}

public bool ShortcutExpressCheckout(string amt, ref string token, ref string retmsg)
{
    if (host_58) {
```

Abstract:
Empty passwords can compromise system security in a way that cannot be easily remedied.

Explanation:
It is never a good idea to assign an empty string to a password variable. If the empty password is used to successfully authenticate against another system, then the corresponding account's security is likely compromised because it accepts an empty password. If the empty password is merely a placeholder until a legitimate value can be assigned to the variable, then it can confuse anyone unfamiliar with the code and potentially cause problems on unexpected control flow paths.
Division of Labour

Tools

• Provide scale
• Aid the manual code review process through pinpointing potential areas of concerns

Human Reviewer

• To review and identify critical issues reported by the tools for action
• To examine and identify the remaining issues not reported by tools due to its various limitations
• To improve overall coding techniques, style and design
Agenda

• What is secure software?
• What is static analysis?
• Why static analysis?
• Where does static analysis fit in a SDLC?
• Static analysis tools – Pros and Cons
• Conclusion
Static Analysis in SDLC
Two of the Established Software Development processes

Traditional (and improved versions)

- Requirements
- Design
- Build + Test
- Deploy/Deliver

Agile (Scrum)

- Requirements
  - Product Backlog
  - Sprint Backlog

  Daily scrum
  2-4 weeks Sprint

  Potentially Shippable Product Increment
Static Analysis in Traditional SDLC

- Users' Requirements
  - Functional Requirements
  - Architectural Design
  - Detailed Design
  - Coding
  - Releases
  - Installing Operations & Maintenance
  - Code review
  - Static/dynamic code analysis
  - Code Unit test cases

- Model
  - Design review
  - Test case design
  - Coding rules & guidelines
  - Security rules, guidelines
  - Establish metrics
  - Threat Modelling
  - Asset classification
  - Detailed Design review/update
  - Threat model review/updates
  - Detail asset classification
  - Test case development

- Validation
  - Validate
  - Validate Verify
  - Verify

- Build
  - Integration Test
  - Unit Test
  - Alpha/Beta Test

- Security Training
  - Establish security requirements
  - Identify assets & adversaries
  - Peer review
  - Test Planning

- System Test
  - Functional Test
  - Non-functional Tests
  - Security Testing

- Releasing
  - Releases
  - Installs

- Operational Testing
  - Penetration Testing
  - Parallel Testing
  - Sociability Testing
  - etc.

- UAT (User Acceptance Test)

- Establish metrics
  - Threat Modelling
  - Asset classification
  - Test case development.
Static Analysis in Agile (Scrum)

- Security requirements
- Do I face any CIA issues?
- Do my attack surface change?

Requirements

Product Backlog → Sprint Backlog → 2-4 weeks Sprint

24 Hours daily scrum meetings

Refactor

Design time security ??
- Attack Surface reduction & analysis
- Thread Modeling

Potentially Shippable Product Increment
- Code Review
- Static/Dynamic Analysis
- Security Testing
- Penetration Testing
- Vulnerability Analysis

Do I face any CIA issues?
Do my attack surface change?
Static Analysis in Development & Test Environment

- **Static Code analysis**
  - Check in
  - Check out

- **Source Code Control System**
  - Software Configuration Management System
  - Symbol server

- **Issue/Defect management System**
  - Periodic Builds
  - Continuous Integration System
    - Unit Testing
    - Integration Testing
  - Test case execution/Management
  - Web-based Reporting
  - Static/Dynamic Code analysis
    - Generate code metrics,
    - Coding conformance,
    - Security issues,
    - Etc.

- Developers
- Sponsors, users and other stakeholders
- Project Manager
- QA/QC
Agenda

• What is secure software?
• What is static analysis?
• Why static analysis?
• Where does static analysis fit in a SDLC?
• Static analysis tools – Pros and Cons
• Conclusion
Limitations of Static Analysis Tools

- Limited bug detection
  - you need to continue with Dynamic Analysis in order to see a more complete picture
- False positives
- Performance and reliability of analysis tool in large scale project (esp for a server-based setup)
- Unable to provide guidelines to use appropriate patterns/libraries
- Limited understanding of complex programming language
- More reliable results with strong-typed languages
- Code interdependency and visibility issues inside third party libraries and frameworks
You can’t check everything!

This is what you write.

This is what you get.
Static Analysis (Manual) Aka - Code Review

There are limits to what a reviewer can do.

- *To err is human*…
- First and foremost – knowledgeable & experience reviewer
- Most of the time – what review is quality, conformance – not security
- Manual security code review is hard…..
  - Code dependency
  - No visibility inside library calls
  - New attack patterns knowledge
- Initial security /quality code review is best started with a static analysis tool
(Manual) Code Review

- Review selectively
  - Threat model & attack surface could provide identification of high risks modules
  - Review access control implementations with a due care
- Understand the language and error conditions of API calls
- Compiler and tools can help you way better than you think
  - Set compiler warning and error levels as a gate keepers
  - Let dev environment do the filtering of unsafe constructs
- More eyeballs are better than a single pair
Review using Static Analysis Tool

Eg:
Understand
(SCITools)
Review using Dynamic Analysis Tool

- Eg:
  - Code Map (Vstudio)
(Manual) Code Review

• Baseline all the source code using a configuration management system
  • File names – Source and Exe files
  • File owner – Owner should not be reviewing his/her own code
  • Priority – depending on the criticality of code and time to review
  • Reviewer – name of the person(s) who will review the code
  • Reviewed Status – yes/no/in-progress
  • Test owner – tester assign to the code
  • Sign off – yes/no
  • Comments
Code Reviews

- Removed 82% of all defects before testing
- 23% increase in productivity
- 38% reduction in bugs
- 14% increase in productivity
- 10x increase in quality
Gartner’s Magic Quadrant (Apr/2014):
Application Security Testing
Conclusion

- Incorporating systematic use of static analysis tools and reviews in software projects can save you a bundle.

- *But it is not a silver bullet.....It is just one important part of a secure software development life cycle.*

- Invest in people – especially in their education.

- Invest in tools and technology to scale up and save time.
(ISC)² Certifications in Secure Software

https://www.isc2.org/csslp/default.aspx

- For everyone involved in the SDLC with at least 4 years' experience.
- CBT Exam
- The comprehensive (ISC)² CSSLP CBK Education Program covers the following domains:
  - Secure Software Concepts
  - Secure Software Requirements
  - Secure Software Design
  - Secure Software Implementation/Coding
  - Secure Software Testing
  - Software Acceptance
  - Software Deployment, Operations, Maintenance and Disposal
  - Supply Chain & Software Acquisition
Thank You for your attention!

Q & A