Software Architecture Risk Analysis (SARA): A Methodology to Assess Security Risks in Software Architectures, and an Application

Frédéric Painchaud
Defence Research and Development Canada – Valcartier
2459 de la Bravoure Road
Québec, QC G3J 1X5
CANADA

Frederic.Painchaud@drdc-rddc.gc.ca

ABSTRACT

Security has been an area of concern in information systems for more than four decades. However, interest has enormously grown worldwide in the last decade with the proliferation of attacks and malware coming to public attention, such as Titan Rain, the cyber attacks against Estonia, GhostNet, Operation Aurora, Stuxnet, Duqu, Flame and Red October, to name only a few. After years of work on cyber security, the community has learned that:

1) *no non-trivial system is ever totally secure; and*

2) *a plethora of activities, and not just a single silver-bullet activity, needs to be performed to bring the security of a system to a substantial level.*

Many of these activities must be done during system engineering, while the system is being developed, making the system more secure right from the beginning. A good example of such activities is testing. Other activities, like monitoring, must be done after the system has been deployed, during its operation and maintenance. Finally, some activities can actually be done both during and after development. One of those is the security risk assessment of the system's architecture. It can be done in the design phase to develop an architecture that will likely have fewer exploitable vulnerabilities, and it can also be done while maintaining the system to evaluate the impact of changes on security risks.

This paper presents a methodology named Software Architecture Risk Analysis (SARA). This methodology is based on established risk analysis processes so it is coherent with actual practices. It also structures the use of the analyst's knowledge in security, focuses on quick, scoped, repetitive and complementary assessments and finally, emphasizes participation of system architects, designers and users. This paper covers each step of the SARA methodology and presents one selected SARA application.

1.0 INTRODUCTION

The last fifteen years have been very intense in the computer security community. We have been fighting against stack-based buffer overflows, format string vulnerabilities, TCP and ASN.1 vulnerabilities, SQL injections, XSS, heap overflows, use-after-free vulnerabilities, memory information leaks and so on and so forth. We have realized that these vulnerabilities have been present in our software as soon as we started using the technologies that give rise to them, whether it is C programming, web applications or even modern operating systems. After a few "We solved security!" announcements that turned out to be untrue, we realized that:

1) no non-trivial system will ever be totally secure; and
2) a plethora of activities, and not just a single silver-bullet activity, needs to be performed to bring the security of a system to a substantial level.

For instance, during development and maintenance, defining security requirements, using secure design patterns, reviewing designs against best practices, hiring developers trained in security or training developers in it, using automatic tools to verify code, using peer review to verify critical code, using programming languages that are intrinsically less prone to vulnerabilities and planning and performing security testing are all activities that help improve the security of a system right from the beginning.

During operation, making sure configuration stays correct, testing the deployed system, performing penetration testing, educating and training users to operate the system in more secure ways, testing new threats against the system to make sure it is not vulnerable to them, monitoring most of the system's activities to detect anomalies and always considering the system's entire stack of technologies (hardware, drivers, operating systems, etc.) are other examples of activities that improve security.

Finally, some activities can actually be done both during development or maintenance and during operation. One of those activities is the security risk assessment of the system's architecture. It can be done in the design phase to develop an architecture that will likely have fewer exploitable vulnerabilities and it can also be done while maintaining the system to evaluate the impacts of changes on security risks.

This paper presents a methodology named *Software Architecture Risk Analysis* (SARA). This methodology is based on established risk analysis processes so it is coherent with actual practices. It also structures the use of the analyst's knowledge in security, focuses on quick, scoped, repetitive and complementary assessments and finally, emphasizes participation of system architects, designers and users. Section 2.0 covers the SARA methodology in detail. Section 3.0 presents one selected SARA application. The paper is then concluded.

2.0 SOFTWARE ARCHITECTURE RISK ANALYSIS (SARA) METHODOLOGY

SARA is a lightweight qualitative security risk analysis methodology to assess software system architectures. It is easy to use and enables quick, repetitive and scoped assessments. While it is designed to be used mostly by security experts and professionals, security novices can use SARA to structure their learning.

2.1 Design Objectives

The following requirements were used during the design of SARA:

- Stay coherent with established risk analysis practices
- Focus on structuring the use of the analyst's knowledge (instead of imposing specific knowledge)
- Focus on quick, scoped, repetitive and complementary assessments
- Emphasize participation of system architects, designers and users

2.2 Rationale

Despite the many risk analysis methodologies that already exist, SARA was developed because:

- No complete methodology could be found to assess software system architectures. A few methodologies were documented but complete documentation could not be found. It was thus cumbersome to use them on real cases.

- Security must be considered during system engineering, not only as protection mechanisms added after deployment. Avoiding some vulnerabilities right from the beginning is certainly good. But moreover, encouraging simple and quick activities related to security during system development increases the stakeholders' future buy-in into security and the development teams' motivation to build better systems. Indeed, the feeling of building better systems usually brings motivation to go further and raise the bar.
- It can be used to assess the security of existing software system architectures. These assessments can be used to compare available solutions or even more interestingly, to fix owned and controlled systems.

2.3 Uses

SARA is good for:

- Assessing the security risks of a software system architecture, one component at a time. Each component usually takes one to three weeks to assess when done by security experts. The architecture is thus assessed in short complementary iterations, each time on a different component, instead of as a whole. If all short assessments are done by the same people, the architecture's big picture becomes clearer and clearer as assessments are completed and thus, components' integration and their interactions naturally become part of considerations in future assessments.
- Slowly increasing stakeholders' buy-in in security-related activities inside projects. Stakeholders can more easily approve the first security-related activities, like security risk analysis, if they demand less resource. Moreover, since it is usually easier to get good results for simpler tasks, when stakeholders realize what was found, they generally become motivated to go forward and approve more security-related activities. Therefore, SARA facilitates the adoption of security considerations in projects.

SARA is not:

- A methodology to perform certification and accreditation (C&A). There are many such methodologies available and they are usually endorsed by particular accreditation entities.
- A silver bullet. It does not find all security vulnerabilities in software system architectures. To maximize security, you must combine risk analysis with better design, reviewed implementations (with peers and tools), thorough testing, deployed protection mechanisms, monitoring, log reviews, etc.
- A methodology to magically transform anyone into a security expert. It helps novices structure their learning but does not provide them the background knowledge they need.

2.4 Terminology

Before presenting the steps involved in SARA, it is appropriate to give two short definitions of risk and risk management within the present context.

Risk is "a function of the likelihood of a given threat-source's exercising a particular potential vulnerability, and the resulting impact of that adverse event on the organization" [1]. In other words, risk is the measure of the gravity of what happens if an attack is successful, weighted by the likelihood of the attack's occurrence; it measures how much should someone worry about something.

Risk management is "the process of identifying risk, assessing risk and taking steps to reduce risk to an acceptable level" [1]. It is divided into three sub-processes:

- risk analysis (also known as risk assessment),
- risk mitigation and
- evaluation and assessment.

From that definition, identifying and assessing risk is thus part of risk analysis, taking steps to reduce risk to an acceptable level is risk mitigation and the final evaluation and assessment sub-process measures the mitigations' effectiveness. SARA is about the first sub-process: risk analysis.

2.5 Methodology

SARA was derived from existing risk analysis processes, mainly from NIST 800-30 [1] and Cigital's process [2]. The objective was to start with existing methodologies, since it was possible, and thus not be too different from the current best practice.

Figure 1 gives the nine steps of SARA.

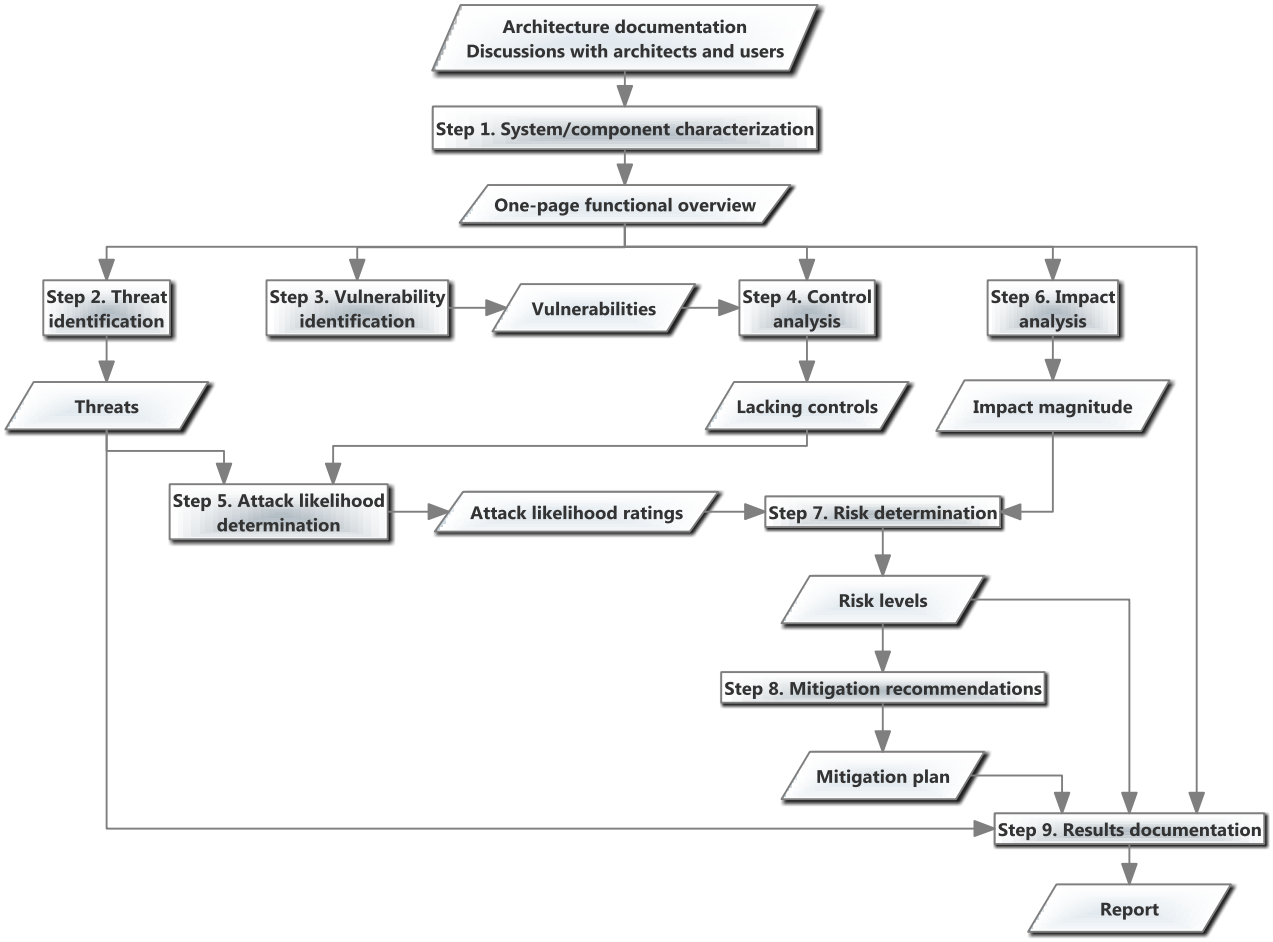

Figure 1: SARA Methodology

Steps 2, 3, 4 and 6 can be, and usually are, done in parallel. Each of the nine steps is detailed in the following subsections. For each step, a table provides additional information on its inputs and outputs, as well as on how it is performed.

2.5.1 Step 1 – System/component characterization

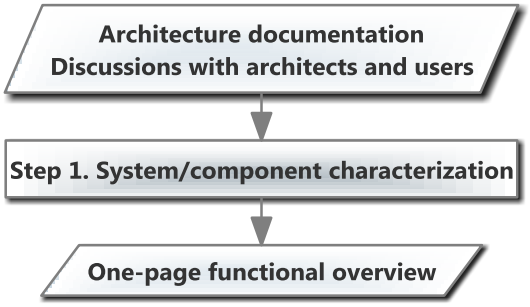

Figure 2: Step 1 – System/component characterization

Table 1: Inputs, output and details of Step 1 – System/component characterization

Inputs	• All architectural documentation that can be found on the software system. Usually, it is not a lot. • Most importantly, access to the system's lead architect or other experts in the system's architecture. A three-hour structured chat with such an expert is more efficient and effective than a week spent alone looking at documentation. • Ideally, access to key users of the system. When they are accessible, it is possible to assess not only technological security risks but also security risks coming from how the system is used.
Step 1: System/component characterization	• This first step is the basis of all others but the goal is not to understand everything at once. The goal is simply to determine which component to analyze next. • From the documentation and discussions with the architecture's expert(s), determine what is the next most security-critical component to analyze in the architecture. Develop a one-page functional overview of that component including its implemented security control mechanisms. • If you are at a point where you have assessed the components interfacing with the one you are selecting, take the time to document assumptions on data moving between the current component and the interfacing ones. • If you have access to users, briefly document how that component is used inside the system.
Output	• The selected component's one-page functional overview, including its implemented security control mechanisms and potentially including assumptions on data coming in and going out of that component and how the data are used in the system.

2.5.2 Step 2 – Threat identification

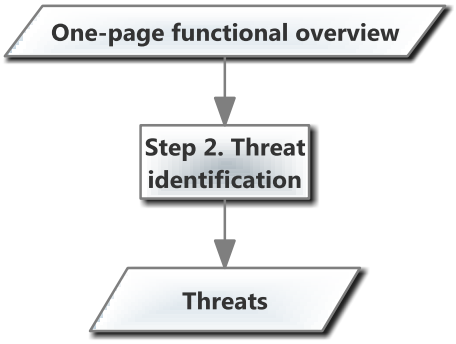

Figure 3: Step 2 – Threat identification

Table 2: Inputs, output and details of Step 2 – Threat identification

Inputs	• The component one-page functional overview. • Security experts should already follow threats' evolution so threats should already be part of their knowledge. To be more systematic about potential threats, the CAPEC website [3] is a good structured view of attacks and threats. Techniques also exist to discover threats, such as threat modelling and threat risk assessment (TRA) processes. • For novices, countless sources of information are available. Online conference proceedings, recorded presentations and hacker forums and communities are good sources of information, while combined with Google searches to find complementary information. While novices cannot expect to perform good security assessments, this methodology can help them structure their learning.
Step 2: Threat identification	• Going through your list of potential threats, determine if each threat is applicable to the component. At this point, do not consider eventual protection mechanisms that could have been implemented. Simply answer the question: "Can this threat be used against the component?".
Output	• Threats identified as applicable to the component.

2.5.3 Step 3 – Vulnerability identification

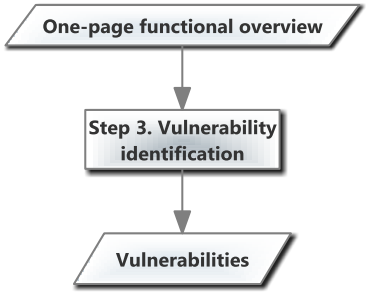

Figure 4: Step 3 – Vulnerability identification

Table 3: Inputs, output and details of Step 3 – Vulnerability identification

Inputs	• The component one-page functional overview. • Security experts should already follow how most common vulnerabilities happen so these should already be part of their knowledge. However, many information sources are helpful to structure that knowledge, such as Secure Design Patterns [4], SANS TOP 25 Most Dangerous Software Errors [5], OWASP Top 10 [6] (for web applications), CWE [7], Open Source Vulnerability Database [8] and CVE [9]. Knowledge of how vulnerabilities work at the code level is very important even when working at the architecture level because it serves as a basis to understand why and how architecture level mechanisms are effective. • For novices, countless additional sources of information are available. As for threats, online conference proceedings, recorded presentations and hacker forums and communities are good sources of information, combined with Google searches to find complementary information. • When code is available (especially source code) and the assessment's scope and resources permit, vulnerability scanning tools can also be used. However, developers knowledgeable with the code base must then be available to analyze the results given by these tools to filter false positives out.
Step 3: Vulnerability identification	• Using your knowledge of potential vulnerabilities, determine which ones the component is likely vulnerable to. At this point, still do not consider eventual protection mechanisms that could have been implemented. Simply answer the question: "Can this vulnerability be present in the component?". • In most assessments, the exception being when the component is at its third or fourth assessment, vulnerability identification should not be conducted in too much depth. This step is primarily here to feed the next one by providing inspiration for controls that should be implemented.
Output	• Vulnerabilities identified as likely present in the component.

2.5.4 Step 4 – Control analysis

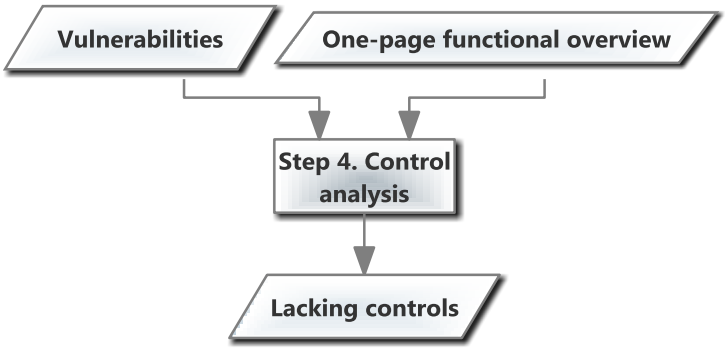

Figure 5: Step 4 – Control analysis

Table 4: Inputs, output and details of Step 4 – Control analysis

Inputs	• The component one-page functional overview. • Vulnerabilities identified as likely present in the component. • Again, security experts should know which controls are effective against which vulnerabilities. But for novices, CSIS' 20 Critical Security Controls [10] is a good source of information. Of course, Google searches are also always helpful.
Step 4: Control analysis	• Determine if the implemented controls are effective against the identified likely vulnerabilities.
Output	• Controls that are lacking.

2.5.5 Step 5 – Attack likelihood determination

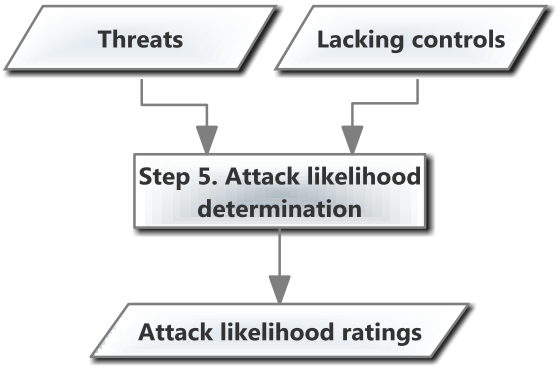

Figure 6: Step 5 – Attack likelihood determination

Table 5: Inputs, output and details of Step 5 – Attack likelihood determination

Inputs	• Threats identified as applicable to the component. • Controls that are lacking.
Step 5: Attack likelihood determination	• For each identified threat, Table 6 is used to determine attack likelihood. • To determine threat likelihood, the analyst should answer the question: "Does the attack require skill or significant resources?". The more skill or resources are required, the lower the likelihood. • To determine control effectiveness, the analyst should answer the question: "Is there a control in place to counter the threat?". If not, the control effectiveness is low. Otherwise, the analyst judges how effective the control is against the threat.
Outputs	• An attack likelihood rating (high, medium, low) for each threat, which identifies the likelihood of the threat successfully exercising a vulnerability.

Table 6: Attack likelihood determination function

		Threat likelihood		
		High	Medium	Low
Control effectiveness	Low	High	High	Medium
	Medium	High	Medium	Low
	High	Medium	Low	Low

2.5.6 Step 6 – Impact analysis

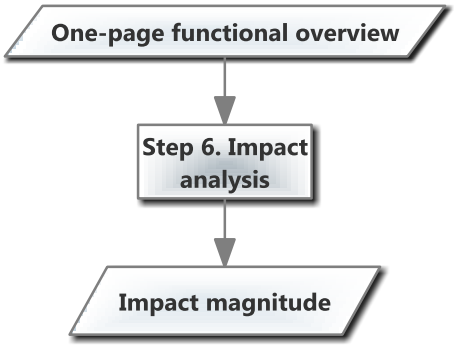

Figure 7: Step 6 – Impact analysis

Table 7: Input, output and details of Step 6 – Impact analysis

Input	• The component one-page functional overview.
Step 6: Impact analysis	• Using interviews with project stakeholders such as senior managers, business operations managers and IT security program managers, determine the impact of a failure of the component. • The impact is usually high because the component was selected for its security-criticality. However, as more and more assessments are done, less security-critical components are selected and thus the impact likely decreases. It is still important to take a few minutes to talk with the stakeholders to confirm the impact. Sometimes, what was first considered a non-critical component can turn out to be more important than expected.
Output	• Impact magnitude (high, medium, low).

2.5.7 Step 7 – Risk determination

Figure 8: Step 7 – Risk determination

Table 8: Inputs, output and details of Step 7 – Risk determination

Inputs	• Attack likelihood ratings. • Impact magnitude.
Step 7: Risk determination	• For each potential attack, Table 9 is used to determine its risk.
Output	• Risk level (high, medium, low) for each potential attack.

Table 9: Risk determination function

		Attack likelihood		
		High	Medium	Low
Impact magnitude	High	High	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

2.5.8 Step 8 – Mitigation recommendations

Figure 9: Step 8 – Mitigation recommendations

Table 10: Input, output and details of Step 8 – Mitigation recommendations

Input	• Risk level for each potential attack.
Step 8: Mitigation recommendations	• Risks are prioritized by their level and by evaluating their mitigation cost-benefit ratio. • New controls and component modifications are recommended to eliminate or mitigate each risk. A mitigation plan can be recommended by proposing mitigations in the order of their corresponding prioritized risk.
Output	• Recommended controls and modifications in an ordered mitigation plan.

2.5.9 Step 9 – Results documentation

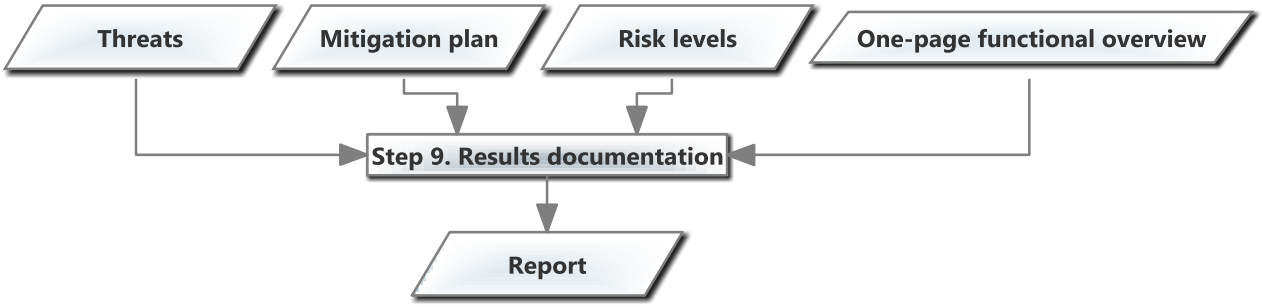

Figure 10: Step 9 – Results documentation

Table 11: Inputs, output and details of Step 9 – Results documentation

Inputs	• The component one-page functional overview. • Threats identified as applicable to the component. • Risk level for each potential attack. • Recommended controls and modifications in an ordered mitigation plan.
Step 9: Results documentation	• A report is written to describe the component, the identified threats, the risk level for each potential attack and the final recommendations on controls and modifications to implement in an ordered mitigation plan. • It can be surprising that low risks are documented in the final report. Indeed, it can be tempting to omit them since, after all, they are low risks. However, it is in fact very important to keep them in the report because it must be explained why these risks are low. As the component is most likely in continuous evolution, designers and developers must understand the conditions that must remain satisfied for these low risks to stay low while the component is modified. Otherwise, these low risks could easily become high in the next assessment of this component.
Output	• The final risk analysis report for the component.

3.0 SELECTED APPLICATION

SARA was applied to military system components embedded aboard Canadian Forces platforms. Vulnerabilities were found in both the technologies used in those systems and the systems' procedures of use. Since details about these assessments are controlled goods, this paper needs to stay generic about which vulnerabilities were found in which system components.

The current selected application is an assessment that was done on a software component consisting of a few computers used to share information among systems of a Canadian Forces aircraft. Since it was the first assessment of this kind being performed on this component, it was done at a fairly high level, thus considering Operating System and application versions used, but no actual code, and Standard Operating Procedures used with the component. For the same reason, the assessment took more than one to three weeks to perform (six weeks in fact). Stakeholders and key players, such as the lead developers of the component, were actively involved in the assessment.

This first assessment targeted "low hanging fruit": security risks that are the most obvious to spot and mitigate. However, it does not mean that these risks have little impact. Mitigating these security risks usually has a very high benefit compared to the cost of the mitigation and constitutes the first line of defence that should be put in place.

3.1 Low Importance Risks

As stated in the methodology, one could think that covering the low importance risks of a system is not essential. It is obviously true that low importance risks should not usually be mitigated first. However, understanding why the low risks are low is key to keep them low as the component evolves. Because computer systems are maintained and modernized, security risks that are low at one point are never guaranteed to stay low during the

entire systems' lifetime. System engineers must understand why those risks are low and maintain the system consequently to keep them low.

The following is a summary of some selected reasons why low risks were considered low during the assessment:

- Potential attacks necessitate very good understanding of the component architecture which makes them unlikely, very targeted attacks;
- Data files use very simple file formats and thus their viewers are not usually vulnerable to attacks;
- Data files and their applications are not widespread so publicly known attacks against them are scarce or inexistent. An attacker looking for such an attack would thus most likely be targeting the specific component and this is unlikely.

3.2 Medium and High Importance Risks

Medium and high importance risks should be mitigated, especially when they are the security "low hanging fruit" as they are in this assessment.

The following is a non-controlled goods summary of some selected medium importance risks found during the assessment:

- The component uses image files and viewers for which there are known attacks but since the images come from DND sources, the chances that they are infected is medium and not high.
- Many data files loaded in and produced by the component are stored unencrypted. If the component gets compromised, these data files are vulnerable to theft or corruption. However, stealing or corrupting those data files would constitute a very targeted attack which was assessed as a medium risk instead of high.
- The component uses an FTP server with a few known vulnerabilities. Because that FTP server software is not a widely spread one, this risk was considered medium instead of high.

And the following is a non-controlled goods summary of some selected high importance risks found during the assessment:

- A single storage medium provides the data interface between many systems and the assessed component. Any storage medium that is frequently connected to different systems has a high risk of becoming infected if minimum care is not taken.
- Only one antivirus software is used to protect a system used to load data in the component. Antivirus software is not particularly effective at detecting malware, so using only one results in a significant chance of not detecting some malware.
- One type of user logs in as an administrator in a system used to load data in the component, even though administrative privileges are not necessary.
- Many operating systems used by the component are out of date. This constitutes a high risk, of course, because many known vulnerabilities exist in those older OSes.

3.3 Recommended Mitigation Plan

This is a non-controlled goods summary of the mitigation plan that was recommended at the end of the assessment:

- Setup a dedicated computer equipped with multiple antivirus software (commercial solutions exist) and modify the procedures so that all storage devices and transferred data files are scanned with this computer prior to being used.
- Reduce the use of removable media to the minimum because they can become infected themselves. Use network transfers whenever possible and scan files with the computer in the bullet above.
- Make sure that all user accounts on the component and systems used to load data into the component have the appropriate minimal required privileges with respect to the level of access they need.
- Keep Operating Systems and applications updated as much as possible.
- Determine the cost of modifying the component to work with encrypted data files, both as inputs and outputs. Implement those modifications if the cost-benefit analysis is positive.

4.0 CONCLUSION

The SARA methodology is not very different from other risk analysis methodologies. This was a design objective to stay as close as possible to existing practices. Despite the fact that SARA must be performed by security experts, it does not impose specific knowledge on them, only structure. And novices can use that structure while they learn.

An example use of SARA on a component architecture, and its Standard Operating Procedures and Operating Systems, was given. Low importance risks were identified. Understanding why those risks are low in the current component version is key to keeping them low in the future. Medium and high importance risks were also identified and a mitigation plan was provided.

This first assessment encouraged buy-in in security-related activities among the component's stakeholders. Future assessments are currently being planned on other components.

5.0 ACKNOWLEDGEMENTS

The author wants to thank his partners in the Canadian Forces who have accepted to actively participate in security assessments. SARA would also not have been possible without the hard work put into existing risk analysis methodologies like OCTAVE, NIST 800-30, Cigital's, OSSTMM, HTRA, etc., by countless other researchers.

Finally, thank you to Daniel U. Thibault for his comments on this paper.

6.0 REFERENCES

[1] G. Stoneburner, A. Goguen and A. Feringa, Risk Management Guide for Information Technology Systems: Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-30, National Institute of Standards and Technology, Gaithersburg, MD, 2002; http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf.

[2] P. Hope, S. Lavenhar and G. Peterson, "Architectural Risk Analysis", Cigital Inc., 2005; https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/architecture/10-BSI.html.

[3] The Mitre Corporation, “CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC)”; http://capec.mitre.org/.

[4] C. Dougherty, K. Sayer, R. Seacord, D. Svoboda and K. Togashi, “Secure Design Patterns”, Software Engineering Institute; http://www.cert.org/archive/pdf/09tr010.pdf.

[5] The SANS Institute, “SANS TOP 25 Most Dangerous Software Errors”; http://www.sans.org/top25-software-errors/.

[6] OWASP, “OWASP Top Ten Project”; http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[7] The Mitre Corporation, “CWE - Common Weakness Enumeration”; http://cwe.mitre.org/.

[8] Open Source Vulnerability Database, “OSVDB: The Open Source Vulnerability Database”; http://osvdb.org/.

[9] The Mitre Corporation, “CVE - Common Vulnerabilities and Exposures”; http://cve.mitre.org/.

[10] Center for Strategic & International Studies, “20 Critical Security Controls”; http://www.sans.org/critical-security-controls/.