Expert assessment on the probability of successful remote code execution attacks

Hannes Holm[1], Teodor Sommestad[1], Ulrik Franke[1], Mathias Ekstedt[1]

[1]Department of Industrial information and control systems, the Royal Institute of Technology, 100 44 Stockholm, Sweden
{hannesh, teodors, ulrikf, mathiase}@ics.kth.se

Abstract. This paper describes a study on how cyber security experts assess the importance of three variables related to the probability of successful remote code execution attacks – presence of: (i) non-executable memory, (ii) access and (iii) exploits for High or Medium vulnerabilities as defined by the Common Vulnerability Scoring System. The rest of the relevant variables were fixed by the environment of a cyber defense exercise where the respondents participated. The questionnaire was fully completed by fifteen experts. These experts perceived access as the most important variable and availability of exploits for High vulnerabilities as more important than Medium vulnerabilities. Non-executable memory was not seen as significant, however, presumably due to lack of address space layout randomization and canaries in the network architecture of the cyber defense exercise scenario.

Keywords. Cyber security, Remote code execution, Software vulnerabilities

1. Introduction

Exploits (i.e. cyber attacks) which can provide administrator privileges of systems are attractive to attackers, particularly if they can be executed remotely. Many types of exploits can be used to achieve this. One example is to take advantage of vulnerabilities in the memory management of a system in order to execute the attacker's own programming instructions.

Many conditions influence whether an attacker can manage to successfully execute such attacks. A few examples include vulnerabilities present in the targeted machine, the competence of the attacker and any protective measures which are in place. Clearly, it is valuable to know under which circumstances such attacks are likely to succeed and under which circumstances they are not. Data regarding this would be useful when prioritizing protective measures, e.g. remediation options, or as input to complex security analysis frameworks, e.g. [1] and [2].

However, such data are costly to collect experimentally, as it would require the involvement of many security experts in a substantial number of scenarios. As a consequence, there are currently very little empirical data available regarding this topic. A less costly method to collect this data is to ask security experts to make judgments based on their experience. This paper presents judgments made by 15 security experts in conjunction with an international cyber defense exercise. The experts estimated the probability of successful remote exploitation of software vulnerabilities given different scenarios.

The rest of the paper unfolds as follows: Sect. 2 describes related work. Sect. 3 explains the variables studied in this paper. Sect. 4 describes the method. Sect. 5 presents the result and our analysis. Sect. 6 discusses this result and Sect. 7 concludes the paper.

2. Related work

This paper focuses on vulnerabilities that enable attackers to execute arbitrary code on a targeted machine from a remote location. A common class of vulnerabilities that accomplish this are buffer overflows [3][4].

As buffer overflow vulnerabilities are severe security problems, a number of techniques and tools have been developed to eliminate them or make exploitation of them more difficult. Common countermeasures include authorization, authentication control, vulnerability management (removing vulnerabilities due to software flaws), and measures focusing explicitly on hardening computers or source code against arbitrary code attacks. Younan [5] divides such measures into ten different types. Seven of these focus on the software product itself and to lower its vulnerability. These seven are: safe languages (e.g. Cyclone [6]), bound checkers (e.g. Cash [7]), hardened libraries (e.g. FormatGuard [12]), separation and replication of information countermeasures (e.g. Libverify [10]), runtime taint trackers (e.g. TaintCheck [13]), dynamic analysis and testing (e.g. Purify [14]), and static analysis like (e.g. [15]). These measures will help removing vulnerabilities in the code or decrease their severity. However, they are not designed to influence the difficulty of exploiting the high-severity vulnerabilities that may remain in the code. As this study focuses on the latter, the following three types of measures are of particular relevance: probabilistic countermeasures (e.g. PaX ASLR [8], instruction set randomization [9] and StackGuard [3]), paging-based countermeasures (e.g. PaX [8]), and execution monitors (e.g. [11]).

Excellent overviews of the different measures possible to implement can be found in [5,16-18]. These overviews contain discussions on the effectiveness of different measures. Also, the descriptions of the techniques and tools come with a qualitative or theoretic evaluation of their effectiveness.

The number of empirical evaluations, however, remains small. The study performed by [19] is an exception. This study focuses on measures that remove or degrade the software product's vulnerabilities and tests seven different techniques for thwarting buffer overflow attacks. It accurately describes the exploits these measures work against. However, it does not capture the competence of attackers or show the exploits they apply in practice. Therefore, it cannot be used to infer probabilities of success in general. This also applies to studies focusing on the effectiveness of specific techniques, for example [20]. In [20] the effectiveness of address space layout randomization is tested under different conditions and weaknesses that are exploitable under specific condition, but how often these conditions apply in practice is not discussed (or known to the community at large).

3. Studied variables

As can be seen in Sect. 2, there are many variables which influence the possibility of successfully perform arbitrary code execution attacks. However, most defense mechanisms are not widely used in practice. This paper aims to assess the significance of one of the most commonly used defense mechanisms, namely non-executable memory. Non-executable memory is an easily implemented countermeasure which is available for most operating systems (e.g. all Windows operating systems since XP). Furthermore, the study assessed the importance of non-executable memory under the presence of two very common conditions: i) accessibility and ii) the severity of the exploited vulnerability. These variables are detailed below.

Non-executable memory (NX). This is a paging-based countermeasure [5] that marks the data segment of a computer memory as non-executable. This is done to make it more difficult for an attacker to execute code that has been injected into the data segment. NX is a feature that is available as an option for most operating systems and has implementations without costs in computational power or memory usage [5].

Access. In all scenarios considered in this study it is assumed that the attacker can connect to the vulnerable service. The access variable involves whether the attacker can access the service and its resources as a user of it. For example, if the service is the SMB service on a machine and access has state *true*, it would mean that the attacker is a part of the Windows domain. If it has the state *false* the attacker is not in the domain but can still send requests to the service as an external entity.

Severity of the exploited vulnerability. In all scenarios considered there is an *exploit* available to the attacker corresponding to a *vulnerability* on the targeted machine. In this investigation, exploits for High and Medium severity vulnerabilities as defined by the Common Vulnerability Scoring System (CVSS) [21] are distinguished. The CVSS is a set of metrics used to quantitatively compare and describe different vulnerabilities regarding various attributes such as ease of exploitation and the consequences of successful exploitation. The characteristics of a *High* vulnerability were informally translated to "There is full impact on confidentiality, integrity and availability and no authentication required". The characteristics of a *Medium* vulnerability were informally translated to "There is partial impact on confidentiality, integrity and availability and authentication required".

NX, access and the severity of the vulnerability exploited are all believed to affect the probability that **successful arbitrary code execution** is in the state *yes***,** i.e. how probable it is to succeed with executing code remotely on a machine with the chosen defense mechanisms.

An overview of the chosen variables and their possible states is shown in Fig. 1.

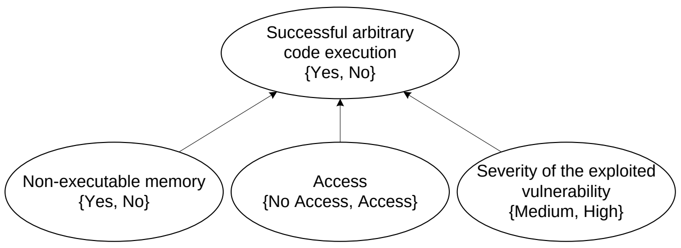
Successful arbitrary code execution {Yes, No}
Non-executable memory {Yes, No}
Access {No Access, Access}
Severity of the exploited vulnerability {Medium, High}

Fig. 1. Studied variables and their possible states.

Naturally, there are numerous other variables that influence the probability of succeeding with remote code execution attacks. Some examples of other protective measures were provided in Sect. 2. In this study, only the security measures depicted in Fig. 1 were evaluated. However, the states of all other relevant measures were fixed by the environment of a cyber defense exercise. Sect. 4 describes this in more detail.

4. Method

4.1 The respondents and the cyber defense exercise

The preferred population for assessing this type of data is IT security practitioners (such as pen-testers) or security researchers. Respondents of this type should be able to give assessments regarding how important the studied variables are as they have practical experience from performing the cyber attack in question under different conditions.

This study describes a survey which was handed out before the start of a cyber defense exercise named Baltic Cyber Shield (BCS). The exercise was managed at the Cooperative Cyber Defense Centre of Excellence in Tallinn, Estonia. Its virtual battlefield was designed and hosted by the Swedish Defence Research Agency with the support of the Swedish National Defence College. The environment was set to mimic a typical critical information infrastructure with elements of supervisory control and data acquisition (SCADA), for instance programmable logic controllers. The exercise included a sixteen person strong red team (i.e. attackers), six blue teams (i.e. defenders) of 6-10 people per team, a white team (i.e. game management), a green team (i.e. technical infrastructure management) and one white observer per team. The majority of the exercise's participants were computer security specialists and some were computer security researchers. They were affiliated with various northern European governments, military, private sector and academic institutions. The interested reader is referred to [22,23] for more thorough descriptions of the exercise.

The survey was distributed to all blue team and red team members. A sample of fifteen respondents from the BCS fully completed the survey and thus provides the empirical data for this study. Members from all teams except one blue team and the red team answered the survey. Respondent ages ranged from 26 to 53 years, with a mean of 33 years. The respondents were asked to grade their general expertise in the IT-security domain from 1 = no expertise to 5 = very high expertise. The respondents' self-assessed expertise ranged from no expertise (1 respondent) to very high expertise (3 respondents), with a mean of 3.33.

4.2 The survey

The respondents were asked to specify the probability of carrying out successful arbitrary code execution attacks given different combinations of the three examined variables. The probability that *successful arbitrary code execution* is in the state *yes* was assessed through a discrete scale of 0-100 percent. Hence, the answer 0 percent means that there is no possibility of a successful attack and the answer 100 percent means that the attack is certain to succeed. Due to the complexity of the question format each question was, as recommended by [24], accompanied by a figure describing the scenario. Respondents were also required to state the confidence of their answers on a scale from 1 (very low confidence) to 5 (very high confidence).

The survey was composed of five main parts: (i) introductory letter, (ii) respondent background, (iii) respondent training, (iv) probability assessments, and (v) importance of other variables. In the final part of the survey, (v), the respondents were asked whether there were any additional important factors, not in the study. Survey training is something of particular importance when trying to assess information for complex questions, to ensure that respondents have correctly interpreted the topic of interest [24]. This survey trained the respondents on several concepts: the question format and the scales used, the overall scenario and the CVSS.

4.3 Implicitly defined variables in the scenario

As stated in Sect. 3, there are numerous variables that are of importance to the probability of successful remote code execution attacks. The survey only explicitly defined the state of three variables. However, the questions were formulated in a way that made all variables of relevance implicitly defined. The exact formulation was *"For the coming questions, assume that you are to carry out an arbitrary code execution attack against a service running on a machine in one of the blue team's networks. Also assume that you can connect the machine and the service in question, i.e. you have access to the service's LAN."*

As all respondents had been given previous access to the network architecture, had equal knowledge of the scenario and therefore should have pictured the same general network and defense mechanisms, this overall description effectively fixed the states of all variables of importance.

5. Results and analysis

An overview of the survey responses regarding the probability of successful remote code execution is given in Table 1. As can be seen, access seems to be the most influential variable, while presence of non-executable memory (NX) seems to be of minor importance. The normal distribution of the eight studied scenarios was determined using QQ-plots. QQ-plots are used to assess the distribution of data sets, e.g. to make sure that statistical methods used are applicable to the distributions at hand [25].

Table 1. Studied scenarios and their results. LCI: 95% Lower Confidence Interval. UCI: 95% Upper Confidence Interval.

Scenario	Access	NX	Exploit	Mean	Stdev	LCI	UCI	Samples
1	Yes	Yes	High	85.6	8.3	81.4	89.8	15
2	Yes	Yes	Medium	74.6	11.5	68.6	80.7	15
3	Yes	No	High	81.2	14.5	73.9	88.6	15
4	Yes	No	Medium	65.7	14.3	58.5	72.3	15
5	No	Yes	High	54.7	25.4	41.8	67.5	15
6	No	Yes	Medium	42.9	21.7	31.9	53.9	15
7	No	No	High	52.3	30.2	37.0	67.6	15
8	No	No	Medium	43.7	28.5	29.2	58.1	15

As all possible combinations (2^3) between the studied variables (non-executable memory (NX), access and available exploits) were evaluated it was possible to analyze the results using traditional experimental design techniques [26]. Experimental design is about extraction of a maximum amount of unbiased information regarding the factors affecting a process from as few observations as possible. This is done though a set of statistical tools such as analysis of variance (ANOVA), regression analysis and QQ-plots.

Analysis of variance, ANOVA, is a collection of tools used for statistical tests to assess the significance of relations between variables. The null hypothesis of an ANOVA, H0, is true if the differences between observed groups of data can be described by chance and false if there are systematic differences large enough to justify rejection of H0. The boundary associated with rejecting a null hypothesis is generally described using a probability, *p*. Datasets containing groups of observed data with differences large enough to reject the null hypothesis are statistically significant. A commonly used probabilistic boundary is $p < 0.05$, which implies that there is less than 5% probability for H0 to be true [25].

A regression analysis provides an equation that models the expected outcome on a quantitative variable Y from data on one or more variables X_1, X_2… X_n [25]. A key concept in regression analysis is to determine how well the identified equation actually models the variation in a dataset. The measure of the spread of points around the regression line can be presented using the coefficient of determination, R^2, where $0 < R^2 \leq 1$ [25]. In other words, R^2 measures how well the regression model explains the variation in the dataset. An R^2 of 1 (100%) means the model explains all variation. The adjusted R^2 is a version of R^2 which also compensates for the number of degrees of freedom [26], and is therefore generally to be preferred over the traditional R^2.

A complete 2^3 designed experiment was carried out with the survey (all possible combinations of the three examined variables; 8 scenarios), treating the respondents' answers as the experiment's outcome. The experimental design was analyzed through QQ-plots, ANOVA, regression analysis and evaluation of model assumptions through an analysis of residuals. No problems regarding residual lack of fit were found, and the model assumptions should thus not be rejected.

Analysis of the experiment's outcome (cf. Table 2) shows that both access ($p < 0.0001$) and the severity of the exploited vulnerability ($p = 0.055$) are of importance for successful remote code executions value. However, non-executable memory is not seen as important by the respondents ($p = 0.32$). Furthermore, there are no seemingly important relations between any of the studied variables (NX, access and exploit), suggesting that the influence of these variables on successful remote code execution are in fact independent from one another. Equation 1 describes how the most important variables, Access (A) and the Severity of the exploited vulnerability (E), relate to the probability of successful remote code execution (P).

$$P = 62.59 + 14.21 * A + 3.69 * E \qquad (1)$$

The value of A is {No access, Access} = {-1, 1} and the value of E is {Medium, High} = {-1, 1}. The chosen regression model (cf. Equation 1) has an adjusted R^2 of 0.32 which can be considered decent but not great. The regression model suggests that it will be quite likely to succeed with a remote code execution, even if there only is an exploit for a Medium vulnerability available and the attacker does not have access (44.7% probability), and highly probable if there is an exploit for a High vulnerability available and the attacker has access (80.5% probability).

Table 2. Results from the designed experiment.

Source	p-value
NX	0.32
Access	< 0.0001
Exploit	0.055
NX-Access	0.44
NX-Exploit	0.32
Access-Exploit	0.13
NX-Access-Exploit	0.11

Table 3 describes how certain the experts were for the different scenarios. All confidence intervals largely overlap and there is thus no reason to believe that the importance of any scenario or variable is more difficult to assess than another. The experts are overall fairly certain: no confidence interval goes below 2.5 (out of 5).

Table 3. How certain the experts were for the different scenarios. LCI: 95% Lower Confidence Interval. UCI: 95% Upper Confidence Interval.

Scenario	NX	Access	Exploit	Mean	Stdev	LCI	UCI	Samples
1	Yes	Yes	High	3.5	1.5	2.8	4.3	15
2	Yes	Yes	Medium	3.4	1.3	2.7	4.1	15
3	No	Yes	High	3.6	1.2	3.0	4.2	15
4	No	Yes	Medium	3.2	1.1	2.7	3.8	15
5	Yes	No	High	3.1	1.2	2.5	3.7	15
6	Yes	No	Medium	3.1	1.1	2.5	3.6	15

7	No	No	High	3.3	1.0	2.8	3.9	15
8	No	No	Medium	3.3	1.3	2.6	4.0	15

6. Discussion

6.1 Variables and their importance

Both access and the severity of the exploited vulnerability are seen as important by the experts. However, non-executable memory is not seen as relevant. This can be due to respondents being able to tune the exploits to counter this defense mechanism. That is, a stack-smashing exploit can quite easily be tuned into an arc-injection (i.e. return-into-libc) [27] exploit, and if this is the case any non-executable memory defense is rendered next to useless. However, it is frequently pointed out that non-executable memory is a potent defense if combined with other countermeasures, such as address space layout randomization [8,20] or canaries such as Stackguard [3]. Thus, the reason why non-executable memory did not turn out as significant during this study could be that the states of the numerous other important variables in place during the cyber defense exercise were fixed in such a way that enabling non-executable memory would not make any difference.

The respondents were also asked to detail any missing variables they perceived as important to the probability of successful remote code execution, both in the survey (cf. Table 4), and through informal discussions after the cyber defense exercise. It is notable that no respondents listed the same missing variables, suggesting a high validity. However, the discussions made it clear that several variables of great practical importance should be evaluated further, for example address spaced layout randomization.

Table 4. Missing variables.

Variable	Frequency
System monitoring	1
System updates	1
Application sandboxing	1
Unknown vulnerability	1
Service misconfiguration	1

6.2 Validity and reliability

While there were no formal quantitative ways of capturing the reliability of the survey, no respondent had any major issue answering any survey question. However, one thing that came up during the informal discussions was that respondents requested the possibility to answer through probability distributions instead of point estimates. Also, the survey was based on a network which the respondents had practical experience from. Numerous variable states were therefore not specified in the survey but were instead known through the respondents' familiarity with the scenario. While this might increase the reliability of the study (by giving all respondents a common understanding of the scenario) it has negative impact on the external validity. Several important variables were static and it is therefore difficult to generalize the results to a more general case, e.g. another enterprise environment. Thus, while the results might be valid and reliable for a cyber defense exercise of this type and the computer network used in it they are not likely to maintain validity for a more general case. We believe, however, that these results should give a clear hint about the relative importance of the studied variables, even for a more general case.

7. Conclusions and future work

Among the variables addressed in this study, experts perceive user access to the service that is to be exploited as the most important one for successful remote code execution. A readily available exploit for a High severity vulnerability as defined by the CVSS gives a greater success rate than a readily available exploit for a Medium severity vulnerability. The presence of non-executable memory is not perceived as important by the experts, presumably due to lack of other related countermeasures such as address space layout randomization.

This study shows that it is possible to obtain approximate estimates from security experts on issues that are difficult to evaluate through experiments. The experts tend to agree with each other, indicating that they are experts in the domain [28].

It would be interesting to study other variables suggested in the literature and by the experts. It would also be interesting to evaluate how close to the "truth" that the expert predictions really are. This is something which we aim to do in the future. Finally, we believe that there is a great need for more quantitative studies assessing the importance of different countermeasures, not only for arbitrary code execution attacks, but for any cyber attack.

8. References

1. J. Homer, K. Manhattan, X. Ou, and D. Schmidt, *A Sound and Practical Approach to Quantifying Security Risk in Enterprise Networks*, Kansas: 2010.
2. T. Sommestad, M. Ekstedt, and P. Johnson, "A Probabilistic Relational Model for Security Risk Analysis," *Computers & Security*, 2010.
3. C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole, "Buffer Overflows : Attacks and Defenses for the Vulnerability of the Decade," *Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems]*, 2003, pp. 227-237.
4. A. One, "Smashing the stack for fun and profit," *Phrack magazine*, vol. 7, 1996, p. 1996–11.
5. Y. Younan, "Efficient countermeasures for software vulnerabilities due to memory management errors," 2008.
6. T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang, "Cyclone: A safe dialect of C," *USENIX*, Monterrey, CA, USA: 2002, pp. 275-288.

7. L.-chung L.T.-cker Chiueh, “Checking Array Bound Violation Using Segmentation Hardware,” *2005 International Conference on Dependable Systems and Networks (DSN’05)*, 2005, pp. 388-397.
8. PaX Team, *PaX address space layout randomization (ASLR).*
9. G. Kc, A. Keromytis, and V. Prevelakis, “Countering code-injection attacks with instruction-set randomization,” *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, p. 280.
10. A. Baratloo and N. Singh, “Transparent run-time defense against stack smashing attacks,” *Proceedings of the annual conference on USENIX*, 2000.
11. S.H. Yong and S. Horwitz, “Protecting C programs from attacks via invalid pointer dereferences,” *ACM SIGSOFT Software Engineering Notes*, vol. 28, Sep. 2003, p. 307.
12. C. Cowan, M. Barringer, S. Beattie, G. Kroah-Hartman, M. Frantzen, and J. Lokier, “FormatGuard: Automatic protection from printf format string vulnerabilities,” *Proceedings of the 10th conference on USENIX Security Symposium-Volume 10*, USENIX Association, 2001, p. 15–15.
13. J. Newsome, “Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software,” *Network and Distributed System Security*, 2005.
14. R.H. and B. Joyce., “Purify: Fast detection of memory leaks and access errors,” *Winter USENIX Conferenc*, San Francisco, California, USA, January: 1992, p. 125--136.
15. N. Dor, “Cleanness checking of string manipulations in C programs via integer analysis,” *Static Analysis*, 2001.
16. N. Frykholm, “Countermeasures against buffer overflow attacks,” *RSA Tech Note*, 2000, pp. 1-9.
17. S.D. Xenitellis, “Identifying security vulnerabilities through input flow tracing and analysis,” *Information Management & Computer Security*, vol. 11, 2003, pp. 195-199.
18. U. Erlingsson, *Low-level Software Security : Attacks and Defenses Low-level Software Security : Attacks and Defenses*, Redmond, WA, USA: 2007.
19. J. Wilander and M. Kamkar, “A comparison of publicly available tools for dynamic buffer overflow prevention,” *Proceedings of the 10th Network and Distributed System Security Symposium*, Citeseer, 2003, p. 149–162.
20. H. Shacham, M. Page, B. Pfaff, and E. Goh, “On the effectiveness of address-space randomization,” *ACM conference on*, 2004, p. 298.
21. P. Mell, K. Scarfone, and S. Romanosky, “A Complete Guide to the Common Vulnerability Scoring System Version 2.0,” *System*, 2007, pp. 1-23.
22. K. Geers, “Live Fire Exercise: Preparing for Cyber War,” 2010.
23. H. Holm, T. Sommestad, J. Almroth, and M. Persson, “A quantitative evaluation of vulnerability scanning,” *Information Management & Computer Security*.
24. P.H. Garthwaite, J.B. Kadane, and A. O’Hagan, “Statistical methods for eliciting probability distributions,” *Journal of the American Statistical Association*, vol. 100, 2005, pp. 680-701.
25. R.M. Warner, *Applied statistics: From bivariate through multivariate techniques*, Sage Publications, Inc, 2008.
26. D.C. Montgomery, *Design and analysis of experiments*, John Wiley & Sons Inc, 2008.
27. J. Pincus and B. Baker, “Beyond stack smashing: Recent advances in exploiting buffer overruns,” *Security & Privacy, IEEE*, vol. 2, 2004, p. 20–27.
28. H. Einhorn, “Expert judgment: Some necessary conditions and an example,” *Journal of Applied Psychology*, 1974.