GRAMMATECH

A FOUR-STEP GUIDE TO SECURITY ASSURANCE **FOR IoT DEVICES**

TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.GRAMMATECH.COM

INTRODUCTION

Powered by the forces of the cloud, connected endpoints, wireless technologies, and big data, the Internet of Things (IoT) and Machine-to-Machine (M2M) evolutions are forming a "perfect storm" for software engineering teams. Vendors are racing to claim a piece of the predicted 19 *trillion* dollar IoT market[1], made up of more than 50 billion IoT devices spanning nearly all markets – automotive, energy/utilities, home appliance, consumer electronics, medical, education, manufacturing, and more. This new landscape for embedded devices means increased connectivity and confidential data storage and transmission. Current manufacturers are still developing products using old and entrenched supply chain, engineering, and quality assurance processes that weren't designed for the complexities of today's highly-connected "smart" devices. Engineering teams are utilizing a progressively diverse set of suppliers and relying on 3rd party software to save while trying to satisfy the business and market thirst for IoT demands.

So how do device software processes evolve to better protect our next-generation IoT devices? First, it starts with a sound plan that includes next-generation software assurance and a "security-first" methodology. Teams need to rethink how they deliver software quickly – with security, safety, and quality in mind from design to deployment. IoT systems are complex and dynamic, as are the adversaries attempting to compromise the system (B. Zorn et. al.[2]) But "rethinking" should not be "restarting." To do this successfully, teams should leverage the best tools available that help them analyze the software they are developing, looking for problems that IoT presents – including both in-house source and 3rd-party binary code. The number of IoT device connections is growing exponentially – both an opportunity and a challenge – with the number of connections doubling every 4-5 years.

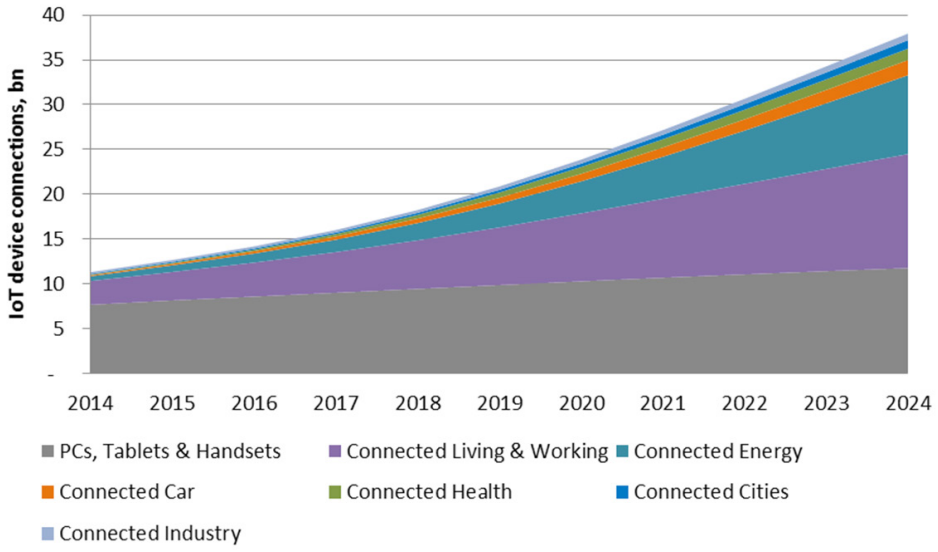


Fig. 1 *IoT device connection growth over time by device type. Source: http://enterprise-iot.org*

A FOUR-STEP QUALITY ASSURANCE GUIDE WILL CREATE BETTER SECURITY IN IoT DEVICES

Incorporating the following four major steps into an embedded software development process can improve security (and quality) for highly connected devices.

1 **Design with a "security-first" philosophy.** For connected devices in the IoT universe, security must be a prime consideration during all stages of development. The smart development team builds security requirements, development, and testing into the schedule and budget. Despite the potential unknowns and risks with device security, automated software tools are a significant boon to security assurance.

2 **A system-wide threat assessment and analysis.** For devices that are part of a larger IoT infrastructure, understanding the potential security issues at a system level are critical. A threat assessment explores and analyzes the attack vectors to a device -- an essential process for all IoT systems.

3 **Leverage automated tools as much as possible.** Security adds additional burdens to development teams and is often outside the realm of their expertise. Source code static analysis, for example, can find defects and security threats that traditional manual and automated techniques miss. Leveraging tools throughout the development process is key to achieving the productivity required to meet schedule and budget constraints.

4 **Use binary analysis to ensure the quality and security of third-party code.** Reliance on third-party software is growing in embedded development and using software of unknown quality and security is risky. Binary static analysis (and a combination of source and binary analysis) provides an automated technique for analyzing third-party software, ensuring it meets the system's quality and security standards.

Fig. 2 *A four-step security and quality assurance process for IoT devices.*

SECURITY-FIRST DESIGN

Security has not always been a primary concern for embedded devices -- connectivity among devices has long been assumed to be local and in the hands of trusted operators and devices. Stuxnet[3], however, quickly proved that even local access can't be trusted, as it infected PCs and laptops that then infected programmable logic controllers (PLCs) that were connected via

a local area network. Modern devices need to be connected to a network (and frequently the Internet), and these devices require more serious attention to security and applying security principles early in the development lifecycle.

SOFTWARE SECURITY IN THE SOFTWARE DEVELOPMENT LIFECYCLE

A security-first design approach means integrating security as a top priority in the software development lifecycle (SDLC). Developers and project managers can expect at least the following types of activities at these key stages[4]:

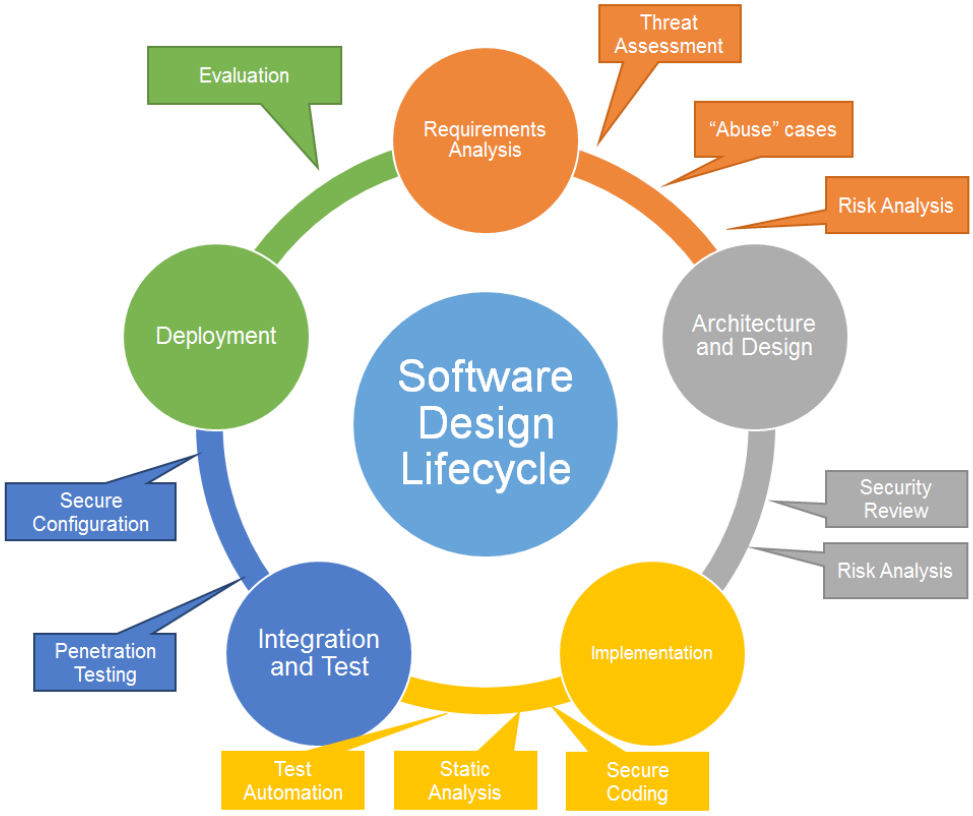
- **Requirements stage:** Once a system-wide threat assessment is available, the device threat surface can be understood. At the requirements stage, security-specific requirements can be introduced, along with known "abuse cases" (use cases that an attacker might follow) and a risk analysis. Security requirements, as listed below, are introduced and accounted for. This stage is critical because it is the point at which security becomes a known development project goal with the appropriate level of risk management, scheduling, and costing.

- **Design and architecture:** As candidate architectures become available, reviews must include security aspects, (where previously, they may not have). Assessing architecture in light of the known threat assessment and security requirements adds an additional dimension to this phase of development. At this stage, testing plans should be created that include security analyses that follow the perceived "abuse cases."

- **Code development:** At the coding stage, following security guidelines and coding standards are critical. The use of automation tools such as static analysis is key to ensure that vulnerabilities are not introduced into the product. Testing and test automation that includes a security analysis are important at this stage.

- **Integration and test:** As the system as a whole start to take form, subsystem and system testing will find vulnerabilities before integration and deployment to the market. Automated penetration testing tools can be very helpful at this stage to uncover vulnerabilities that may not have been accounted for in earlier stages of development. Packaging and configuration of the end product for deployment is key at the final stages of this phase. Ensuring that the out-of-the-box product is as secure as possible prevents many of the security issues we see today in connected devices.

- **Deployment and maintenance:** When a product enters the market and starts wide deployment, security vulnerabilities become exponentially costlier to fix. A product designed with a security-first approach is less likely to end up with a security breach, but companies must be prepared to deal with security on an ongoing basis. Designing the product with the ability to update firmware and software is critical to addressing new-found issues expeditiously. However, as a product goes through maintenance and revision, security is an on going concern and new vulnerabilities and new threats need to fed back in to the system in an iterative approach.



SECURITY REQUIREMENTS

Securing an embedded device requires many considerations. Key examples of security requirements that might go above and beyond existing functional requirements are as follows:

- **User authentication:** validating user access and enforced privileges for different classes of users.
- **Tamper resistance:** preventing physical and software changes to the device that allow circumvention of security functions.
- **Secure storage:** ensuring stored data is protected from online and offline access, including techniques such as encrypted file storage and Digital Rights Management (DRM).
- **Secure communications:** keeping data-transfer secure but also preventing unwanted access through connected channels (network, USB, etc.). Although network connectivity is top of mind, other channels are vulnerable to attack.
- **Reliability and availability:** maintaining safe operation of the device in the face of on going attacks.

THE ROLE OF SAST TOOLS IN A SECURITY-FIRST APPROACH

Static Application Security Testing (SAST) tools like Grammatech's CodeSonar provide critical support in the coding and integration phases of development. Ensuring continuous code quality, both in the development and maintenance phases, greatly reduces the costs and risks of security and quality issues in software. In particular, it provides some of the following benefits:

- Continuous source code quality and security assurance
- Tainted data detection and analysis
- Third-party code assessment
- Secure coding standard enforcement

THREAT ANALYSIS AND ASSESSMENT

A key ingredient to a security first design approach is an end-to-end threat assessment and analysis. The NSA has published a framework[5] for assessing and improving the security of industrial control systems, which is applicable to all embedded device markets included in IoT. Embedded devices are often part of a larger IoT infrastructure, and understanding the potential security issues at a system level is critical. A threat assessment includes taking stock of the various physical connections, potential losses/impacts, threats, and the difficulty of the attack. Importantly, addressing these threats needs to be prioritized based on likelihood and potential impact.

CONNECTIVITY ASSESSMENT

Cyber-attacks are perpetrated through the various connections a device has with the outside world. Although a network connection is a high-profile interface, designers should not ignore all possible connectivity. For example, the user interface, USB and other I/O ports, serial and parallel connectors are all possible vectors for attack. Since security incidents occur from both internal and external sources, physical connections to the device need to be considered. Equally important is the context of the device's connectivity. Is it physically available for people to access? If networked, is it behind a firewall and gateway? How many IP addresses, ports, and protocols are planned to be used?

LOSS ASSESSMENT

If an attacker gets unauthorized access to a device, what are the potential impacts of the access? For example, in an industrial control system, would unauthorized access cause damage or unintended functioning of the system? Would access compromise critical data? Would access disable the device or decrease its reliability? In some cases, erroneous input on an external connection can be enough to crash the system, causing an outage or damage. It's important to consider every possibility and the impacts of such unauthorized access.

LOSS METRIC

When considering each impact, categorize them into key areas and then assign an impact value to each of these metrics. For example, in most industrial control systems, loss of confidential data is not as severe as loss of integrity (which could include serious malfunctioning of the device). NIST[6] defines three loss metrics as follows:

- **Confidentiality** - unauthorized theft of sensitive information.
- **Integrity** - unauthorized alteration or manipulation of data. In embedded devices that control systems in the real world, this can include manipulation of command and control.
- **Availability** - loss of access or loss of use of the device.

In light of these loss metrics, a relevant impact value is assigned based on the type of device, its context, and its operational usage. Industrial control systems, for example, that control physical systems that have potential to injure (e.g. robot) or to impact people and property (e.g. energy grid, nuclear power plant, or water processing plant), would place a high impact on loss of integrity and availability.

THREAT ASSESSMENT

The impact of a cyber-attack on a device is a function of both its potential impact (loss metric, above) and the likelihood of the attack being possible, which, in turn, is a function of motivation and intent. If we look at Stuxnet as an example, the attack was sophisticated and relatively difficult to achieve; however, it was accompanied by a high level of motivation and intent.

A threat assessment is performed by looking at the motivations and intents of potential attackers, their possible avenues to attack your system, and the probability of them being successful in that attack.

Attack sources and motivations.
Threats can be insiders, activists, terrorist, criminals, state-sponsored actors, or competitors. Understanding the source of attacks and their motivation help you understand the goals of an attack and whether such a group could achieve the attack. For example, industrial control systems may not be interesting to criminals if there is no direct monetary gain from an attack.

Roles and privileges of authorized users.
Identifying users and their access rights is essential to enforcing a key security principle of least privilege. Limiting access of operational users to prevent dangerous operation or leakage of important data prevents insiders and attackers from gaining more than their privilege level allows. In so doing, gaining access to a user-level account may not be dangerous in a properly designed system.

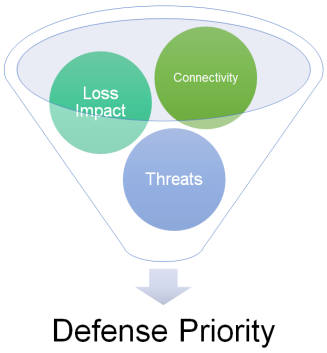
Identify potential electronic attack vectors.
Typically, network connections and other peripheral I/O provide intrusion access to attackers. In some cases, the attack vector maybe internal from local access to the user interface

or local area network. In other cases, access via a wide area network or even the Internet is possible. Understanding the scope of the devices connectivity is needed to understand the potential vectors.

Assess attack difficulty.
The loss assessment indicates which services and functions would have the most impact when attacked, the relative difficulty of these attacks must be evaluated based on the attackers and their intrusion vector.

Assign a threat metric.
It's not possible to foresee every attack nor is it efficient to attempt to protect against every possible attack. Attacks from outside the defendable network segment, for example, that have a large impact (loss metric) and a low attack difficulty would have the high threat metric. Scoring each combination of source and motivation, attack vector and attack difficulty is required.

PRIORITIZING DEFENSE



The relative priority of security defenses is derived from the loss and threat assessments – a combination of high loss and high threat yields the highest priority. This seems like common sense, however, without the extensive analysis of loss and threats, a development team can't objectively evaluate the priority of each defense. The results of this priority calculation leads to specific security requirements for the system, test and "abuse" cases for evaluation, as discussed previously.

It's important to note that this is an iterative process, not all threats are known at design time and attackers and vectors may change over time. Building threat assessment into a device software lifecycle is key to continuous security assurance over the life of a product.

AUTOMATED SOFTWARE DEVELOPMENT TOOLS FOR IMPROVING SECURITY

For IoT and M2M device security assurance, it's critical to introduce automated software development tools into the development lifecycle. Although software tools' roles in quality assurance is important, it becomes even more so when security becomes part of a new or existing product's requirements.

There are three broad categories of automated software development tools for embedded IoT products, that are important for improving quality and security:

- **Application Lifecycle Management (ALM):** Although not specific to security, these tools cover requirements analysis, design, coding, testing and integration, configuration management, and many other aspects of software development. However, with a security-first embedded development approach, these tools can help automate security engineering as well. For example, requirements analysis tools (in conjunction with vulnerability management tools) can ensure that security

requirements and known vulnerabilities are tracked throughout the lifecycle. Design automation tools can incorporate secure design patterns and then generate code that avoids known security flaws (e.g. avoiding buffer overflows or checking input data for errors). Configuration management tools can insist on code inspection or static analysis reports before checking in code. Test automation tools can be used to test for "abuse" cases against the system. In general, there is a role for ALM tools in the secure development just as there is for the entire project.

» **Dynamic Application Security Testing (DAST):** Dynamic testing tools all require program execution in order to generate useful results. Examples include unit testing tools, test coverage, memory analyzers, and penetration test tools. Test automation tools are important for reducing the testing load on the development team and, more importantly, detecting vulnerabilities that manual testing may miss.

» **Static Application Security Testing (SAST):** Static analysis tools work by analyzing source code, bytecode (e.g. compiled Java), and binary executable code. No code is executed in static analysis, but rather the analysis is done by reasoning about the potential behavior of the code. Static analysis is relatively efficient at analyzing a codebase compared to dynamic tools. Static analysis tools also analyze code paths that are untested by other methods and can trace execution and data paths through the code. Static analysis can be incorporated early during the development phase for analyzing existing, legacy, and third-party source and binaries before incorporating them into your product. As new source is added, incremental analysis can be used in conjunction with configuration management to ensure quality and security throughout.

Although adopting any class of tools helps productivity, security, and quality, using a combination of these is recommended. No single class of tools is the silver bullet[7]. The best approach is one that automates the use of a combination of tools from all categories, and that is based on a risk-based rationale for achieving high security within budget.

THE ROLE OF STATIC ANALYSIS TOOLS IN A SECURITY-FIRST APPROACH

Static analysis tools like GrammaTech's CodeSonar provide critical support in the coding and integration phases of development. Ensuring continuous code quality, both in the development and maintenance phases, greatly reduces the costs and risks of security and quality issues in software. In particular, it provides some of the following benefits:

- **Continuous source code quality and security assurance:** Static analysis is often applied initially to a large codebase as part of its initial integration; however, where it really shines is after an initial code quality and security baseline is established. As each new code block is written (file or function), it can be scanned by the static analysis tools, and developers can deal with the errors and warnings quickly and efficiently before checking code into the build system. Detecting errors and vulnerabilities (and maintaining secure coding standards, discussed below) in the source at the source (developers themselves) yields the biggest impact from the tools.

- **Tainted data detection and analysis:** Analysis of dataflows from sources (i.e. interfaces) to sinks (where data gets used in a program) is critical. Any input, whether from a user interface or network connection, if used unchecked, is a potential security vulnerability. Many attacks are mounted by feeding specially-crafted data into inputs, designed to subvert the behavior of the target system. Unless data is verified to be acceptable both in length and content, it can be used to trigger error conditions or worse. Code injection and data leakage are possible outcomes of these attacks, which can have serious consequences.

- **Third-party code assessment:** Most projects are not greenfield development and require the use of existing code within a company or from a third party. Performing testing and dynamic analysis on a large existing codebase is hugely time consuming and may exceed the limits on the budget and schedule. Static analysis is particularly suited to analyzing large code bases and providing meaningful errors and warnings that indicate security and quality issues. CodeSonar can analyze binary-only libraries and provide similar reports as source analysis, when source is not available. In addition, CodeSonar's binary analysis can work in a mixed source and binary mode to detect errors in the usage of external binary libraries from the source code.

- **Secure coding standard enforcement:** Static analysis tools analyze source syntax and can be used to enforce coding standards. Various code security guidelines are available, such as SEI CERT C[8] and Microsoft's Secure Coding Guidelines[9]. Coding standards are good practice because they prevent risky code from becoming future vulnerabilities. Integrating these checks into the build and configuration management system improves the quality and security of code in the product.

As part of a complete tools suite, static analysis provides key capabilities that other tools cannot. The payback for adopting static analysis is the early detection of errors and vulnerabilities that traditional testing tools may miss. This helps ensure a high level of quality and security on an on-going basis.

ASSESSING THIRD-PARTY CODE

According to VDC Research[10], 45% of embedded projects are outsourcing product development. The use of outsourced, open source, commercial software (COTS), legacy source and binary code are increasing each year (e.g. VDC claims embedded Linux[11] will be the embedded operating system of choice for 64.7% of all embedded shipments by 2017). Reliance on third-party software in embedded development and using software of unknown quality and security is risky and requires more scrutiny before forming part of an IoT and Machine to Machine (M2M) product. Static analysis of source and binaries is an essential part of the toolset needed to asses third party code.

RE-USE IS GOOD

Reinventing the wheel has never been productive. Buying versus building often favors re-use, whether through open source or COTS solutions. Projects are rarely greenfield developments, so legacy code and reusing other corporate assets are a reality. Some of the most common uses of third-party code include the following:

- **Communications** - Enabling an application to communicate via the Internet or wirelessly with other applications.
- **Databases** - Third-party software is used extensively to manage, optimize, monitor, and backup databases.
- **Standard libraries** - These typically include definitions for commonly used algorithms, data structures, and mechanisms for input and output. Developers have become so accustomed to some of them that they forget the libraries are not part of the language itself.

With re-use's reward, there is the risk of security vulnerabilities hiding in the code. Your end customer is going to hold your company responsible for the product regardless of where your source comes from. Evaluating the quality and security of existing source and binaries (libraries, object files, executables, and dynamic link libraries) is a particular strength of static analysis tools.

STATIC ANALYSIS FOR QUALITY AND SECURITY ASSURANCE OF THIRD-PARTY CODE

The code you receive can come in all forms; uncompiled source, partial source and binaries, or binaries only. Given this, tools are needed to handle each type of situation, as described below.

» **SOURCE ANALYSIS**
When full source is available, security assessment before integration into your product with static analysis tools such as GrammaTech CodeSonar is warranted. If the code is of "unknown pedigree (provenance)" or SOUP, as it is known, investigating quality and security errors is critical. In many cases, after initial assessment and repair, this acquired source can be integrated into the continuous build/analysis cycle.

» **BINARY ANALYSIS**
If precompiled libraries or executables are the only form your third party code is available in, there is still the possibility to assess the software. CodeSonar's binary analysis technology can evaluate binary code for quality and security vulnerabilities. Although the possibility of investigating and fixing the issues is often limited, it does provide a bellwether of the quality and security of the code. Customers of COTS products can go back to technical support of the vendor and ask for confirmation and analysis of the discovered vulnerabilities. The key here is that potential vulnerabilities are detected and accounted before code is used in a final product. More details on eliminating security vulnerabilities with binary analysis is available from GrammaTech[12].

» **SOURCE AND BINARY HYBRID ANALYSIS**
Binary analysis really shines when used in a hybrid fashion with source analysis. Source static analysis has much more information about the intent and design of the software than binary analysis. However, whenever an external library is called, including standard C/C++ libraries, static analysis can't tell if the use of the function is correct or not (assumptions are made, of course, for well known functions like `strcpy()`). By combining source and binary analysis, a more complete analysis is possible. For example, if an external function takes a pointer to a buffer and a buffer overflow is possible with misused parameters, hybrid static analysis can detect this problem.

SUPPLY CHAIN RISK MANAGEMENT FOR IoT

The growth rate in product development and the potential proliferation of millions of insecure devices is a serious threat to the success of IoT in the marketplace. Companies serious about security need to manage the security risk in their supply chain, including software (COTS and FOSS). Static analysis provides an easy-to-adopt and efficient approach to improving the quality and security of the third-party software.

The success of IoT will depend somewhat on the smart build versus buy decisions on behalf of the device manufacturers. However, bringing in outside source and binary code has its risks and proper assessment of this software is critical. Static and binary analysis used individually and together provides a practical way to assess third-party code. To benefit from software reuse, it is important to keep all code at the same high quality and security standards.

SUMMARY

M2M and IoT device manufacturers incorporating a security-first design philosophy, with formal threat assessments and leveraging automated tools, will produce devices better secured against the accelerating threats on the Internet. Modifying an existing successful software development process that includes security at the early stages of product development is key. Smart use of automated tools to develop new code and analyze existing and third party code allows development teams to meet strict budget and schedule constraints. Static analysis of both source and binaries plays a key role in a security-first development toolset.

IoT is here, and the responsibility is ours to ensure our software is ready for it.

REFERENCES:

1. Cisco CEO Pegs Internet of Things as $19 Trillion Market.
2. Systems Computing Challenges in the Internet of Things.
3. The Real Story of Stuxnet, David Kushner, IEEE Spectrum, Feb., 2013.
4. Security as a New Dimension in Embedded System Design.
5. A Framework for Assessing and Improving the Security Posture of Industrial Control Systems (ICS).
6. NIST Guide to Industrial Control Systems (ICS) Security.
7. No Silver Bullet – Essence and Accident in Software Engineering, Fred Brooks, 1986.
8. SEI CERT C Coding Standard.
9. Outsource Code Development Driving Automated Test Tool Market, VDC Research, IoT & Embedded Blog, October 22, 2013.
10. The Global Market for IoT and Embedded Operating Systems. VDC Research, 2015.
11. Guidelines for Writing Secure Code, Microsoft, 2010.
12. Eliminating Vulnerabilities in Third-Party Code, GrammaTech, 2014.

GrammaTech, Inc. is a leading developer of software-assurance tools and advanced cybersecurity solutions. GrammaTech helps organizations develop and release high quality software, free of harmful defects that cause system failures, enable data breaches, and increase corporate liabilities in today's connected world. GrammaTech's CodeSonar is used by embedded developers worldwide.