Software Security Assurance SOUP to NUTS

Dr. C. Warren Axelrod, Delta Risk LLC

Abstract. The ability to assess risks of and from specific software supply chains depends in large part on the amount, accuracy and availability of essential information. Only when such information is at hand can we hope to assure ourselves of the quality and security of installed software. In this paper we use an expanded version of the Cynefin Framework to come up with preferred approaches to categorizing software supply chains not only based on the potential knowledge levels of those responsible for evaluating, approving and operating systems, but also according to what can be known about particular supply chains. We suggest how each category of supply chain might be evaluated and fixed in the face of adverse incidents.

Introduction

For this context the most appropriate definition of "supply chain risk" is:

"… the risk that an adversary may sabotage, maliciously introduce unwanted function, or otherwise subvert the design, integrity, manufacturing, production, distribution, installation, operation, or maintenance of a covered system so as to surveil, deny, disrupt, or otherwise degrade the function, use, or operation of such system."[1]

In order to manage software supply-chain risk, accurate and extensive data must be collected, analyzed and responded to. All too often, however, crucial data are not readily at hand or they are difficult and/or expensive to collect, if indeed they can be gathered at all.

According to a 2004 report on "Defense Acquisitions,"[1] the GAO found that the U.S. DoD acquisition and software security policies were inadequate particularly in addressing risks relating to foreign suppliers developing weapon system software. Because of increasing difficulty and costs of testing computer code, the GAO suggested that, rather than testing code, those responsible for approving systems learn more about who developed the software and where they were located in order to arrive at a more informed vendor selection decision, which could mitigate risks. While such an approach is better than nothing, it does not come close to the level of software assurance obtained from independent in-depth testing of computer code. Furthermore, software makers usually incorporate software components from other sources, including open sources, which may not be known to vendors, contractors, or their customers.[2]

In this article, we investigate why so much necessary information is not forthcoming and propose approaches for obtaining elusive and costly software supply-chain data. Such information can provide analysts with the ability to anticipate, detect and react to adverse issues before, during and after they occur, rather than well after the fact, which is unfortunately more usually the case. Investment in the collection and analysis of software supply-chain metrics offers the potential of significant returns on investment in an ever more complex environment.

In addition, a worldwide data-sharing infrastructure is needed in order to allow entities comprising global supply chains to inform one another of events that will likely have a significant impact on the quality and availability of supplied software and equipment components. In order to understand what data need to be collected and how they should be used by decision-makers to manage the vagaries of software supply, we take the Cynefin Framework and extend it to cover additional software supply-chain characteristics. Based on this approach, we are able to suggest appropriate data-gathering and decision-making methods that meet each of a large variety of situations.

DoD and National Security Context

In a 2012 report on "IT Supply Chains," [3] the GAO affirmed that, among the four U.S. national security-related departments, the DoD had made greater progress by defining supply chain protection measures and implementing procedures for IT supply-chain assurance than had the departments of Energy, Homeland Security and Justice. Nevertheless there is still much work to be done by the latter three agencies with national-security responsibilities, as recommended by the GAO report.

This does not mean, however, that the DoD is free and clear when ii comes to IT supply-chain risk management. Despite all the progress in methods, procedures and tools that has been made over the last decade, there are still many areas that remain unknown, and may not even be knowable, to DoD program managers, particularly since extensive code reviews and software assurance testing have not been required. This implies that full assurance of IT supply chains remains a goal rather than a reality. Little has appeared in the literature on the ability of analysts to know each and every component of IT supply chains so that many of the structures of, and participants in, supply chains remain obscure or unknown, particularly with respect to commonalities [4]. Consequently, many vulnerabilities are not known either. As stated in [5]:

"[The DoD needs] to better "see" into some legs of the supply chain, especially where critical components are involved."

While a report by Adams [6] is oriented towards the manufacture of physical products rather than software in regard to supply chains of the U.S. defense industry, its conclusions also apply to IT products, software, and services. The report recommends the following:

1. Increase long-term federal investment in high-technology Industries
2. Apply and enforce existing laws and regulations
3. Develop domestic sources for key … resources
4. Develop plans to strengthen the defense industrial base
5. Build consensus … on the best ways to strengthen the defense industrial base
6. Increase cooperation among federal agencies and between government and industry
7. Strengthen collaboration among government, industry and academic research institutions
8. Ensure collaboration on economic and fiscal policies for long-term budgeting
9. Modernize and secure defense supply chains [emphasis added]
10. Identify potential defense supply-chain chokepoints and plan to prevent disruptions

It should be noted that the report [6] does not generally focus on the need to collect the knowledge necessary for making appropriate supply-chain decisions, although the exhortation to "identify chokepoints" implies some degree of information gathering. Facilitating the acquisition, analysis and understanding of data about software supply chains is a dominant objective of this article. That is to say, we want to bring to light how decision-makers should go about determining what is known, what is not known, what it will take to acquire the necessary knowledge, what is unknowable, and what they need to do under various circumstances.

Similarly, neither the recently published NIST Special Publication [7], which applies across all Federal information systems and organizations, nor the CNSS report [8], which addresses national security systems, specifically examine the "ability to know" supply-chain information. They proceed with the understanding that required information is readily available, which is far from the case in many circumstances. Nevertheless, both of these publications set forth invaluable guidance and the CNSS report [8] provides a very useful list of references with which DoD managers responsible for supply chains should become familiar.

The Provenance of Software

If you don't know where critical software comes from, then you may well be in the SOUP, literally, where SOUP means "Software of Unknown Provenance (or Pedigree)" Such software products may not be trustworthy because their origins are questionable or unknown. At the other extreme, if you think that you know everything about a particular piece of software, e.g., who designed it, who wrote it, and who tested it, the results of the tests, and so on, then you might be willing to rely on NUTS[3] or "Not Unreasonable Tracking Systems," in order to verify that the software development lifecycles (SDLCs) involved follow predetermined routes and are subject to appropriate levels of oversight.

Of course, there are many other situations between no knowledge and complete knowledge, such as knowing something about the backgrounds of some of the developers and their works, but not enough to give one much confidence that there aren't any little malware devils that might be lurking within the overall system, often for years, until they are revealed through some incident or other. Even when software is "open source," meaning that its source code is available to anyone wishing to look through the programs and modify them (under certain predetermined conditions), there are no guarantees that errors or deficiencies have not been introduced or that there is sufficient funding to provide suitable levels of technical and operational support. The exploitation of Heartbleed and Shellshock malware demonstrated this.[4]

Furthermore, there are times when everyone else appears to have known about some threat or vulnerability, but you just didn't happen to have been aware of them (oblivious), in which case there will some answering to do in order to satisfy management ... or not, as the case may be.

Goals of Decision Makers

In order to establish the best possible situation, given the proliferation of buggy software and the ability of evildoers to take advantage of these deficiencies, one's goals should be to:

- determine what is known about a piece of software's provenance and what is not
- understand which risks are known to the community and which are not
- find out more about unfamiliar risks so that they might be mitigated
- take steps to mitigate known risks or have good reasons for not having done so
- come up with approaches for dealing with unknown or unexpected risks
- establish a professional and industry/sector network to stay informed about risks relating to supply chains of software that you plan to acquire and install
- maintain current knowledge about software supply-chain research, industry/sector and professional publications, conferences, podcasts, webinars, etc.
- understand that there are certain software products that operate covert systems about which you may never know but which can affect you in some way or another, purposely or inadvertently

We will gain a better understanding of how to achieve these goals by expanding an established decision framework to incorporate additional contexts found in software supply chains.

The Known/Unknown (K/U) Model

Since lack of knowledge is a major contributor to inadequate and inappropriate responses to supply-chain malfunctions and failures and the ability to recover quickly, it is important to fill in where there are clearly deficiencies. The first step is to understand what makes up the universe of knowledge and then determine which areas need to be augmented with a higher level of understanding. In Table 1, we show how knowledge about software supply chains might be categorized depending upon how knowledgeable cybersecurity professionals might be concerning particular software supply-chain deficiencies or weaknesses.

The underlying concept here is that either you know or don't know in advance about specific threats or vulnerabilities with respect to particular software products' supply chains. If you did know, the question then arises as to whether you responded appropriately. If you didn't know, then how are you going to ensure that you will get advance notification if and when a similar situation is occurs in the future? If you didn't know but should have known, then your suitability to the task is in question. If you could not have known, you need to examine whether you have appropriate monitoring and incident-response mechanisms in place to react correctly.

These concepts of whether one is aware or unaware of various situations have been incorporated into a framework, called the Cynefin Knowledge Framework ("Cynefin"), which is designed to assist leaders in their decision making. The model is described in [9]. As mentioned above, we will expand this framework to facilitate decision-making with respect to software supply chains.

The Cynefin Knowledge Framework

Cynefin (translated from the Welsh as "habitat" or "place") is roughly analogous to the above K/U model. Cynefin suggests how decision-makers should respond to events that fall within

Analysts' Knowledge	Information Available	
	Knowns	**Unknowns**
Known	**Obvious** – I knew all about this in advance but didn't act on it quickly enough	**Obscure** – I knew that I didn't know anything about this, but couldn't get the data for economic or other reasons
Unknown	**Oblivious** – I was not aware of this even though my peers were	**Unfathomable** – I didn't have a clue that this existed, nor did my peers

Table 1. K/U model categories of knowledge by information available

various contexts. In this article, we extend the framework to cover situations not specifically addressed in Cynefin.

In a video,[5] Snowden differentiates between categorization models (such as the 2 x 2 matrix K/U model above) and "sense-making" frameworks, such as Cynefin. With categorization models, the framework precedes the data; but for sense-making frameworks, "the framework itself emerges from the data ..." Figure 1 illustrates Cynefin, which has evolved over time.[6] For example, "simple" contexts in have been replaced with "obvious" contexts, and the fifth category "disorder" seems to have been dropped. Also, there are subtleties that do not show up in the diagram, but are described in the video, such as catastrophic consequences of a transition from "obvious" to "chaotic" contexts. "Disorder" contexts cover otherwise uncategorized items.

Cynefin divides the contexts between "ordered systems," which are highly constrained and predictable, whether obvious or complicated contexts, and "unordered systems," which have fewer constraints and, for chaotic contexts, exhibit unpredictable random behavior.

Categories within the known/unknown (K/U) model are quite similar to Cynefin contexts, except for two instances. One instance is the chaotic system, the context of which is "unknowable," and the other instance is K/U Model's category of "unknown knowns," which is not represented explicitly in Cynefin. Table 2 shows similarities and differences between the two models.

In Table 2, we have added three contexts, namely, "oblivious," "obscure," and "stealth," which are shown in the shaded entries. "Oblivious" contexts, which are part of the K/U model, are those where decision-makers are not aware of certain information generally known to many practitioners. Note that "oblivious" is a characteristic of decision-makers rather than of systems. "Obscure" contexts, which belong to neither Cynefin nor K/U, are those where surreptitious methods are needed to find out about system vulnerabilities.[7] "Stealth" contexts are for systems which are meant to be kept secret.[8] The expanded Cynefin framework is illustrated in Figure 2.

According to the definitions of Cynefin realms, "knowable "and "known unknowns" realms are equivalent—for "knowable," decision-makers are aware that certain items, which are not known, may become known through analysis. For "known unknowns," items are known to some but not to others.

If items are "unknowable," then nobody knows about them and you are generally "off the hook" if they occur. However, if you

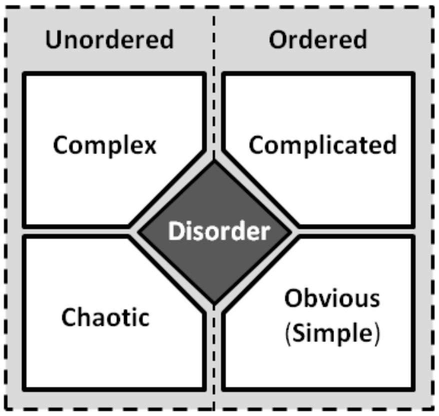

Figure 1: The Cynefin Knowledge Framework

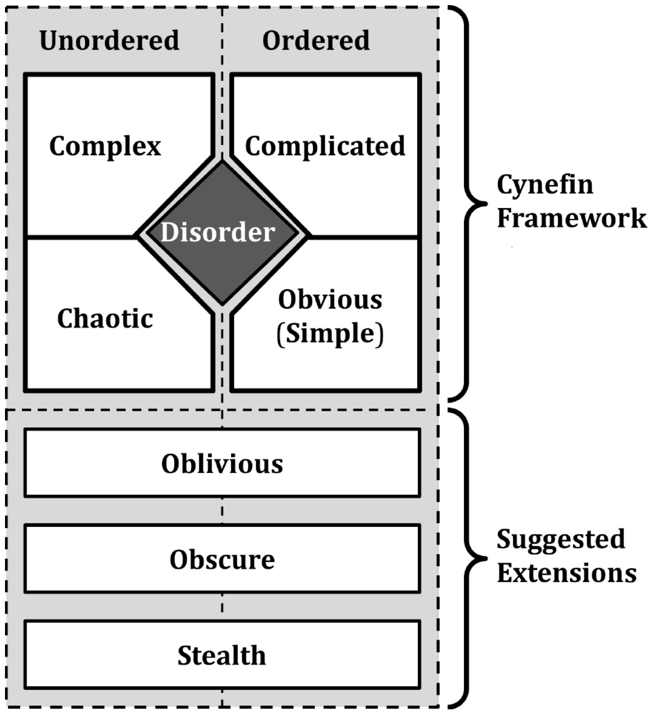

Figure 2. Suggested Expanded Framework

Contexts	Practices	K/U Model	Cynefin Realms	Differentiating Activities
Obvious	Best	Known knowns	Known knowns	Categorize
Complicated	Good	Known unknowns	Knowables – Known unknowns	Analyze
Complex	Emergent	Unknown unknowns	Unknown unknowns	Probe
Chaotic	Novel	--	Unknowable unknowns	Act
Oblivious	Ignorant	Unknown knowns	--	Investigate
Obscure	Clandestine	Unknown knowables	--	Deal
Stealth	Secret	Unknowable unknowables	--	Respond

Table 2: Knowledge for system contexts of an extended framework.

don't know about something that you should (this is not addressed specifically in Cynefin), then you might be accused of not maintaining currency in the field. This latter situation is most dangerous with respect to software supply chains, since decision-makers might be considered ignorant (or worse, negligent) in the event that something goes wrong.[9] This is why information sharing is crucial for successful management of software supply chains.

We now extend Cynefin to include the K/U model so as to determine the decisions that need to be made and the amount of effort to be expended on assessing and mitigating risks. In Table 3, we show Cynefin (unshaded areas) with extensions derived from the K/U model (shaded areas).

Realistically, there are those with software supply-chain responsibilities who are somewhat unaware of what is going on in the outside world as it pertains to their supply chains. Published reports about how organizations scramble in response to malware and hacking incidents and other forms of supply-chain disruption support the contention of ignorance, even when information about vulnerabilities and weaknesses are already in the public domain.[10]

Most academic treatments of this topic do not address dealing with criminal elements to obtain obscure information about malware and back doors that may have been inserted into software products during their supply-chain lifecycles. However, it is common knowledge that there are large and lucrative markets for the sale of exploits and vulnerability information.[11] Some might consider such information to be "unknowable," if they refuse to deal with dubious, clandestine or criminal elements. Also, the news about secret software systems is usually mere happenstance as might occur through some error or by the leakage of classified information by insiders.

Software Supply-Chain Risks

It can be difficult to come up with meaningful risk assessments for each of the seven contexts in the extended Cynefin. In the first place, analysts and/or decision-makers are often not aware of supply-chain weaknesses. Whether such defects will have serious personal and organizational consequences depends largely on efforts made to find out about vulnerabilities preemptively. As mentioned, an important consideration is whether one's peer group is already aware of such vulnerabilities. It is much more damaging to one's career if you are one of only a very few who lack knowledge than in a situation where everyone is just as ignorant.

The reverse may not be true, however. If you anticipate an issue that others don't or won't recognize as important, whether it is to your advantage or not when an incident occurs depends on whether you acted on the knowledge. For example, if you expect the electrical power grid to be unreliable in a particular country or region and you installed a generator when others in the area did not, you become a hero when a power outage occurs. However, if you just mentioned the power problem but did not install a backup generator, you might be considerably worse off than if you hadn't mentioned the problem in the first place, since you might be accused of not being aggressive enough in making your case.

Software Supply-Chain Lifecycles

As described in [12], software supply chains differ significantly from those of physical products. Software's unique characteristics include the following:

- Software can be copied without affecting the original and sold on the black market
- Software can be distributed in electronic form without transporting physical media
- Malware and back doors can be inserted into authentic software without leaving any trace

Because of these characteristics, the software supply-chain lifecycle is also somewhat unique. Table 4 lists specific attributes of software supply chains for each phase.

Information and Communications Technology (ICT) Supply-Chain Risks

A particularly extensive report [13], developed by the DoD, provides a list of threats that can, and do, impact software and software supply chains, including: Sabotage, Tampering, Counterfeiting, Piracy, Theft, Destruction, Disruption, Exfiltration—theft, Exfiltration—disruption, Infiltration, Subversion, Diversion, Export Control Violations, Corruption, Social Engineering, Insider Threat, Pseudo-insider Threat, and Foreign Ownership.

Ordered/ Unordered	Knowledge	Knowns	Unknowns
Ordered	Known (Knowable)	• Contexts: **Obvious** (Simple) • Realm: **Known knowns** • Domain: **Best practice** • **Standard process** invoked with review cycle & clear measures	• Contexts: **Complicated** • Realm: **Known unknowns** • Domain: **Good practice** • **Analytical techniques** used to determine facts
Ordered or Unordered	Unknown	• Contexts: **Oblivious** • Realm: **Unknown knowns** • Domain: **Ignorant** • **Investigations** of vendors, contractors and industry and professional groups to find out what is generally known	• Contexts: **Complex** • Realm: **Unknown unknowns** • Domain: **Emergent** • **Diverse interventions** needed to create options
Ordered or Unordered	Unknowable	• Contexts: **Obscure** • Realm: **Unknowable knowns** • Domain **Clandestine** • **Clandestine** dealings to try to get information	• Contexts: **Chaotic** • Realm: **Unknowable unknowns** • Domain: **Novel** • **Single or multiple actions** required to stabilize situation
Ordered or Unordered	Unknowable		• Contexts: **Stealth** • Realm: **Unknowable unknowables** • Domain: **Secret** • Able to **respond** only when secret is unintentionally disclosed

Table 3: Extensiaons to the Cynefin framework compared to the K/U model

Phase	Software Supply-Chain Lifecycle Attributes
Requirements Design Building (Development)	Requirements (specifications), design and development can be done virtually anywhere that has suitably educated staff and reliable, low cost telecommunications
Distribution Warehousing	Although some software is still distributed on physical media, it is common to distribute software electronically and increasingly software is available in the Cloud so no distribution as such is necessary.
Deployment	Software is deployed via various wholesale and retail outlets although it is often downloaded from vendor and or distributor websites, including open-source.
Operation Maintenance and Support	In theory, software can be run indefinitely although there are reasons for it becoming obsolete, such as cessation of vendor support, replacement of operating systems and platforms, changes in hardware, etc.
Disposal	Software can generally be deleted or replaced without having to destroy media, although having users properly eliminate all traces of the software, including backup copies, is unreliable.

Table 4: Software characteristics for phases of the supply-chain lifecycle

While many of these threats apply to software products generally, including those built in-house, they all can occur in both national and global software supply chains. Table 5 suggests some risk mitigation approaches for each context of our extended model:

In general, risk mitigation comprises obtaining as much advance warning as possible from a broad population of sources and responding in ways that improve, rather than exacerbate the situation. It is strongly advised to have a complete set of contingency plans in place so that they can be drawn upon as circumstances require.

Software Assurance Factors

Much of software supply-chain risk management involves information sharing and decision making based upon contexts in order to mitigate the many risks that affect software supply chains. However, many incidents that occur can be avoided by proactively making sure that the software goes through a rigorous software assurance process, which might include various forms of certification.

Knowledge	Knowns	Unknowns
Known **(Knowable)**	**Obvious**—Activate preplanned response procedures which should have been developed as part of the software acquisition process	**Complicated**— • Try to avoid using particular software that is known to have issues (although specific issues may not be know) • If use is unavoidable, monitor status of software and apply patches immediately
Unknown	**Oblivious**—Activate incident-response procedures and quickly link up with professional and industry "grapevines" so as to be forewarned of future threats	**Complex**—Activate incident-response process and try to determine whether similar incidents might be anticipated and avoided in the future
Unknowable	**Clandestine**—Determine who might know about unknowable vulnerabilities and make deals with those with relevant information	**Chaotic**—React to unexpected chaos with creative responses in order to stabilize the situation before being able to take corrective or restorative actions
Unknowable		**Secret**—First, understand the relevance of the revelation of a secret system to your organization and then respond as appropriate, if at all

Table 5: Risk mitigation approaches for various contexts

In order to incorporate software assurance standards into supply chains, it is first necessary to determine what those standards should be and how they should be used and managed. As described in [10], this could be accomplished addressing a number of technical, economic and governance issues including:

- Development of software assurance technical standards
- Management of software-assurance and certification standards
- Evaluation of tools and techniques for assuring software
- Determination of update frequency for tools and techniques
- Focus on the most pressing threats to software and supply chains
- Establishment of models of the economics of software-assurance solutions, and testing and certifying software

Once such standards have been established, we come to the far greater task of enforcing them on third parties both domestically and internationally. As can be imagined, this would require a major political effort far beyond anything that has been attempted so far in this arena. Nevertheless, some significant part of this goal needs to be implemented if trust in software is to be achieved at even a rudimentary level. The only real possibility to make progress here is to use economic means of encouragement as can be brought about with a carrot, by (for example) requiring government agencies only to buy software that meets agreed-upon international standards, or with a stick by invoking legal measures that places liability on software manufacturers, as suggested in [10].

Conclusions

Before one can reasonably address the quality of software emanating from supply chains, it is necessary to understand the various contexts within which knowledge of software products' provenance can exist. It is suggested that the known/known model combined with the Cynefin framework can provide a basis for decision-making possibilities.

Risks relating to software supply chains come from both the software itself and the supply-chain process that served to create the software. We looked at many of these risks and suggested how they might be addressed.

Finally we looked at software assurance requirements that, if addressed appropriately into software supply chains, would serve to ensure that the software products themselves have the desired security and integrity.

In general, we are far behind where we should be in the fight against vulnerable and dangerous software and the practices that govern them. We therefore need to take a holistic view of the factors that affect software supply chains and the software products that emanate from them, and we must mitigate the risks with due deference to the need for efficient and effective means of manufacturing the software that is at the base of practically all new systems of any importance.

ABOUT THE AUTHOR

Dr. C. Warren Axelrod is a senior consultant with Delta Risk LLC specializing in cyber security, risk management and business resiliency. Previously, he was the Business Information Security Officer and Chief Privacy Officer for US Trust.

He was a founding member of the FS/ISAC (Financial Services Information Sharing and Analysis Center) and represented national financial services cyber security interests during the Y2K date rollover. He testified before Congress in 2001 on cyber security.

His recent books include Engineering Safe and Secure Software Systems (Artech House, 2012) and Outsourcing Information Security (Artech House, 2004).

He holds a Ph.D. in managerial economics from Cornell University, and a B.Sc. in electrical engineering and an M.A. in economics and statistics from Glasgow University. He is certified as a CISSP and CISM.

Phone: 917-670-1720

Email: waxelrod@delta-risk.net

ADDITIONAL READING

1. Axelrod, C. Warren. "Malware, 'Weakware,' and the Security of Software Supply Chains," CrossTalk (March/April 2014): 20-24.
2. Axelrod, C. Warren. "Addressing Supply-Chain Complexity Using Closed-Loop Simulation-Based Exercises," Proc, of the Complex Systems 2015 Conference, New Forest, UK, May 2015.
3. Davidson, Don, "Managing Global Supply Chain Risk: Security & Resiliency (of the Chain) and Integrity (of Product)," <http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Past%20Meetings/Program%20Protection%20Workshop%20May%201-2,%202012/Don_Davidson_ManagingGlobalSupplyChainRisk.pdf>

REFERENCES

1. U.S. General Accounting Office (GAO), Defense Acquisitions: Knowledge of Software Suppliers Needed to Manage Risks, GAO-04-678, May 2004.
2. Vaughan, Steven J. "It's an Open Source World: 78 Percent of Companies Run Open-Source Software," April 6, 2015. Available at <http://www.zdnet.com/article/its-an-open-source-world-78-percent-of-companies-run-open-source-software/>
3. U.S. General Accountability Office (GAO), IT Supply Chain: National Security-Related, Agencies Need to Better Address Risks, GAO-12-361, March 2012.
4. Axelrod, C. Warren. "Risks of Unrecognized Commonalities in the Information Technology Supply Chain," Proceedings of the 2010 IEEE International Conference – Technologies for Homeland Security, Waltham, MA, November 2010.
5. Davidson, Don, and Stephanie Shankles. "We Cannot Blindly Reap the Benefits of a Globalized ICT Supply Chain," CrossTalk, (March/April 2013): 4-7.
6. Adams, John, ReMaking American Security: Supply Chain Vulnerabilities & National Security Risks Across the U.S. Defense Industrial Base, Alliance for American Manufacturing, May 2013.
7. Boyens, Jon et al. Supply Chain Risk Management Practices for Federal Information Systems and Organizations, NIST Special Publication 800-161, U.S. Department of Commerce, April 2015.
8. Committee on National Security Systems (CNSS), Supply Chain Risk Management (SCRM), CNSSD No. 505, March 2012.
9. Snowden, David J., and Mary E. Boone. "A Leader's Framework for Decision Making," Harvard Business Review (November 2007).
10. Denning, Dorothy E. "Privacy and Security: Toward More Secure Software," Communications of the ACM, April 2015, pages 24-26.
11. Verizon Enterprise Solutions, 2015 Data Breach Investigations Report, 2015. Available via link at <http://www.verizonenterprise.com/DBIR/2015/>
12. Axelrod, C. Warren. "Mitigating Software Supply Chain Risk," ISACA JOnline, August, 2013. Available at <http://www.isaca.org/Journal/archives/2013/Volume-4/Pages/JOnline-Mitigating-Software-Supply-Chain-Risk.aspx>
13. Goertzel, Karen M., et al. State of the Art Report on Supply Chain Risk Management for the Off-the-Shelf (OTS) Information and Communications Technology (ICT) Supply Chain, Department of Defense, Information Assurance Technology Analysis Center (IATAC), USA, 2010.
14. Axelrod, C. Warren. "Reducing Software Assurance Risks for Security-Critical and Safely-Critical Systems," Proc of the 2014 IEEE LISAT (Long Island Systems, Applications and Technology) Conference, Farmingdale, NY, May 2014.

NOTES

1. This definition is from Section 806 of the Ike Skelton National Defense Authorization Act for Fiscal Year 2011 which is available at <http://www.gpo.gov/fdsys/pkg/BILLS-111hr6523enr/pdf/BILLS-111hr6523enr.pdf>
2. It is claimed in Vaughan [1] that 78 percent of companies use open-source components. While the source code of open-source software is readily available for viewing and testing, there remain many unknown issues with respect to the provenance, quality and support of particular widely-used software as recent incidents have shown.
3. The acronym NUTS already exists with several connotations, one of which is used by the military and has the meaning "Nuclear Utilization Target Selection." The author's designation of NUTS as "Not Unreasonable Tracking Systems" does not have any such provenance.
4. Two of the most damaging cyber attacks in recent times occurred against open-source software, namely OpenSSL (Heartbleed) and Bash (Shellshock). OpenSSL runs on a substantial population of web servers and Bash is integrated into many popular operating systems.
5. David Snowden's explanation of his framework is at <https://www.youtube.com/watch?v=N7oz366X0-8>
6. See Craig Brougham's July 9, 2014 posting "Cynefin 101–An Introduction," available at <http://www.infoq.com/articles/cynefin-introduction>
7. In [10], Denning mentions having the U.S. government pay "bug bounties" to obtain information about software vulnerabilities that they would then make available to the public, but she opposes such a program in favor of developing a suitable liability regime for software developers and users.
8. Stuxnet is an example of covert malware which was supposed to be kept secret but was accidentally released into the general Internet and was then analyzed and publicized, thereby losing much of its value by alerting potential victims as to its form and function.
9. There are many situations in which culpability depends upon who knew what and upon what one might reasonably be expected to have known at the time of an incident. The knowledge gap is attributable to decision-makers in such cases and not to contexts. This is perhaps why Cynefin does not include the "unknown known" category.
10. Verizon's latest data breach report [11] indicates that "99.9% of the exploited vulnerabilities were compromised more than a year after the CVE (Common Vulnerabilities and Exposure) was published."
11. See [10]