Defining a Security Metric: A Modern Search for the Holy Grail.

Christopher Foley

Z15092976

Florida Atlantic University, Boca Raton, Florida

Abstract

The search for a quantifiable security metric is of great interest to systems developers, engineers and managers. No standard definition exists for a secure system. While it is agreed that it should be possible to create a ratio between known threats and implemented safeguards, there is no standardization or clear definition. Similarly there is no clear definition for legacy systems. This paper examines some of the methods proposed. This paper suggests that for certain implementations a degree of security may be calculated, however for others this may not even be possible. Similarly, when systems are combined their security is not necessarily additive and may be detrimental to the overall security of the system under development. Thus the search for quantifiable security is a search for something that once existed (when there were few computers and even fewer hackers), but now no longer exists.

Introduction

Bridgekeeper: Hee, hee, heh. Stop. What… is your name?
Arthur: It is 'Arthur.' King of the Britons.
Bridgekeeper: What… is your quest?
Arthur: To seek the Holy Grail.[1]1

In their work on *Measuring Security,* Stolfo, Bellovin and Evans begin by reminding us of the quote from William Thompson, 1st Baron of Kelvin "Lord Kelvin" where he declares that if we can not measure something, we can not fully understand it.[2] With wide range connectivity, everytone is talking about keeping systems secure. It is assumed that a developer of a system that contains/uses critical information desires to build a system that is secure and will be used. In the world of security much has been written about the desire to obtain a meaningful measurement to permit a system to be evaluated on the basis of its ability to resist attacks. Fifty years ago, security involved a security guard, likely close to retirement, walking a predefined route and recording their location with a unique key or identifier. With the advent of data networks and the interconnectivity provided by the

internet threats involve more than the typical “midnight requisition” from an intruder. Todays security guard will need different tools and his walk around the perimeter will most likely be performed by an automated test tool.

In a perfect world each software and hardware package would have a number stamped on the side indicating the ability to resist attack. Users could then judge the effectiveness of a product before integrating into their processes. However, security professionals will agree that combining two highly secure products does not guarantee increased security, since the measurement does not take into account interactions between system components. Many modern systems are highly complicated making a definitive security metric an elusive goal, thus the reference to the search for the Holy Grail.

In his 2014 article on security metrics William Sanders states “If systems could be made perfectly secure, security metrics would be simple: a system would be either secure or not.”[3] Sanders points out that given the interconnectivity and complexity of systems a perfectly secure system would not be practical given real world time and cost considerations. It is well known that for the world of hardware MTBF can be easily calculated for most physical components and using metrics/policies physical security can be reduced to a metric that can be compared to known or comparable entities.

Individuals and companies face increasing risk to their brands, reputations, assets and intellectual property due to numerous risks.[4] When we talk about security, we are referring to the ability of a system to protect it’s assets.[5] It is the intent of this paper to explore different methods of defining a security metric and discuss methods of defining a system wide metric. Authors have proposed numerous metrics for determining security of a system during the software design process[6] while others have proposed a metric for development based upon patterns, which involves a known threat matrix.[5] We will also explore security for projects in the field.

In their work *Measuring Security,* A. Bilbao and E.Bilbao developed a metric based upon ISO31000 which compares physical security and operations matrices with designated policies. They conclude with the statement “The need for measuring elements in the performance of security is evident.”[7] Although their work and that of ISO 31000 is concerned with physical security, the security of a distributed system must begin with the security of physical components, therefore my first discussion will be an overview of the ISO31000 standards. The more complicated part and the Holy Grail of software security is securing software driven systems. Therefore , the second part of the discussion will involve a review of Software with a focus on two development practices, object oriented design and development and its complement pattern oriented design and development.

Background

When we discuss *Security Metrics* we need to be clear what we are discussing. *Security* is the

ability to protect data and systems from exposure.

Following from the advice of Lord Kelvin a *metric* is a measure, which implies we intend to measure or create a numeric equivalent that can be compared to similar values and find an objective evaluation of a system.

Physical Security

Authors have attempted to define Systems Security and develop metrics to measure the degree of security of a system. In 2009 and 2018, the ISO published a standard for Risk Management (ISO 31000: Risk Management standard) which attempted to define a standard for physical security of systems.[8] They defined a process with continual feedback and accountability within an organization as shown in Figure 1: Relationship between Risk Management Principles, Framework and Process, below.

The ISO 31000 standard, defines training, accountability and feedback which we can then apply to software. Key to the standard is the framework shown above. Policies (Principles) create mandates and a corporate commitment to security. Companies then implement standards to secure their systems against nefarious actors where they then apply a cycle of communication, evaluation and monitoring. We learned from the Equifax breach in 2016, no training, feedback or accountability. The lack of any one of the three will produce disastrous results.[A]

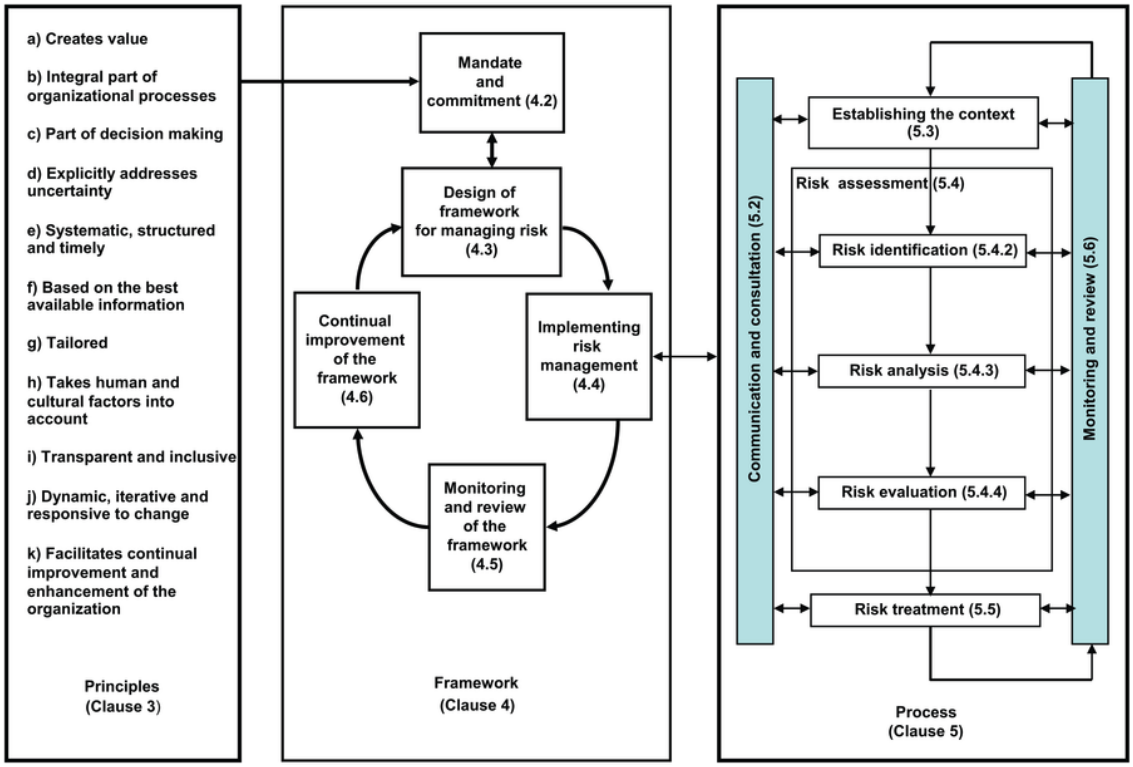


Figure 1: Relationship between Risk Management Principles, Framework and Process

A Disclaimer: the authors data was identified as being subject to a breach and Equifax provided 1 free year of reputation monitoring which, while not admitting any liability, was subsequently extended another year.

Measuring Security – The Holy Grail

Measuring security is a non trivial task. Many of the top system security professionals have backgrounds in mathematics/science where they learned that if it can be studied, its should be measured. If it can be measured, it can be compared and if it can be compared qualitative judgments can be formed about the state of security. If security were absolute it would be, as noted earlier, a simple boolean operation: secure or not secure. We can divide the security of a system into two physical security and system security. Physical security, although not often discussed when determining systems security, is the first step. When the author first started working, the new employee orientation included a reminder from his supervisor that "we keep the door to the computer center locked at all times. If someone walked in with a magnet, they could walk away with everything." This admonition still applies, secure the perimeter and control access, first to the physical building then the logical systems. An example of this was the 2017 Equifax breach[B], which is being studied extensively in universities[C] where the entire corporate work product was placed in a location accessible from the Internet with one login (or carefully constructed HTTP/GET request). [9] Where it could be determined that reasonable security was not being followed and that the advice of external auditors/experts was being ignored.

ISO 31000

When companies face risk, it is no longer just to their physical plant, but they also face damage to their reputations and subsequently their bottom line. The ISO31000:2018 recommendations define risk as the "effect of uncertainty on objectives". Key to the standard is the establishment of policies and procedures with continual feedback to analyze the effectiveness. Although not a checklist and a specific standard, ISO 31000 provides a framework of continuous feedback. The ISO 31000 standard was created with the intent to focus on the

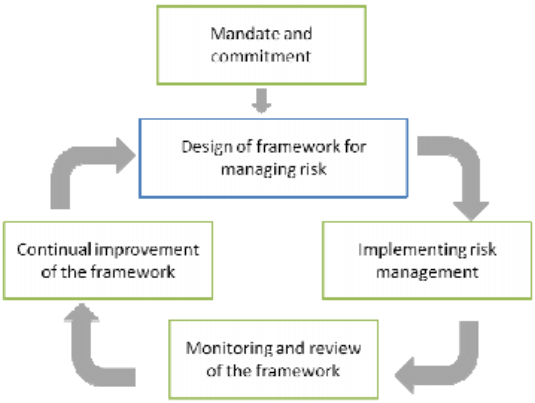


Figure 2: ISO 31000 Framework

B a.k.a. "The Holy Fail", a failure of security in which policies were undefined or ignored, security was never considered as part of the design and accountable individuals were held accountable for skill sets that they did not posess.

C CIS6370 – Computer and Internet Security, Florida Atlantic University (Fall 2017).

creation and protection of value.[4] Referring to Figure 2: ISO 31000 Framework it is evident that the process begins with commitment and policies established by management. To determine the effectiveness of these policies the implementation can involve a series of checklists and tasks to ensure security. Monitoring and review of the policies will determine if they are effective and if they are being followed. As changes are made the framework is modified and altered.

In *Measuring Security[7],* Bilbao et al. argues that the collection of metrics gives quantitative data for managers and security professionals to use in their guidance. They define metrics which may be used to compare to similarly sized plants performing the same tasks.

Software and Systems Security

Measuring software security is considerably more difficult. Numerous authors have developed metrics for measuring software system security. Debra Herrmann has reportedly cataloged over 900 metrics for measuring software security.[10] In their work on security patterns, Fernandez, Yoshioka and Washizaki argue that when a system is built with patterns a security ratio may be defined[5]. In it they argue that a security threat T_i may be constructed mathematically as $T_i = \sum_{1}^{j} T_{i,j}$. To stop an attack, they state, it is only necessary to interrupt one $T_{i,j}$. Likewise it is clear that for a given system S composed of finite subsystems S_i we have a similar summation of systems where $S = \sum_{k=1}^{n} S_k$. Thus our threat matrix becomes $T = \sum_{k=1}^{n} \sum_{1}^{j} T_{k,i,j}$ where k is the number of subsystems and *i* and j represent the threats and steps for each step. If the systems are built using patterns, the threats to the system T may be catalogued and the number stopped N may define a ratio $\frac{N}{T}$ which would give an indication of the system security.

Similar to the metrics suggested by Bilbao et al.[7], Stolfo et al.[2] argue that relative metrics may be the best approach to measuring security. Fundamental their argument if *f* is a system and *f'* is a system derived from *f* then a comparison between attributes (*a)* of each system is valid. Therefore we are safe in comparing *f(a)* and *f'(a).* While comparing attributes we need to define which attributes whe are comparing. Stolfo argues that we can scale different metrics for each system. They present different metrics that we could use to compare. Stolfo presents a security argument framework, which they call the Measurement Based on Security Argument (MBSA) based on a degree of belief in a security claim. They base their framework on four principles: Claim, Evidence, Argument and develop a Metric. The key to their argument is a confidence assessment and interpretation. They outline their argument structure similar to the following:

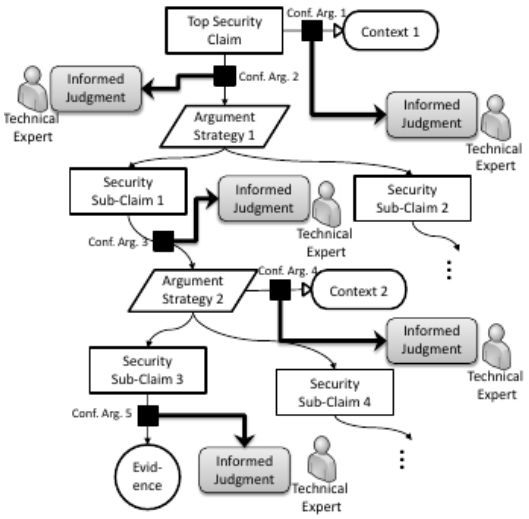


Figure 3: MBSA - Confidence Assessment

The confidence assessment has security arguments or claims which are then associated with a vector of measurements. Each vector is intended to demonstrate confidence in the claims.

The metrics are presented to the decision makers who are responsible for the ultimate security decisions. The methodology has benefits, but is highly labor intensive for complicated products and would require, for practical use, a metric for all shared/open source products used in a system.

Savola indicates that while security may not be measured, we can measure factors leading to it. Savola also surveyed experts and determined that the three greatest criteria in security are accuracy, the ability to measure and meaningfulness.[11] Surprisingly, the study showed that security experts could only form a weak consesus on the importance of usability. The view was that most security factors measured are subjective.

Patterns and Object Oriented Designs

Measuring security at the design phase of a project is not normally done. Through the use of patterns and object-oriented design techniques a measure of security may be discovered. A pattern is a solution to a recurring problem in a specific context.[12] A pattern is usually expressed using UML. The pattern is described using benefits and consequences which will describe the effect of the pattern and the problem it is intended to solve. A pattern may be applied to another pattern and further combined (composite patterns). Patterns are a description of a solution and do not impose a particular development style on a designer or developer. A subset of patterns are *Abstract Security patterns* which suggest patterns to create a solution to a security threat, policy or regulation [5].

With respect to security metrics, the use of patterns provides a reference for the security metric. Each security pattern is designed to combat a series of threats. The system can be inspected to verify that the pattern is present during development. Thus a series of threats can be neutralized by the development process. Key to neutralizing the threats

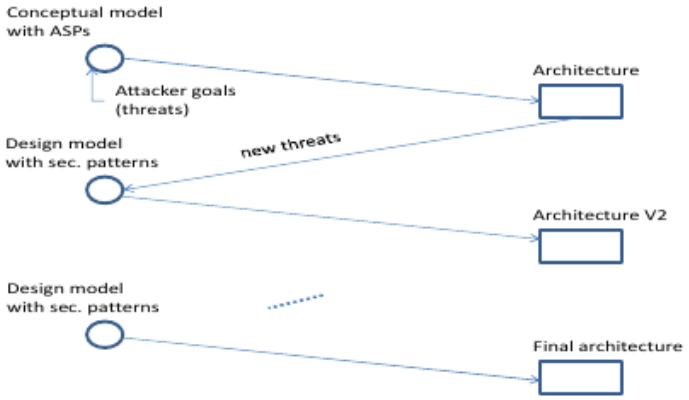


Figure 4: Software Refinement Using Twin Peaks Method

is constant refinement. Refinement is accomplished by a spiraling process commonly referred to the Twin peaks method[13], where the development process begins with the broadest definition and through continuous feedback between Requirements and Architecture a product is finalized.

One approach, identified by Alshammari et al.[14] is to enforce security early in the software development lifecycle. In a 2002 report The National Institute for Standards and Technology (NIST) estimated that flaws in a product found while the product is operational can cost 470-880 times the cost of finding the problem in the requirements phase. Some estimates show that if an architectural (Engineering Change Order) change is required the cost may rise to 2900 times the cost of requirements phase.[15] To develop a metric they assume that the system will be designed using annotated UML diagrams with classes containing predefined labels if they and/or the data they contain are of a critical or confidential nature. Alshammari et al. define 9 metrics related to securing the data and the system which they then apply to a hypothetical case. Their metrics show the graphically the impact of design changes on the classes and those with the greatest vulnerability.

Legacy Systems

The final area of measurement involves already existing software. For purposes of discussion, our review will break it into two areas: Open Source and Existing Systems. The difficulty of defining a security metric for both types of system will be discussed. Defining a security metric for existing code involves a myriad of difficulties. Primarily the concern is having the code and the expertise to carefully review and determine the level of security. Some software may be easy to review, such as a text parsing library. While other software, a cryptographic library, may require special expertise to analyze. Although the methods discussed above can be used, generally with existing systems there is no access to source and/or creation of patterns is difficult. We may need to rely on the authors statements or determine a different metric for security.

Open Source Systems

One of the fastest growing areas of software is the use of Open Source software in systems. Open Source proponents state that Open Source is more secure since it is widely distributed and more widely viewed. To test this claim researchers Jungwoo Ryoo, Bryan Malone, Phillip A. Laplante and Priya Anand examined the source and claims of Open Source software and attempted to determine the reliability of their claims of secure systems.[16] They reviewed 53 projects and searched for keywords indicating some degree of security. What they discovered is that there exists an inconsistent use of security tactics and that a majority of security work involves the use of cryptographic functions to hide data.

By itself this does not indicate that Open Source is more or less secure, however when we examine the statement "Open Source is more secure because there are more people looking at the code suggesting changes, fixes and improving the code." we need to pause for a discussion. Looking at the work of Schryen[17]in 2011 where three conclusions were made:

- Open source security is more a matter of faith than fact. At the time of the publication there were few studies of Open Source Security.
- The type of development does not drive the security, rather the policies of the developing company.
- Incentives need to be made to developers to maintain and patch their open source code.

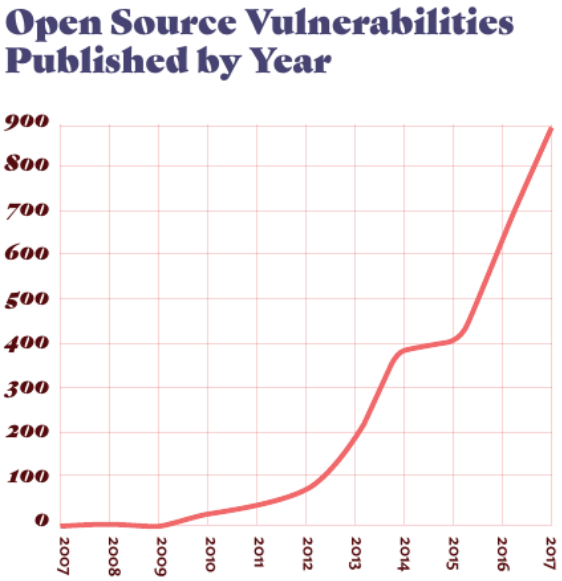


Figure 5: Open Source Vulnerabilities (SNYX Report)

The first conclusion is most controversial, but appears to have some basis in fact. According to the SNYX report "The State of Open Source Security 2017" [18]there seems to be a wide chasm between secure Open Source projects and security. According to the study over 43% of Open Source code maintainers have never conducted a security audit. The report indicates that 88% of Open Source code developers notify their users of vulnerabilities via release notes. Only 3% use a service to notify and only 10% file for a CVE. According to their report the number of vulnerabilities seems to be increasing on an annual basis. Equally troubling, but not surprising, is the statistic from the same report that 59% of the maintainers either deprecate the faulty version or simply don't fix problems.

The one bright spot, according to the report, is Red Hat Linux which seems to have a declining number of vulnerabilities. The Red Hat Linux, according to the SNYX, report appears to be going contrary to the trend of increasing vulnerabilities. This is probably due to Red Hat Linux having a corporate support and most Open Source development projects are spare time projects by developers. This follows the recommendation made by Ryoo et al., they recommend that financial incentives be established for Open Source developers and testers. This would be similar to large scale corporate projects (Windows, Oracle, Google) where there are bounties for White Hat hackers and developers maintain as part of their job.

Referring to the now infamous Equifax fiasco, we learned that the fault was due to use of a very commonly used Open Source library that was not patched to eliminate/block the threat. In the SNYX report vulnerabilities exist for a median time of 2.5 years before they are discovered and the

fix has a median time of 16 days from discovery to fix. The report indicates that a shocking 16.8% of recipients do not update their Open Source software and over 38% do not have tools to periodically scan their installed base and check if revised versions are available.

The second observation by Schryen indicates that the product security is driven by the policies of the company doing the development not the type of development. Both open and closed source software products seem to exhibit linear rates of vulnerabilities, the rate at which and types of vulnerabilities patched appear to be driven by other means. Schryen does indicate that patching behavior would change in the open source world with proper economic incentives. It has been observed that many open source packages are being created and maintained by developers who divide their time between their day jobs/careers and open source development. Thus is is believable that many developers are reluctant or resistant to providing free support and maintenance for code throughout the lifetime of the product.

To develop a security metric for Open Source we can submit the code to analysis and determine if it fits any of the qualifications above or we can look at CVE/bugs fixed with respect to the number of installed sites and view over the the lifetime of the product. This assumes that as the number of users increases the number of faults reported/fixed will increase until most vulnerabilities are fixed.

Existing Systems

The biggest concern is establishing a reliable metric for systems that were not developed as Open Source, Patterns or Object-Oriented or for which no source can be determined (eg. NORAD's Missile control systems or Blackberry 10 OS). A simple, but effective, approach would be to use the methodology described by Fernandez, Yoshihoka and Washizaki[5] in which a similar system built with known methods, patterns and structures is compared to an unknown system by subjecting both systems to attack from known threats and comparing the results. A comparative metric can be established much like those used in advertising, in which we state that ours is better than the leading brand ("Better ingredients better pizza"[D] or "...Mom's Robot Oil, made with 20% more love than the next leading brand"[E]).

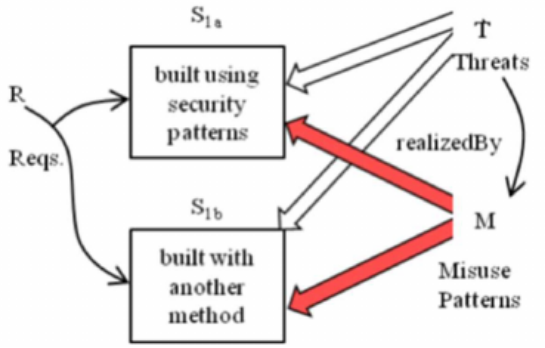


Figure 6: Subjecting Similar Systems to Identical Threats (Fernandez)

This same methodology of creating a system and subjecting it to attack is discussed by Nogoorani, Hadavi and Jalili.[19] In their work they propose creating three security metrics: Probability of attack success, Mean Time to First Breach, System Misuse. They propose building a Stochastic Activity

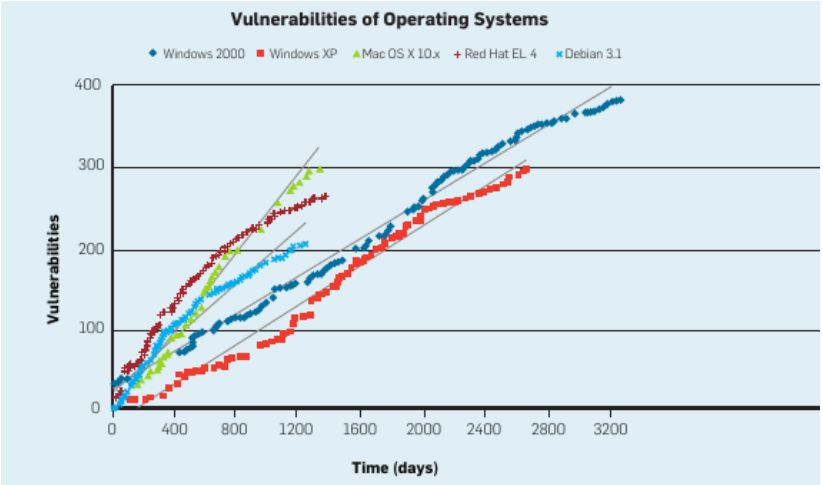
D https://caselaw.findlaw.com/us-5th-circuit/1193659.html where the court held that a tagline "Better Ingredients Better Pizza" is advertising and not absolute comparison.

E Futurama, "Fishful of Dollars", Season 1, Episode 6, one wonders how to quantify love.

Network (SAN) to model the system under attack and derive data from the model. They acknowledge the cost of building the model, however they balance the cost against the need to keep the system secure. The model then measures the progress of an attack and determines the ability of an attacker to take the system from a secure state to insecure. What is different from a simple attack is that their model then allows the system to continue under attack and the extent to which the critical components are compromised is then measured.

Another possible metric for existing systems is using existing data mining techniques to determine or predict the likelihood of software flaws. This has been explored in the work of Khoshgoftaar, Jones, Allen, Hudepol where data mining techniques are used to mine software databases and identify modules with the greatest likelihood of vulnerabilities.[20] Comparable systems may be compared which would indicate the likelihood of vulnerabilities.

The easiest approach is to return to Schryen's comparison of Open/Closed source software . As can be seen from the chart comparing the known vulnerabilities of open and closed source systems, the nunber of vulnerabilities form a linear relationship.[17] Extracting the Operating Systems Vulnerabilities chart as our reference, since it is clear that operating systems must perform similar functions, there are multiple products performing similar functions we see an interesting trend. Looking at a comparison between the vulnerabilities of operating systems vs their time in operation, we see that vulnerabilities seem to take a linear relationship to the time a system is in use. Taking historic data for a system, we could form a comparison between similar systems. It is recognized that this approach does not give a direct security measurement, however one could also create a table of CVE or conduct an analysis of the sources of patches (security, cosmetic or general bug fix). Thus when comparing systems a historic analysis of vulnerabilities and patches over time can provide an indication of the overall quality and inspection of the patches can indicate the number of security related patches.



Related Work

A search of library databases on the keywords “Software Engineering”, “Security Metrics” yielded 297,285 results. The text *Software Security: Building Software In* [32] is referenced by a sizeable number of authors. There is significant work going on in the field of security metrics. In their work in 2018, Morrison, Moye, Pandita and Williams reviewed 4818 papers in which they state they have cataloged 378 unique software security metrics.[21] They state that 85% of the metrics have been proposed and evaluated by their authors with 60% empirically evaluated by others. They found that the bulk of the specification is being done in the implementation and operations phases. Only about 1.5% involved software testing.

ISO31000

The ISO is concerned with creating standards guidance which will lead security policy makers to a better decision. Security metrics are becoming standardized as part of the Common Vulnerability Scoring System (CVSS)[22] and Common Weakness Scoring System (CWSS)[23].

A significant example of standardization is the National Vulnerability Database (NVD) from the National Institute of Standards and Technology (NIST). The NVD provides a central database of vulnerabilities. Their reporting provides a standard way and terminology for referring to vulnerabilities. They also provide tools to analyze vulnerabilities, such as Figure 7: NVD Visualization - Vulnerability Word Cloud found on page 15. NIST also provided in 2010 a Common Configuration Scoring System (CCSS)[24] which is derived from the CVSS it is designed to provide quantitative measures of the security of a system which can be used to provide input as to security configuration issues.

In the October 2018 issue of *Computer Networks* researchers published the results of their investigation of the effectiveness of security metrics using a Hierarchical Attack model and Moving Target Defenses to dynamically change the attack surface of a network.[25,]

Patterns and Object Oriented

Use of patterns and Object Oriented design seems to be the most promising for Software Security. Beginning with a pattern and UML design, tools exist which can validate implementation and ensure that the final product matches the initial design.

It is well known that patterns can guide a developer to a refereed design that takes into many factors that developer may never consider. A critical and growing subset of patterns, is Security Patterns. Security patterns are designed to be included into patterns to protect data or critical systems. A third subset of patterns Security Test Patterns mimic the behavior of an attacker and may be used to repeatedly attack a system or group of systems. Security Test Patterns have the benefit of defining in a safe environment what areas of a system are faulty or weak.[26]

Mumtaz, Alshayeb, Mahmood and Niazi of the King Faud University of Petroleum and Mining are extending the work of Fowler in which they improve security through refactoring designed to remove bad code smells.[F,27] They contend that examination of code and minor refactoring can improve security. The refactored code is then compared to the original UML design and the result is therefore more secure.

Network security is examined in "k-Zero Day Safety: A Network Security Metric for Measuring the Risk of Unknown Vulnerabilities".[28] Here the authors assume that a network will be compromised and attempt to develop a metric where they examine how many zero day vulnerabilities need to exist to compromise a system. The authors of k-Zero contend that it is not important to quantify how many known vulnerabilities we can handle but how many the system is capable of handling before failure.

In their research C. Banerjee, A. Banerjee, and S. K. Sharma[29] define the Misuse Case Oriented Quality Requirements metric Framework which is intended to assist the developer by providing a way of discovering security defects in software during the development process.

Open Source Security

Not surprising the area of Open Source Security is very busy. According to Wen[17], The Black Duck 2017 Report on Open Source Security indicated that 3623 vulnerabilities were discovered in Open Source software in 2016 (a 10% increase over the previous year). Using the Florida Atlantic University Libraries Discovery Service the keywords "Open Source software" and "Software Security" returned over 1,000,000 articles covering both topics. Shao-Feng Wen of the Norwegian University of Science and Technology conducted an examination of the literature concerning open source security and attempted to catalog some of the areas they addressed.[30] Of note is that 47% involved verification of security in open source

In the October 2018 issue of Computers[31], Wen discusses the state of security knowledge sharing in the Open Source Communities. The study was able to show a positive correlation between knowledge sharing and security knowledge in open source communities. More work needs to be done.

Security Metrics

In the Morrison paper [21] noted at the beginning of the Related Work section (page 11) they stated that their goal was to determine security metrics related to phases of software development. Software developers can use the metrics to determine the level of risk and mitigation their systems

F Wikipedia indicates that the term "code smell" was introduced in the 1990's to identify a characteristic of source code that is indicative of a deeper problem. An example of one type of "code smell" which is referred to as "Excessively long line of code" or "God code" would be:

```
new XYZ(s).doSomething(buildParam1(x), buildParam2(x), buildParam3(x), a +
   Math.sin(x)*Math.tan(x*y + z)).doAnythingElse().build().sendRequest();
```

provide and use the metrics to determine areas needing additional work. They see the most often use of a metric is to study the relationship between software and reported vulnerabilities. The most often used metric is vulnerabilities. They conclude by stating that more research is needed in the areas of software requirements, design and testing. Not surprisingly they conclude that the software development security field has yet to agree on a set of metrics.

Conclusions

In his book Gary McGraw begins with “Software Security – the idea of engineering software so that it functions correctly under malicious attack”[32] Parsing this quote we realize how difficult it is to define system security. We first must realize that software is engineered to perform a function, then while performing a function it must protect or be protected from nefarious actors. Software security is a vast rapidly growing field. A user of a system can reasonably ask, “Will my data be secure?”. The developer will answer yes. However, unless the developer is omniscient, no iron clad assurance can be given. What is desired is a metric to measure system security. As was noted, Debra Herrmann has cataloged over 900 metrics for security[10]. Vilhelm Verendel, in 2009, reviewed literature concerning quantifiable security methods and referred to it as a *weak hypothesis* [33](emphasis added) since there were multiple metrics with little validation. With that many it is critical for a developer to find a subset that will give reliable metrics that can be used to provide a metric of system security that a non security professional can use to determine the security of a system. To do this we determine the following factors (in order of priority):

1. How do we use the data we intend to collect? Data used for fault analysis and prediction may be coarser than data used to make a purchasing decision. Data used for policies may be very different from data used for planning.
2. What data do we intend to collect? Failure rates, stress load….
3. How can we collect the data? Collecting failure data is difficult in a well defined system.

First we examined physical security and its importance to systems security. ISO 31000:2018l provides a framework we can use to define and discuss physical security. Key to the model is the creation of and commitment to policies that directly affect security with measurable goals, review and feedback. This ensures that the policies are being implemented and that they solve the desired problem.

We attempted to quantify software security, but like software there are a number of variables. The work of Fernandez, Yoshioka and Washizaki[5] defined two metrics that could be used. First they defined a threat T_i Which was composed of steps T_{ij} it was only necessary to stop one step to stop the threat. This allows a simple calculation of a security metric as the number of threats stopped divided by the number of known threats. Their metric relies on the use of patterns and the pattern catalog which states the threats stopped by the pattern. The concern with this method is that it treats

the threat as a whole before the system changes its state from secure to compromised. However, it seems the simplest and most reliable solution to a complicated problem.

Fernandez et al. also presented a different metric to the problem which could be used with all systems. They proposed submitting a known system built with patterns (f) and a system performing similar functions to the same tests (f'). A comparison between the results of the entities could become a new metric that permits comparison. This idea is similar to that of Nogoorani, Hadavi and Jalili[19] where they proposed utilizing a stochastic activity model to model attacks by a semi-Markov attacker and the system studied to determine the outcome. They defined certain parameters to show system security. They demonstrated their system with an attack on a password authentication subsystem.

Stolfo et al. presents a metric that involves consensus by experts on the security of certain aspects of the system. Their MBSA relies on the informed judgement of experts in certain areas of paractice who can examine the code and determine if it meets certain requirements. While this is a good idea, it is, in my opinion, impractical. Although Open Source is available for inspection, closed source and many commercial products are not available for inspection and companies building secure products may be reluctant or hostile to the idea of submitting their work product to experts not under their employ (even with NDA's signed with blood).

Alshammari[14] argues that security be enforced at the beginning stages of development. This is a good policy for software development teams, however it does not give the user a clear indicator of the security of a system. This is similar to the trend in the 1990's of becoming ISO9000 certified and claiming that it means you have a high quality product. The standard only certified that if a company made widgets, widget having a serial number of 10 was the same as widget 10000000. They did define a useful process or procedure for identifying data or modules at risk. Using UML and Object oriented development procedures, they identified data at risk through use of the designer identifying critical data and critical structures. They then determined the complexity needed to obtain the data and created a metric which could be used to identify areas of concern. They presented hypothetical graphs of data in a manner similar to Stolfo[2] which visually indicated the data/areas most at risk.

Finally we waded into the world of Open Source and "everything else". Building upon a classroom exercise[G] we reviewed the declaration "...more eyeballs means safer code". We looked at research on security in Open Source and determined that according to experts, other than Cryptography there may not be much security in open source. The idea that more "more eyeballs safer code" was equated to puffery of Papa John's ("Better Ingredients Better Pizza") and through the "2017 State of Open Source Security" [18] report from SNYX we realized how true that was. The report emphasized the importance of companies having clear policies involving the use of Open Source and policies involving the periodic auditing and updating of open source code to prevent Equifax

G CIS6375 – Distributed Systems Security – Assignment 02.

sized data breaches.

While we did not examine GDPR, it must be considered for all companies utilizing Open Source code. It's policies specify monetary damages (based on the number of records compromised) to those who fail to safeguard individuals data.

During our examination of Open Source AND close source data, we determined that a useful, but not iron clad, metric was the number of CVE's fixed compared to the total. It was discovered that they appear to form a linear relationship. Measuring the individual graphs/ratios over time can provide an indication of some degree of future performance (unlike my stock broker). Examination of the data also indicated that when incentives are provided to fix flaws (Red Hat Linux) the linear relation can become more like a logistic relation.

An analysis of security reveals that the best defense is a comprehensive set of security policies which:

1. limit access to data
2. limit access to systems
3. provide regular auditing of logs
4. provide regular updating of systems using Open Source Code
5. provide regular training on security matters to employees
6. provide incentives to developers to review and fix Open Source

We conclude with the thought that there are numerous methods of defining and measuring system security. Each method relies on how the system was developed. Finding a universal metric for security is like finding the Holy Grail. Henry Jones, Jr. (Indiana Jones) found the Holy Grail, but couldn't bring it to the world. Indiana Jones is fictional.

Figure 7: NVD Visualization - Vulnerability Word Cloud

ENDNOTE REFERENCES

1 "Monty Python and The Holy Grail (1975) - Quotes", Internet Movie Data Base (IMDB), https://www.imdb.com/title/tt0071853/quotes

2 *Stolfo, Sal, Steven M. Bellovin, and David Evans. 2018. "Measuring Security." IEEE SECURITY & PRIVACY 9 (3): 60–65. Accessed December 6. doi:10.1109/MSP.2011.56.*

3 W. H. Sanders, "Quantitative Security Metrics: Unattainable Holy Grail or a Vital Breakthrough within Our Reach?," in *IEEE Security & Privacy*, vol. 12, no. 2, pp. 67-69, Mar.-Apr. 2014. doi: 10.1109/MSP.2014.31 URL: http://ieeexplore.ieee.org.ezproxy.fau.edu/stamp/stamp.jsp?tp=&arnumber=6798561&isnumber=6798534

4 Tranchard, S. Tranchard, "The new ISO 31000 keeps risk management simple", ISO/News/Archive, https://www.iso.org/news/ref2263.html , published 15-Feb-2018.

5 E.B. Fernandez, N. Yoshioka and H. Washizaki, "Evaluating the degree of security of a system built using security patterns", ARES Conference, August 27-30, 2018; Hamburg, Germany, copy provided by author.

6 K. Sultan, A. En-Nouaary and A. Hamou-Lhadj, "Catalog of Metrics for Assessing Security Risks of Software throughout the Software Development Life Cycle," *2008 International Conference on Information Security and Assurance (isa 2008)*, Busan, 2008, pp. 461-465. doi: 10.1109/ISA.2008.104 URL: http://ieeexplore.ieee.org.ezproxy.fau.edu/stamp/stamp.jsp?tp=&arnumber=4511611&isnumber=4511515

7 A. Bilbao, E. Bilbao, "Measuring Security." *2013 47th International Carnahan Conference on Security Technology (ICCST), Security Technology (ICCST), 2013 47th International Carnahan Conference On*, 2013, doi:10.1109/CCST.2013.6922054.

8 "ISO 31000 – Risk Management", International Organization for Standardization, https://www.iso.org/files/live/sites/isoorg/files/store/en/PUB100426.pdf

9 L. Macvittie, "Security's "Rule Zero" Violated Again With Zero-Day Apache Struts 2 Exploit", F5 Networks, Inc., https://www.f5.com/labs/articles/threat-intelligence/securitys-rule-zero-violated-again-with-zero-day-apache-struts-2-exploit-25619

10 Herrmann, Debra S. 2007. *Complete Guide to Security and Privacy Metrics : Measuring Regulatory Compliance, Operational Resilience, and ROI*. Boca Raton : Auerbach Publications, c2007. http://ezproxy.fau.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,cookie,url,uid&db=cat06361a&AN=fau.021509385&site=eds-live&scope=site.

11 Savola, Reijo M.1, Reijo.Savola@gmail.com. 2013. "Quality of Security Metrics and Measurements." *Computers & Security* 37 (September): 78–90. doi:10.1016/j.cose.2013.05.002.

12 F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, M. Stahl. *Pattern-Oriented Software Architecture: A System of Patterns*, Vol. 1, J. Wiley, 1996

13 B. Nuseibeh, "Weaving together requirements and architectures," in *Computer*, vol. 34, no. 3, pp. 115-119, March 2001. doi: 10.1109/2.910904 URL: http://ieeexplore.ieee.org.ezproxy.fau.edu/stamp/stamp.jsp?tp=&arnumber=910904&isnumber=19651

14 B. Alshammari, C. Fidge, D. Corney, "Security Metrics for Object-Oriented Designs." 2010. *2010 21st Australian Software Engineering Conference, Software Engineering Conference (ASWEC), 2010 21st Australian*, 55. doi:10.1109/ASWEC.2010.34.

15 NIST, "*The Economic Impacts of Inadequate Infrastructure for Software Testing*", NIST, Gaithersburg, MD, Planning Report 02-03 Technical Report, 2002, URL:https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf

16 "The Use of Security Tactics in Open Source Software Projects." 2016. *IEEE Transactions on Reliability, Reliability, IEEE Transactions on, IEEE Trans. Rel*, no. 3: 1195. doi:10.1109/TR.2015.2500367.

17 Schryen, Guido. 2011. "Is Open Source Security a Myth?" *Communications of the ACM* 54 (5): 130–40. doi:10.1145/1941487.1941516.

18 "2017 State of Open Source Security", https://snyk.io/stateofossecurity/, last accessed 12-Nov-2018.

19 S. Dorri Nogoorani, M. A. Hadavi and R. Jalili, "Measuring software security using SAN models," *2012 9th International ISC Conference on Information Security and Cryptology*, Tabriz, 2012, pp. 80-86. doi: 10.1109/ISCISC.2012.6408195, URL: http://ieeexplore.ieee.org.ezproxy.fau.edu/stamp/stamp.jsp?tp=&arnumber=6408195&isnumber=6408181

20 Khoshgoftaar, T.M., Allen, E.B., Jones, W. et al. "Data Mining of Software Development Databases", Software Quality Journal (2001) 9: 161. https://doi.org/10.1023/A:1013349419545

21 Morrison, Patrick, David Moye, Rahul Pandita, and Laurie Williams. 2018. "Mapping the Field of Software Life Cycle Security Metrics." *Information and Software Technology* 102 (October): 146–59. doi:10.1016/j.infsof.2018.05.011.

22 Peter Mell, Karen Scarfone, and Sasha Romanosky. 2006. "Common Vulnerability Scoring System." *IEEE Security & Privacy Magazine* 4 (6): 85. http://ezproxy.fau.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,cookie,url,uid&db=edb&AN=52179315&site=eds-live&scope=site.

23 Common Weakness Scoring System (Version 1.0.1), 2014, http://cwe.mitre.org/cwss/cwss_v1.0.1.html

24 K. Scarfone, P. Mell, NIST Interagency Report 7502, "The Common Configuration Scoring System (CCSS): Metrics for Software Security Configuration Vulnerabilities", https://doi.org/10.6028/NIST.IR.7502

ENDNOTE REFERENCES

25 Hong, Jin B., Simon Yusuf Enoch, Dong Seong Kim, Armstrong Nhlabatsi, Noora Fetais, and Khaled M. Khan. 2018. “Dynamic Security Metrics for Measuring the Effectiveness of Moving Target Defense Techniques.” *Computers & Security* 79 (November): 33–52. doi:10.1016/j.cose.2018.08.003.

26 Smith, Ben, and Laurie Williams. 2012. “On the Effective Use of Security Test Patterns.” *2012 IEEE Sixth International Conference on Software Security & Reliability*, January, 108. http://ezproxy.fau.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,cookie,url,uid&db=edb&AN=86572580&site=eds-live&scope=site.

27 Haris Mumtaz, Mohammad Alshayeb, Sajjad Mahmood, Mahmood Niazi “An empirical study to improve software security through the application of code refactoring”, *Information and Software Technology*, Volume 96, 2018, pp. 112-125

28 Wang, Lingyu1, Sushil2 Jajodia, Anoop3 Singhal, Pengsu1 Cheng, and Steven2 Noel. 2014. “K-Zero Day Safety: A Network Security Metric for Measuring the Risk of Unknown Vulnerabilities.” *IEEE Transactions on Dependable & Secure Computing* 11 (1): 30–44. doi:10.1109/TDSC.2013.24.

29 Banerjee, Chitresh1, chitreshh@yahoo.com, Arpita2 Banerjee, and S. K.3 Sharma. 2017. “Estimating Influence of Threat Using Misuse Case Oriented Quality Requirements (MCOQR) Metrics: Security Requirements Engineering Perspective.” *International Journal of Hybrid Intelligent Systems* 14 (1/2): 1–11. doi:10.3233/HIS-170237.

30 S. Wen, “Software Security in Open Source Development: A Systematic Literature Review.” 2017. *2017 21st Conference of Open Innovations Association (FRUCT), Open Innovations Association (FRUCT), 2017 21st Conference Of*, 364. doi:10.23919/FRUCT.2017.8250205.

31 Shao-Fang Wen. 2018. “An Empirical Study on Security Knowledge Sharing and Learning in Open Source Software Communities.” *Computers, Vol 7, Iss 4, p 49 (2018)*, no. 4: 49. doi:10.3390/computers7040049.

32 G. McGraw, *Software Security: Building Security in,* Addison-Wesley Professional (2006)

33 V. Verendel, “Quantified Security is a weak hypothesis”, NSPW '09 Proceedings of the 2009 workshop on New security paradigms workshop Pages 37-50 , doi:10.1145/1719030.1719036