EXPLOITABILITY OF VULNERABILITIES

Draft Assignment Block 2

WM0824 - Economics of Cybersecurity

Yorick Breukers - 4021053
Kirsten Meeuwisse - 4079612
Maria Baltoglou - 4412842
Alma Valencia López - 4514882

Introduction

Everyday multiple software vulnerabilities are listed by the National Institute of Standards and Technology in the National Vulnerability Database. This database is increasing everyday with the detection of new software vulnerabilities. According to the website of CVE a "Vulnerability is a mistake in software code that provides an attacker with direct access to a system or network" (MITRE Corporation, 2013). The presence of vulnerabilities in software causes concern among users and software vendors. Patching these vulnerabilities in order to protect information system is essential, however not all the detected vulnerabilities have to become an incident. The fact that a vulnerability exist does not mean that the vulnerability is actually being exploited. But how do you know if the vulnerability which you are facing will be exploited or not? And with all these different vulnerabilities how do you know on which vulnerability you should focus on in terms of security resources? In this assignment we will provide a suggestion for metrics to answer both questions.

Security Issue

The main goal of this assignment is to provide additional insights into which vulnerabilities are actually being exploited and which vulnerabilities are left out. The security issue lies in the fact that it is impossible to make a system complete secure. The best we can do is to make sure that the system is secure enough to make the required effort to hack the system higher than the perceived award. Being able to focus security efforts on specific vulnerabilities could improve the adds of the defenders, since it allows them to secure their system more efficiently. But this requires knowledge on which type of vulnerabilities are most likely to be exploited. In this assignment we look at which vulnerabilities exist and which are actually exploited.

Data

Two different sources of data are used for this assignment: the National Vulnerability Database (NVD) and the Open Sourced Vulnerability Database (OSVD). Both databases can be linked on the unique key: CVE-identifier. CVE stands for Common Vulnerabilities and exposures. Every CVE-identifier can be categorized into a CWE (Common Weakness Enumeration) category. The CWE category represents a single vulnerability type. For this assignment decided is to focus on four CWE categories namely: XSS (Cross-site scripting), cryptographic Issues buffer issues and sql injections. These four were all in the top 10 list of vulnerabilities in 2013 and/or in the trend watch report for 2015 (OWASP, 2013; Symantec, 2015). "Cross-site scripting (XSS) is the class of web application vulnerabilities in which an attacker causes a victim's browser to execute JavaScript from the attacker with the privileges of a trusted host" (Wasserman, Zhendong , 2008).
Cryptographic issues can occur when encryption is not good or not secure implemented in your system. (Lazar, Chen, Wang, Zeldovich, 2014). "An SQL injection occur when untrusted values are used to construct SQL commands, resulting in the execution of arbitrary SQL commands given by an attacker". (Chen, Wang, Fu, 2015). "Buffer issues and then especially buffer overflow is a source code level vulnerability that allows unchecked inputs to overflow data buffers during memory writing operations." (Wang, Wen, Zhang, 2014)

Acknowledgement

Two different sources of data are used for this assignment: the National Vulnerability Database (NVD) and the Open Sourced Vulnerability Database (OSVD). Both databases can be linked on the unique key: CVE-identifier. CVE stands for Common Vulnerabilities and exposures. Every CVE-identifier can be categorized into a CWE (Common Weakness Enumeration) category. The CWE category represents a single vulnerability type. For this assignment decided is to focus on four CWE categories namely: XSS (Cross-site scripting), cryptographic Issues buffer issues and sql injections. These four were all in the top 10 list of vulnerabilities in 2013 and/or in the trend watch report for 2015 (OWASP, 2013; Symantec, 2015). "Cross-site scripting (XSS) is the class of web application

vulnerabilities in which an attacker causes a victim's browser to execute JavaScript from the attacker with the privileges of a trusted host" (Wasserman, Zhendong , 2008).
Cryptographic issues can occur when encryption is not good or not secure implemented in your system. (Lazar, Chen, Wang, Zeldovich, 2014). "An SQL injection occur when untrusted values are used to construct SQL commands, resulting in the execution of arbitrary SQL commands given by an attacker". (Chen, Wang, Fu, 2015). "Buffer issues and then especially buffer overflow is a source code level vulnerability that allows unchecked inputs to overflow data buffers during memory writing operations." (Wang, Wen, Zhang, 2014)

IDEAL METRICS

Security metrics play an important role in collecting and analyzing data in an organization thus helping it to manage in a better way its performance. The value of metrics in cyber security is reflected on the fact that many critical infrastructures are at risk of attack and the potential protection can be realized only if security is firstly measured (Boyer & McQueen, 2007; Patriciu, Priescu & Nicolaescu, 2006). Practically speaking, a metric is the output of a mathematical model that measures a technical object (Boyer & McQueen, 2007).

The choice of the metrics to be used in a company is important because it provides insights and leads to decisions that can seriously affect the viability of the company (Boyer & McQueen, 2007). For this reason it is critical for any firm to realize which vulnerabilities are going to affect it the most once exploited. In order to facilitate this decision, several institutes have provided standards and codes of practices by using standardized metrics, such as ISSEA and SECMET (Patriciu, Priescu & Nicolaescu, 2006).

Ideally, an organization should be able to know what is the probability that each vulnerability is going to be exploited, what is the impact of an exploited vulnerability on the organization itself and which are the future threat trends. More specifically, table 1 depicts in more detail the ideal metrics that security decision makers should have available. These metrics can be divided in three categories; management, operational and technical ones. The first category provides information about the impact on the organization, the second category is used to understand and optimize the business activities while the latter one provides technical details of the activities.

Management Metrics	• Cost of incidents • Percent of systems with known vulnerabilities • Patch Policy Compliance
Operational Metrics	• Mean incident recovery cost • Mean time between security incidents • Mean time to incident recovery • Mean time to mitigate vulnerabilities • Mean cost to mitigate vulnerabilities • Mean cost to patch • Mean time to patch • Future threat trends
Technical Metrics	• Number of incidents • Vulnerability scanning coverage • Number of known vulnerability

	instances • Patch management coverage • Configuration management coverage

Table 1: Ideal metrics for security decision makers (Source: CIS Security Metrics, 2010).

The above metrics can stand as a succor to the decision makers when they execute the risk analysis by estimating the vulnerabilities' probability, consequences and impact (Tashie & Ghernaouti-Helie, 2007). In this way, they can also identify the strength of the organization's security considering the time a malevolent spent to penetrate in the systems and the time it took for patching. However, it should be noted that the metrics used in practice by firms can differ from the aforementioned ideal metrics.

Metrics in practice

The Common Vulnerability Scoring System (CVSS) is the open industry standard to assessing the severity of the system security vulnerabilities. It is meant to quantify and reflect the overall severity and risk of a computer vulnerability.

These metrics are divided into three submetrics (Mell, Scarfone, Romanosky, 2007):

- **Base score metrics:** Measure the characteristics (intrinsic and fundamental) that do not change over time or different environments. Consisting of:

$$BaseScore = roundTo1Decimal(((0.6 \times Impact) + (0.4 \times Exploitability) - 1.5) \times f(Impact))$$

Explotability Metrics:

$$Exploitability = 20 \times AccessVector \times AccessComplexity \times Authentication$$

1. **Access vector (AV):** Measures how the vulnerability is exploided (Local or Remote). The more remote the attack is, the greater the vulnerability score.
   AccessVector = Local access: 0.395
   Local Network accessible: 0.646
   Network accessible: 1
2. **Access complexity (AC):** Measures the complexity of the attack required to exploit the vulnerability once the attacker has access to the system. The lower the complexity, the higher the score.
   AccessComplexity = High: 0.35
   Medium: 0.61
   low: 0.71
3. **Authentication (Au):** Measures the number of times the attacker has to authenticate to a target for exploit. The fewer authentication instances, the higher the vulnerability score.
   Authentication = No authentication: 0.704
   Single instance of authentication: 0.56
   Multiple instances of authentication: 0.45

Impact Metrics.

$$Impact = 10.41 \times (1 - (1 - ConfImpact) \times (1 - IntegImpact) \times (1 - AvailImpact))$$

$$f(Impact) = \begin{cases} 0, & \text{if } Impact = 0 \\ 1.176, & \text{otherwise} \end{cases}$$

1. **Confidentiality impact (C):** Measures the impact on confidentiality of a successfully exploited vulnerability. Increased confidentiality impact increases the vulnerability score.
   ConfImpact = None: 0
   Partial: 0.275
   Complete: 0.660
2. **Integrity impact (I):** Measures the impact on integrity of a successful exploited vulnerability. Increased integrity impact increases the vulnerability score.
   IntegImpact = None: 0
   Partial: 0.275
   Complete: 0.660
3. **Availability impact (A):** Measures the impact on availability of a successful exploited vulnerability. Increased availability impact increases the vulnerability score.
   AvailImpact = None: 0
   Partial: 0.275
   Complete: 0.660

- **Temporal score metrics:** Measure the attributes that change over time but not in different environments. Consisting of:

TemporalScore = round_to_1_decimal (BaseScore * Exploitability * RemediationLevel * ReportConfidence)

1. **Exploitability (E):** Describes the current status of exploitation techniques or automated exploitation code.
   Exploitability = Unproven (U): 0.85
   Proof-of-concept (P): 0.9
   Functional (F): 0.95
   High (H): 1.0
   Not Defined (ND): 1.0
2. **Remediation Level (RL):** Allows the temporal score of a vulnerability to decrease as mitigations and official fixes are made available.
   RemediationLevel = Official Fix (O): 0.87
   Temporary Fix (T): 0.90
   Workaround (W): 0.95
   Unavailable (U): 1.0
   Not defined (ND): 1.0
3. **Report Confidence (RC):** Measures the level of confidence in the existence of the vulnerability and also the credibility of the technical details of the vulnerability.
   ReportConfidence= Unconfirmed (UC): 0.9
   Uncorroborated (UR): 0.95
   Confirmed (C): 1.0
   Not defined (ND): 1.0

- **Environmental score metrics:** Measure the characteristics that are proper to an specific user environment. Consisting of:

AdjustedImpact = min(10,10.41*(1-(1-ConfImpact*ConfReq)*(1-IntegImpact*IntegReq)*(1-AvailImpact * AvailReq)))

AdjustedTemporal = TemporalScore recomputed with the BaseScores Impact sub-equation replaced with the AdjustedImpact equation

EnvironmentalScore = round_to_1_decimal((AdjustedTemporal+(10-AdjustedTemporal)* CollateralDamagePotential)* TargetDistribution)

General Modifiers:

1. **Collateral Damage Potential (CDP):** Measures the potential loss or impact on either physical assets or financial impact if the vulnerability is exploited.
   CollateralDamagePotential = None (N): 0
   Low (L): 0.1
   Low-Medium (LM): 0.3
   Medium-High (MH): 0.4
   High (H): 0.5
   Not Defined (ND): 0
2. **Target Distribution (TD):** Measures the proportion of vulnerable systems in the environment.
   TargetDistribution = None (N): 0
   Low (L): 0.25
   Medium (M): 0.75
   High (H): 1.0
   Not Defined (ND): 1.0

Impact Subscore Modifiers:

1. **Confidentiality Requirement (CR), Integrity Requirement (IR) and Availability Requirement (AR):** Measures the security requirements allowing the environmental score to be fine-tuned according to the users' environment.
   ConfReq / IntegReq / AvailReq = Low (L): 0.5
   Medium (M): 1.0
   High (H): 1.51
   Not Defined (ND): 1.0

Metric 1

Definition of the metrics you can design from the dataset

The first relation we want to research is the relation between the CVSS score and the time before a patch was available. Gut feeling tells us that when it takes long before a patch being available, would make a vulnerability more dangerous. Therefor it would be logically that this can be seen in the CVSS score. So the hypothesis here is: How longer the time to patch how higher the CVSS score.

This hypothesis is tested for the vulnerabilities 'sql-injection' and 'XSS'. Especially these two are chosen, because the OSVD has the most data of both vulnerabilities.

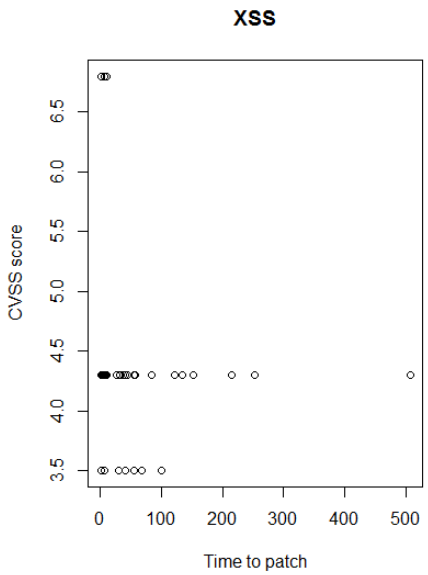


Figure 1: XSS

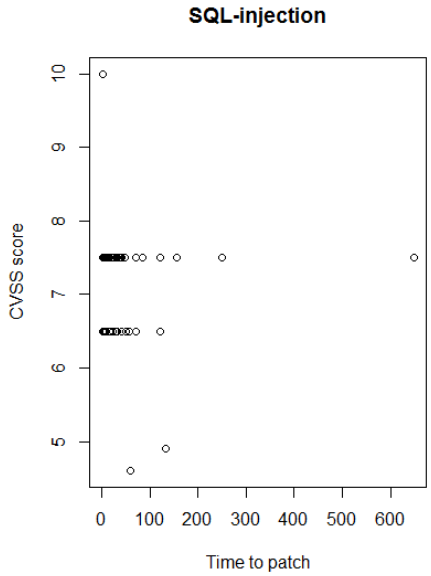


Figure 2: SQL Injection

	Correlation
XSS and CVSS score	-0,123
SQL injection and CVSS score	-0,036

Table 2: Correlation scores

Figure 1 and 2 as show that it looks like there is no relation between time to patch and the CVSS score; both are independent of each other. Table 2 confirmed this with the low values of correlation. Concluded can be that there is no correlation between the Time to patch and the CVSS score of a vulnerability.

Conclusion

As already mentioned in the paragraph about metrics in practice, the CVSS score is based on different variables and time to patch is not one of them. Therefor a correlation cannot be seen in figure 1 and 2. However here lies a recommendation for improvement of the CVSS score. It would be logical to implement also a variable about the time to patch in the CVSS score. Due to the fact that a CVSS score tells something about the threat level of the vulnerability.

Metric 2

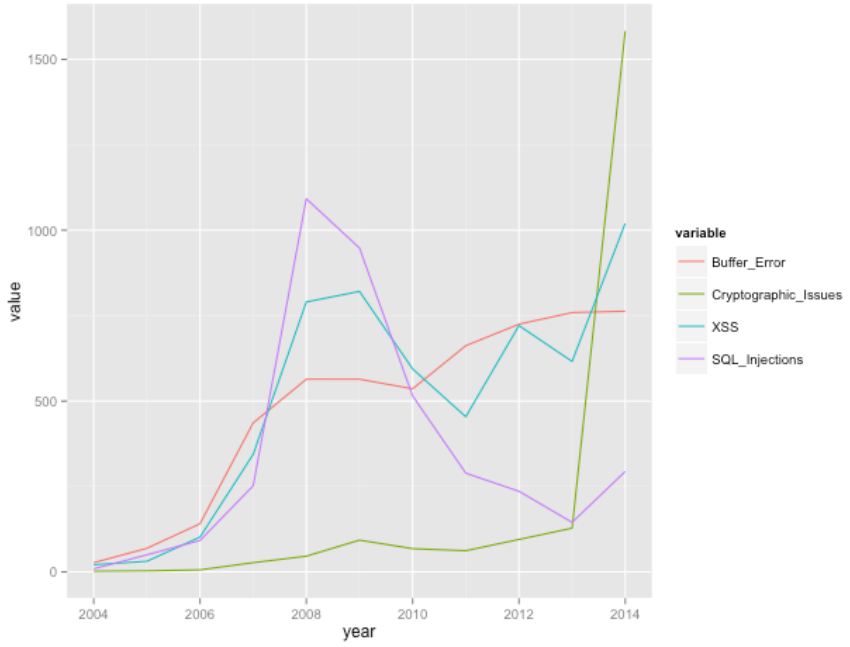
Definition of the metrics you can design from the dataset

Designing a metric from the dataset provided by the NVD is challenging. The dataset provides data on many types of vulnerabilities, but it does not provide much data that could be used for further analysis. If we look at the statistics page we can select different categories of vulnerabilities based on their CWE number and enables us to view the frequency of reported vulnerabilities over time and their CVSS. Using this data alone to create reliable and valuable metrics is difficult. The previous examples of possible metrics use a second data source. This metric on the other hand is solely based on the dataset provides by the NVD and thus limits by the limit amount of information it contains.

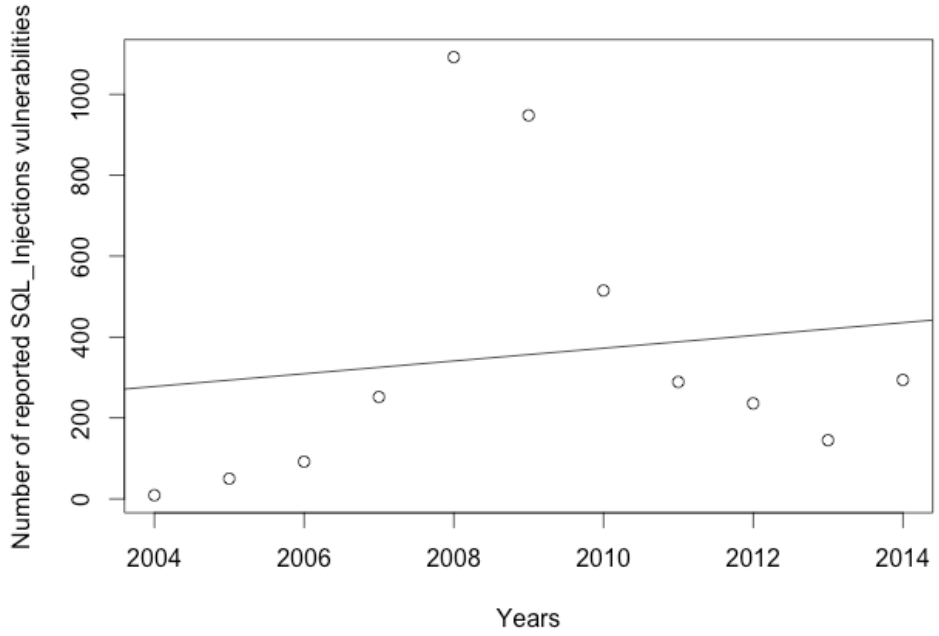
Cybersecurity is constant race against hackers who try to find new vulnerabilities they can exploit. This makes it constant race to patch vulnerable parts of the system. Resource are limited and this requires organizations to focus to security efforts. For it would useful to know what kind of vulnerabilities or becoming more attractive to hackers, and which category of vulnerabilities are becoming less attractive? This is of course hard to establish, but looking at the trends of different categories over the past years might provide additional insights that could be used in order to focus security efforts.

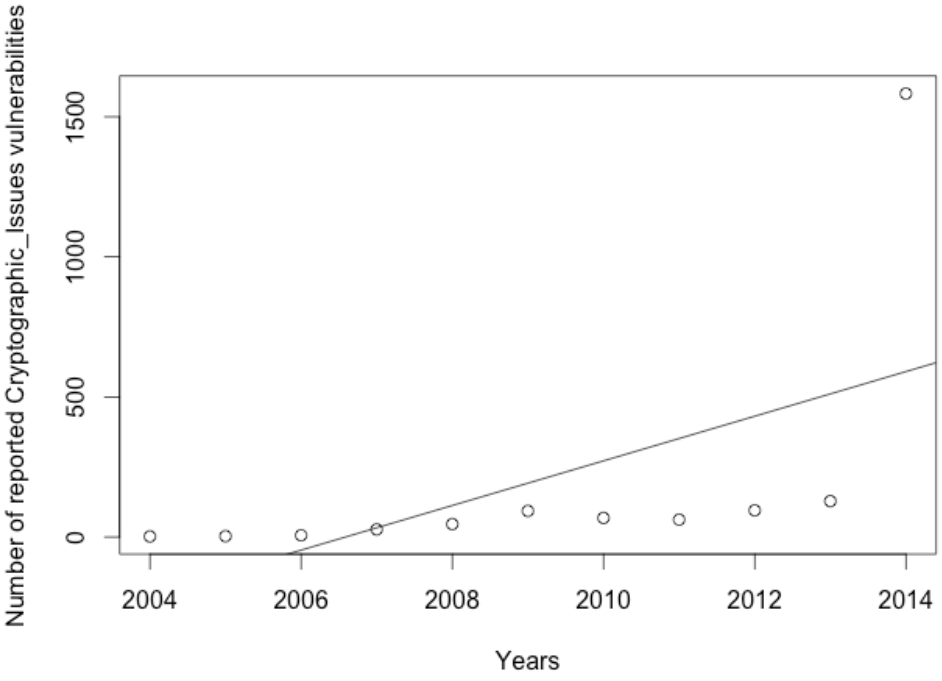
The metric proposed by this section focusses on the trends within reported vulnerabilities and tries to use this data to provide insight in possible future behavior of particular vulnerabilities. More strictly speaking, linear regression is used to predict the number of vulnerabilities for the next year. We are well aware that there a limitation for using regression as means to predict future behavior a we do not claim to able to use this as extremely reliable predictor, but more as an indicator possible future behavior.

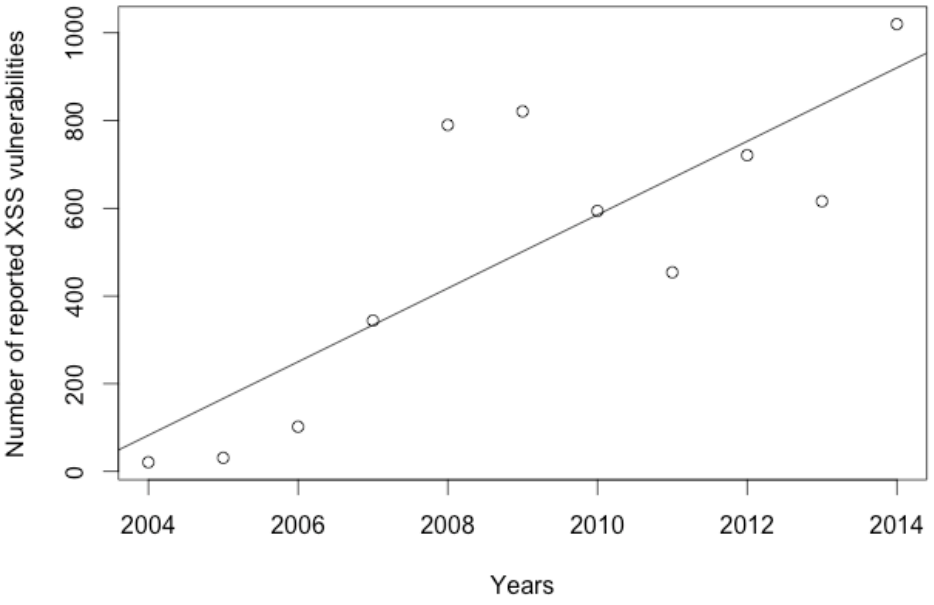
For this we looked at four types of vulnerabilities that are contained within the dataset of the NVD. These include buffers errors, cryptographic vulnerabilities, cross-site scripting, and SQL injections. The following graph provides an overview of the frequencies per category per year.

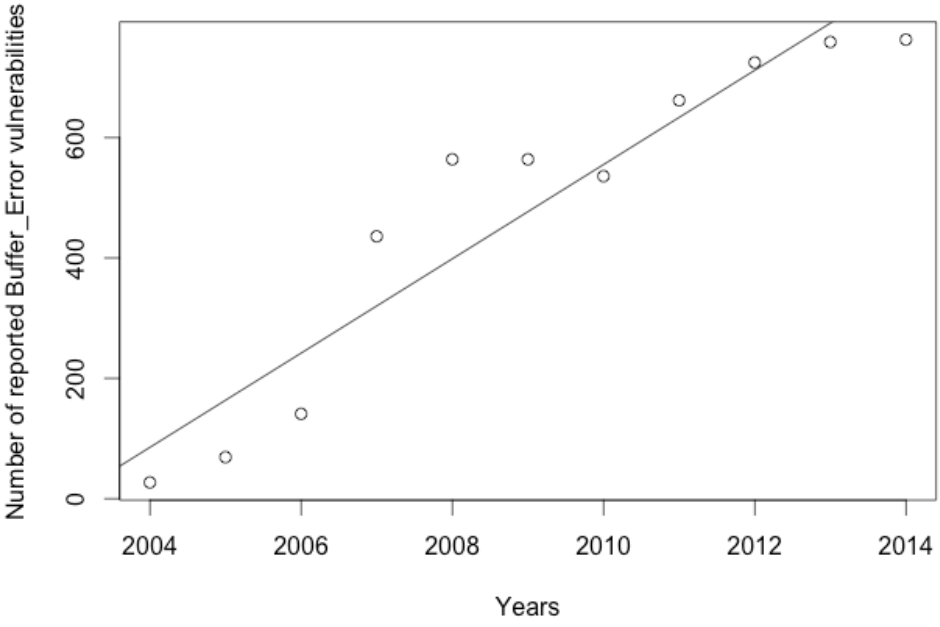


The graph shows the dynamic behavior of the frequencies over the past years. Judging for this fluctuating behavior one could argue that a linear model would not be the most suitable model to fit this behavior, but for prediction a linear model has proven to be more reliable due to fact that linear models tend to have a lower bias and higher variance, even when the data shows rather dynamic behavior. The following graphs show the frequencies per category over the period 2004-2014, with their corresponding fitted linear model.









The graphs show that in some cases the linear model is better capable of mapping the previous behavior than others. The following table shows the model coefficients and their corresponding prediction for 2015.

Category	Coefficient	Intercept	Prediction 2015
Buffer Errors	78.3	-156882.6	946.9
Cryptographic Issues	79.7	-159870.4	670.1
XSS	83.8	-167816.4	1003.9
SQL Injections	15.8	-31312.6	451.1

Conclusion

The coefficients could be used as a metric to determine the likelihood of increase for the next year. In this case more attention should be paid on XSS vulnerabilities while SQL injections should disserve less attention.

References

Allodi, L., & Massacci, F. (2013). My Software has a Vulnerability, should I worry? *arXiv preprint arXiv:1301.1275*.

Boyer, W., & McQueen, M. (2007). *Ideal Based Cyber Security Technical Metrics for Control Systems*.

Chen, Y., Wang, D., Fu, L. (2015). A detection model for SQL injection attack. In *International Journal of Collaborative Intelligence*, Vol. 1, No 2, 137

Mell, P., Scarfone, K., Romanosky, S. (2007). *A complete guide to the common vulnerability scoring system version 2.0*

Lazar, D., Chen, H., Wang, X., Zeldovich, N. (2014). Why does cryptographic software fail? A case study and open problems. In *Proceedings of 5th Asia Pacific Workshop on Systems.*

MITRE Corporation (2013). Terminology. In *cve.mitre.org*. Retrieved September 19, 2015 from http://cve.mitre.org/about/terminology.html

OWASP (2013). Top 10 2012-Top 10 in *owasp.org.* Retrieved September 15, 2015 from https://www.owasp.org/index.php/Top_10_2013-Top_10

Patriciu, V-V., Priescu, I., & Sebastian, N. (2006). Security Metrics for Enterprise Information Systems. In *Journal of Applied Quantitative Methods.*

Symantec (April 2015). *Internet Security Threat Report.* USA: Symantec Corporation World Headquarters

Tashi, I., & Ghernaouti-Helie, S. (2007). Security metric to improve information security Management. In *6th Annual Security Conference*. Las Vegas.

The Center for Internet Security (2010). *The CIS Security Metrics.*

Wang, X., Wen, Q., Zhang, Z. (2014). Buffer Overflow Vulnerability Detection based on Format-Matching on Source Level. In *International Conference on Logistics Engineering, Management and Computer Science (LEMCS 2014)*, 298 – 301.

Wasserman G., Zhendong S. (2008). Static Detection of Cross-Site Scripting Vulnerabilities. In *ACM/IEEE 30the International Conference on Software Engineering*. 171-180, Leipzig

Appendix

R-code Metric 1

```
> library(xlsx)
Loading required package: rJava
Loading required package: xlsxjars
> sqlinjection<-read.xlsx("C:/Users/Kirsten/Dropbox/EconomicsofCyberSecuri
ty/SQLInjection_OnePage.xlsx", 1)
> attach(sqlinjection)
> names(sqlinjection)
[1] "CVE.ID"                  "CVSSv2Score"
[3] "Time.to.patch"           "Time.to.exploid"
[5] "Time.to.vendor.response" "Days.of.Exposure"
> class(Time.to.patch)
[1] "numeric"
> class(CVSSv2Score)
[1] "numeric"
> plot(Time.to.patch,CVSSv2Score, main="SQL-injection", xlab="Time to patc
h", ylab="CVSS score")
> cor(Time.to.patch,CVSSv2Score)
[1] -0.03611324
[1] -0.03611324

> xss<-read.xlsx("C:/Users/Kirsten/Dropbox/EconomicsofCyberSecurity/XSS-da
ta_withoutNA.xlsx",1)
> attach(xss)
> names(xss)
[1] "CVEid"                "TimetoPatch"
[3] "TimetoExploit"        "TimetoVendorResponse"
[5] "CVSSv2Score"
> class(TimetoPatch)
[1] "numeric"
> class(CVSSv2Score)
[1] "numeric"
> plot(TimetoPatch,CVSSv2Score,main="XSS", xlab="Time to patch", ylab="CVS
S score")
> cor(TimetoPatch,CVSSv2Score)
[1] -0.1234796
```

R-code Metric 2

```
library("ggplot2",
lib.loc="/Library/Frameworks/R.framework/Versions/3.2/Resources/libr
ary")

library("reshape2",
lib.loc="/Library/Frameworks/R.framework/Versions/3.2/Resources/libr
ary")



Buffer_Error <- c(27, 69, 141, 436, 564, 564, 536, 662, 725,759,
763)

Cryptographic_Issues <- c(2, 3, 6, 27, 46, 93, 68, 62, 95, 128,
1583)

XSS <- c(21, 31, 102, 344, 790, 821, 594 ,454, 721, 616, 1020)

SQL_Injections <- c(9, 50, 92, 252, 1092, 948, 515, 289, 236, 145,
294)
```

```
year <- c(2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012,
2013, 2014)


vul_freq <- data.frame(
  Buffer_Error = c(Buffer_Error),
  Cryptographic_Issues = c(Cryptographic_Issues),
  XSS = c(XSS),
  SQL_Injections = c(SQL_Injections),
  year = c(year)
)


vul_freq_long <- melt(vul_freq, id="year")


ggplot(data=vul_freq_long,
       aes(x=year, y=value, colour=variable)) + geom_line()


############


#Calculate linear regression XSS


XSS <- c(21, 31, 102, 344, 790, 821, 594 ,454, 721, 616, 1020)
year <- c(2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012,
2013, 2014)


plot(year, XSS, ylab="Number of reported XSS vulnerabilities")


lm(XSS ~ year)
linearmodXSS <- lm(XSS ~ year)
summary(linearmodXSS)
plot(year, XSS, xlab = "Years", ylab = "Number of reported XSS
vulnerabilities")
abline(lm(XSS ~ year))


#Expected value XSS based on linear regression


expectedXSS = linearmodXSS[[1]][2]*2015+linearmodXSS[[1]][1]
```

```
expectedXSS


##############


#Calculate linear regression Buffer_Error


Buffer_Error <- c(27, 69, 141, 436, 564, 564, 536, 662, 725,759,
763)

year <- c(2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012,
2013, 2014)


plot(year, Buffer_Error, ylab="Number of reported Buffer_Error
vulnerabilities")


lm(Buffer_Error ~ year)

linearmodBuffer_Error <- lm(Buffer_Error ~ year)

summary(linearmodBuffer_Error)

plot(year, Buffer_Error, xlab = "Years", ylab = "Number of reported
Buffer_Error vulnerabilities")

abline(lm(Buffer_Error ~ year))


#Expected value Buffer_Error based on linear regression


expectedBuffer_Error =
linearmodBuffer_Error[[1]][2]*2015+linearmodBuffer_Error[[1]][1]

expectedBuffer_Error


##############


#Calculate linear regression Cryptographic_Issues


Cryptographic_Issues <- c(2, 3, 6, 27, 46, 93, 68, 62, 95, 128,
1583)

year <- c(2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012,
2013, 2014)


plot(year, Cryptographic_Issues, ylab="Number of reported
Cryptographic_Issues vulnerabilities")
```

```
lm(Cryptographic_Issues ~ year)

linearmodCryptographic_Issues <- lm(Cryptographic_Issues ~ year)

summary(linearmodCryptographic_Issues)

plot(year, Cryptographic_Issues, xlab = "Years", ylab = "Number of
reported Cryptographic_Issues vulnerabilities")

abline(lm(Cryptographic_Issues ~ year))


#Expected value Cryptographic_Issues based on linear regression


expectedCryptographic_Issues =
linearmodCryptographic_Issues[[1]][2]*2015+linearmodCryptographic_Is
sues[[1]][1]

expectedCryptographic_Issues


##############


#Calculate linear regression SQL_Injections


SQL_Injections <- c(9, 50, 92, 252, 1092, 948, 515, 289, 236, 145,
294)

year <- c(2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012,
2013, 2014)


plot(year, SQL_Injections, ylab="Number of reported SQL_Injections
vulnerabilities")


lm(SQL_Injections ~ year)

linearmodSQL_Injections <- lm(SQL_Injections ~ year)

summary(linearmodSQL_Injections)

plot(year, SQL_Injections, xlab = "Years", ylab = "Number of
reported SQL_Injections vulnerabilities")

abline(lm(SQL_Injections ~ year))


#Expected value SQL_Injections based on linear regression


expectedSQL_Injections =
linearmodSQL_Injections[[1]][2]*2015+linearmodSQL_Injections[[1]][1]
```

expectedSQL_Injections