A Model for Predicting the Likelihood of Successful Exploitation

Hannes Holm
Swedish Defence Research Agency (FOI),
Olaus Magnus väg 42,
Linköping, Sweden
hannes.holm@foi.se

Ioana Rodhe
Swedish Defence Research Agency (FOI),
Olaus Magnus väg 42,
Linköping, Sweden
ioana.rodhe@foi.se

Abstract

This paper presents a model that estimates the likelihood that a detected vulnerability can be exploited. The data used to produce the model was obtained by carrying out an experiment that involved exploit attempts against 1179 different machines within a cyber range. Three machine learning algorithms were tested: support vector machines, random forests and neural networks. The best results were provided by a random forest model. This model has a mean cross-validation accuracy of 98.2% and an F1 score of 0.73.

1. Introduction

Cyber security vulnerabilities are common and problematic, and there are various tools available for detecting them. A commonly used type of tool is the automated network vulnerability scanner, for example, OpenVAS, Nessus and Nexpose. This type of tool analyzes responses to carefully chosen network queries thought to expose security vulnerabilities. The validity of each vulnerability reported by such a tool is, however, uncertain [1]. Many existing vulnerabilities are missed (false negatives) and non-existent vulnerabilities are erroneously reported (false positives). To many network administrators, false positives are arguably as bad as false negatives as vulnerability mitigation can be costly and resources are scarce. Various models have been created to aid decision makers regarding which vulnerability to mitigate first. The most renown vulnerability severity model is the Common Vulnerability Scoring System (CVSS). All vulnerabilities within the US National Vulnerability Database (NVD[1]) are scored with the CVSS, and most tools that concern known vulnerabilities reference it. Many researchers even convert the CVSS score into a probability and include it in their models (e.g., [2]).

Unfortunately, while the CVSS is widely used, it is largely unknown whether its estimates (or those of alternative models) are valid or reliable [3]. Experts believe that one of the four most important aspects that the CVSS fail to properly address concern exploitation reliability [3], i.e., the likelihood of successful vulnerability exploitation. Exploit reliability is an important factor as, from a threat agent's perspective, an exploit attempt generate activity that can alert the victim. Additionally, a failed exploit attempt sometimes causes the targeted software or system to crash. For these reasons, threat agents generally avoid unreliable exploits. Furthermore, if a vulnerability that can be reliably exploited is located in a server software, it is typically wormable [4] (i.e., exploitation is possible to completely automate). The cost of a zero-day network worm outbreak has been estimated to US$2.6 billion [5]. In other words, exploit reliability is an important predictor of both the occurrence and success rate of cyber attacks. Knowledge on the reliability of different exploits would thus be of great value to the community. Unfortunately, no such public knowledge is available in neither the industry nor academia. This is likely due to the lack of publicly available reliable empirical data of actual exploit attempts.

Obtaining reliable empirical data on exploit attempts is certainly not a simple matter, as exemplified by network intrusion detection research, where a dataset from 1999 [6], that was subjected to severe criticism within a year of its release [7], still is used by researchers to train and test their models (e.g., [8, 9]). Other researchers have been able to obtain data concerning real-world attacks on operational systems (e.g., [10, 11]). Models created based on such data are unfortunately also flawed as there is no ground-truth available concerning which attacks that have been attempted, how attacks were configured, or which attacks that were successful.

A method for obtaining data that negates the reliability issues concerning real-world attacks is to carry out controlled experiments in cyber ranges [12, 13, 14], where all events are known and technical events

[1]`https://nvd.nist.gov`

URI: https://hdl.handle.net/10125/64531
978-0-9981331-3-3

can be monitored in-depth. This paper describes a controlled experiment within a cyber range that was carried out to develop machine learning (ML) classifiers that can be used to estimate the likelihood that a detected vulnerability is exploitable.

The remainder of the paper is structured as follows: Section 2 presents related work. Section 3 presents the methodology of the study and section 4 its results. Finally, section 5 critically discusses these results and presents future work.

2. Related work

There are few published empirical efforts to determine exploit reliability. The closest related work is that of Holm and Sommested [14], who analyzed the success rates of 45 different server-side exploit modules that were tested across 1223 configurations. The study found that a mere eight exploit attempts were successful. The present paper extends and improves upon the methodology described in [14] by also testing remote client exploits, file exploits and local exploits. Furthermore, it analyzes exploit reliability through a set of carefully chosen features and ML models.

The second closest identified related work is that by Bozorgi et al. [15], who propose a method for predicting the mere occurrence of published exploits based on vulnerability information available within the CVE and Open Source Vulnerability Database (OSVDB[2]). The authors employed linear support vector machines (SVM) trained with approximately 14 000 samples and approximately 94 000 features, most being bag of word-vectors. Their best model had an error rate of 14%. They also found that all parts of a vulnerability report contained useful features. The present work differs from the work by Bozorgi et al. [15] in the sense that it analyzes vulnerability exploitability through execution of real attacks.

A marginally relevant work is that by Allodi and Massacci [16], who use a case-control study methodology to quantify the risk reduction achievable by acting on a risk factor, where the risk factor is computed by using the CVSS (v2) Base Score and other metrics such as existence of a proof-of-concept exploit or exploit presence in black markets. The analysis is based on four datasets: The NVD (vulnerability and CVSS data), Exploit-DB[3] (proof-of-concept exploits), EKITS (a database created by the authors that contain information regarding exploit kits) and SYM (Symantec's AttackSignature and ThreatExplorer datasets). The paper concludes that a patching policy based on Base Score provides a risk reduction of about 4% (which is equivalent to randomly picking vulnerabilities to fix). A patching policy based on the presence of a proof-of-concept exploits (Exploit-DB) can provide risk reductions up to 45%, and the presence of an exploit in an exploit kit (EKITS) can increase the risk reduction with up to 80%. Here, risk reduction concerns the portion of detected vulnerabilities that have entries in SYM that would be mitigated given a certain patching policy.

Another marginally relevant work is that by Holm et al. [17], who examine whether system-level vulnerability metrics based on the CVSS (v2) correlate with the actual time required to compromise systems. The dataset used for analysis was collected during an international cyber defence exercise with more than 100 participants and includes 34 actual system compromises. The results of the study show no strong correlation between the time to compromise and any of the studied metrics.

3. Methodology

This chapter describes the methodology for creating the ML classifiers that measure the likelihood that a detected vulnerability can be successfully exploited.

3.1. Cyber range and used tools

The Cyber Range And Training Environment (CRATE) is a cyber range built and maintained by the Swedish Defence Research Agency (FOI) that has been used for a number of research studies related to vulnerability assessments and situational awareness [13, 14, 18, 17]. Physically, CRATE consists of approximately 500 servers, a number of switches and various auxiliary equipment (e.g., for remote access). During an experiment these servers are instrumented with VirtualBox machines of various kinds connected through VxLAN and other communication technologies to form a "mini-internet" of sorts. Instrumentation is straightforward for any type of operating system and application software that VirtualBox can handle.

Scanning, Vulnerabilities, Exploits and Detection (SVED) is a tool developed by FOI for planning and executing actions within CRATE [13, 14]. Two central resources in the SVED framework are *managers* and *injectors*. These form a traditional master (manager) and slave (injector) relationship. The manager orchestrates which actions that are to be executed based on graph dependencies set by the user. Executing an action means notifying the responsible injector, which has the actual logic and necessary tools for running the action. The injector then updates the manager regarding the progress

[2]The OSVDB was discontinued during 2016.
[3]`https://www.exploit-db.com/`

of the action during its execution. This architecture enables carrying out high-fidelity and low-effort tests of next to arbitrary scale.

The present study employed CRATE as a testbed for safely testing exploits, and SVED for running all attacks as well as their auxiliary requirements (e.g., file copying, restoring virtual machine snapshots and simulating users browsing the web).

3.2. Generation of samples

This section describes how CRATE and SVED were used to generate a selection of samples concerning success rates of exploits. An overview of this process is illustrated by Figure 1 and described in the remainder of this section. In short, this process ensured that each exploit was tested under valid conditions.

3.2.1. Vulnerability identification. Authenticated vulnerability scans were performed using the tool OpenVAS to identify vulnerabilities in the machines deployed in CRATE. Authenticated scans were chosen as system credentials generally are available to IT administrators and this kind of scan provides more in-depth and accurate results (e.g., local tests such as analyzing the registry of a Windows machine compared to only probing open sockets). OpenVAS was chosen as it is open-source and freely available. Previous research indicate that the results likely would have been similar if other tools such as Nessus and Nexpose were used [1]. An in-house developed tool called AutoVAS was used to enable swiftly scanning the cyber range through multiple parallel instances of OpenVAS.

A total of 1179 machines were operational in CRATE during the present study. These contained a total of 719 132 vulnerabilities, i.e., on average 199 vulnerabilities per machine. The number of vulnerabilities in a vulnerable machine varied greatly, from a few to several thousand.

3.2.2. Exploit selection and execution. The detected vulnerabilities were matched with exploits in Metasploit that referenced them. The use of Metasploit is motivated by its popularity and accessibility. Examined exploit attack vectors include server exploits, remote client exploits, local exploits and file format exploits. All employed exploits provided or escalated privileges. I.e., no gathering- or denial of service-type attacks were used. The rational is that vulnerabilities are frequent, and vulnerabilities that do not provide attackers with privileges thus tend to have a low priority for network administrators. Each exploit category required different preparation and execution steps in SVED to enable reliable tests. These steps are described in the remainder of this section.

Server exploits were tested in the same fashion as in [14]. In summary, this methodology involved running three preparation steps before running the actual exploit. First, the victim virtual machine (VM) was restored to a snapshot corresponding to a known vulnerable state scanned by OpenVAS. Then, if necessary, the VxLAN of the injector was changed so that it could communicate with the victim. An ICMP Echo request was then sent to ensure that the victim was reachable. Finally, the exploit itself was attempted. If any of these steps were unsuccessful, and the victim had untested network interfaces, these steps were repeated until all available network interfaces had been explored. This was done as a server could have been setup to listen only to traffic incoming to a specific network interface.

The remaining exploit categories shared the same three preparation steps of server-side exploits, but required more effort to automatically test in a reliable manner as they all required end-user interaction. The SVED class `TriggerExploit` was built to enable automatically running the necessary end-user commands to fulfill such interaction requirements.

Remote client exploits in general require victim interaction in terms of running an application with specific arguments that facilitate communication with a malicious server. A typical example is a web browser exploit. This kind of exploit requires starting a web server with malicious content on the injector. Then, the victim needs to browse the content of this web server with a designated vulnerable web browser. Finally, the victim might need to press some buttons to run scripts provided by the web server. For a general remote client-side exploit, `TriggerExploit` first logs into the victim in the context of an administrator user. This step first required extending VirtualBox as it did not support graphical login for most operating system graphical environments used in the experiment.

When the designated user has been logged in, a series of events related to application execution and mouse/keyboard inputs are triggered. Which events that are run depend on the vulnerable software and the victim's operating system. Needless to say, extensive manual labor was spent to build profiles corresponding to different exploits and applications. Support was built for all exploits that corresponded to vulnerabilities identified by OpenVAS for this experiment.

Local exploits involved copying and executing a specially crafted malware on the victim through the VirtualBox API, yielding a back door compatible to the Metasploit framework. Then, a Metasploit

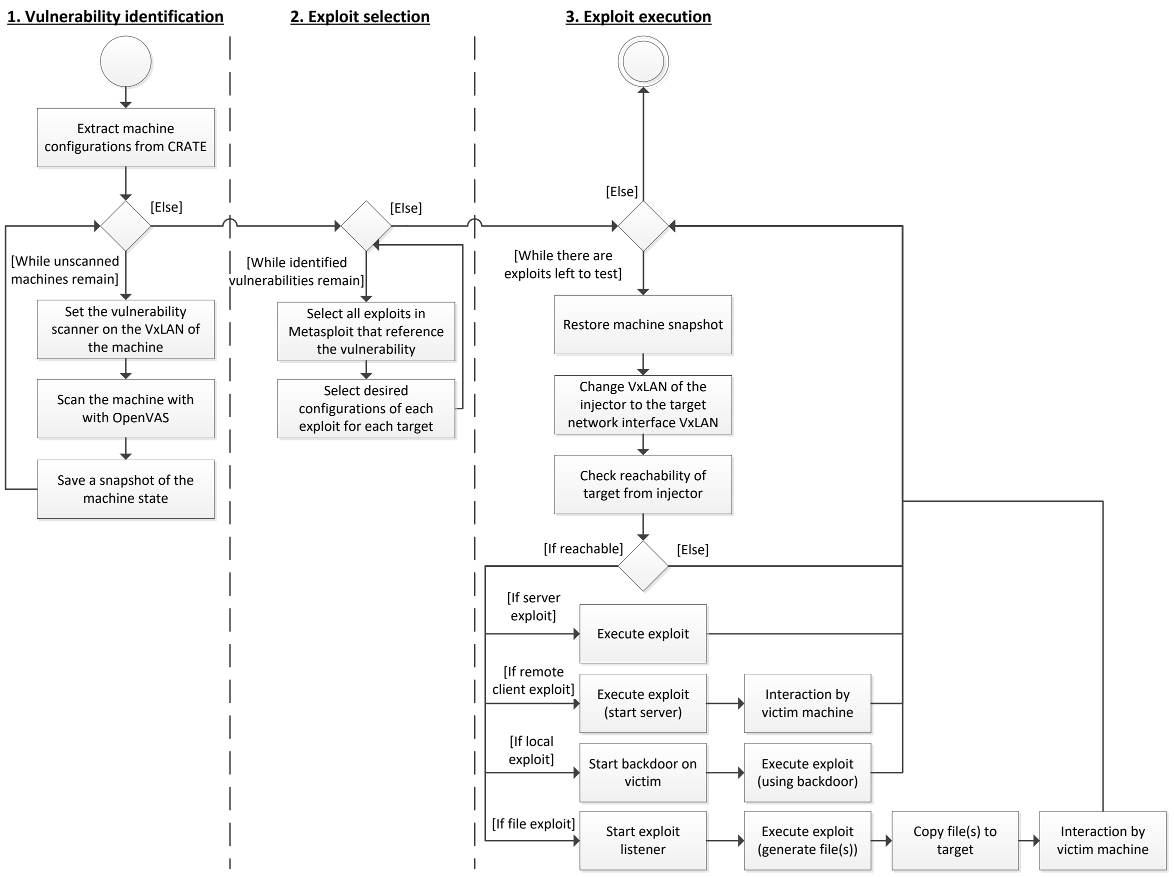
1. Vulnerability identification
2. Exploit selection
3. Exploit execution
Extract machine configurations from CRATE
[Else]
[While unscanned machines remain]
Set the vulnerability scanner on the VxLAN of the machine
Scan the machine with with OpenVAS
Save a snapshot of the machine state
[Else]
[While identified vulnerabilities remain]
Select all exploits in Metasploit that reference the vulnerability
Select desired configurations of each exploit for each target
[Else]
[While there are exploits left to test]
Restore machine snapshot
Change VxLAN of the injector to the target network interface VxLAN
Check reachability of target from injector
[If reachable]
[Else]
[If server exploit]
Execute exploit
[If remote client exploit]
Execute exploit (start server)
Interaction by victim machine
[If local exploit]
Start backdoor on victim
Execute exploit (using backdoor)
[If file exploit]
Start exploit listener
Execute exploit (generate file(s))
Copy file(s) to target
Interaction by victim machine

Figure 1. Experiment design.

`multi/handler` was used to obtain a session to the victim machine. Finally, if all these steps were successful, the designated local exploit was run in the context of the obtained session.

File format exploits involved creating one or more malicious files, copying them to the victim, and then executing them through specially crafted application commands, keypresses and mouseclicks (similarly to remote client-side exploits).

Exploits were configured through the same methodology that has been employed by popular automated attack tools such as DeepExploit [19], APT2 [20] and AutoSploit [21]. This methodology is described next.

Each exploit and payload in the Metasploit framework has a list of settings that can be used to tune it, and changing a setting sometimes requires extensive domain knowledge. For instance, the exploit module `lsa_transnames_heap` has 52 settings that can be altered. To specify most settings, such as which SMB pipe name to use, deep technical understanding of the exploit as well as the target application is required. This experiment only altered settings that were required to test each exploit (e.g., `RHOST`, `SRVHOST`, `UNCPATH` and `URIPATH`), and left non-obvious settings to their default states.

One important setting that should be mentioned is `TARGET`, which concerns a configuration within an exploit that corresponds to a certain application version or environment. For instance, `ms08_067_netapi` has 73 targets for Microsoft Windows 2000, Windows Server 2003 and Windows XP (e.g., Microsoft Windows XP SP2 with a Greek language pack and NX enabled). Choosing the correct target is important as an incorrect target renders a valid exploit unsuccessful in most cases. Some exploits in Metasploit have an option to automatically choose the correct target depending on a run-time check. The present research employed automated targeting for exploits that supported it. In other cases, each `TARGET` configuration was tested. The default payload for each target was chosen.

3.2.3. Overview of conducted attacks. The experiment was carried out over a period of eight calendar days using 40 SVED injectors and produced a total of 1 013 691 log entries. All data are available for download[4]. On overall, a mere 185 out of 5839 (3.2%) exploit attempts were successful. Here, an *exploit attempt* means that all necessary conditions for a reliable exploitation test were fulfilled. Some exploits could not be properly triggered due to errors in VirtualBox or victim machines. For example, some file format exploits could not be copied to victims due to corrupt VirtualBox machine states. Such tests were disregarded from analysis.

A total of 194 unique exploit modules were tested. An overview of the 37 exploits types that succeeded is presented by exploit type in Table 1. The three most successful exploit types that were tested ten or more times were `ms08_067_netapi` (75% of all attempts successful), `firefox_webidl_injection` (74% of all attempts successful) and `ms07_017_ani_loadimage_chunksize` (52% of all attempts successful).

Table 1. Successful exploit attempts.

Exploit	Vector	Attempts	Success
ms08_067_netapi	Server	53	40
ms17_010_eternalblue	Server	236	33
firefox_webidl_injection	Client	19	14
ms07_017_ani_loadimage_chunksize	Client	23	12
java_verifier_field_access	Client	32	6
ms03_026_dcom	Server	29	6
ms10_002_aurora	Client	21	6
ms10_018_ie_behaviors	Client	20	6
ms06_040_netapi	Server	232	5
msvidctl_mpeg2	Client	26	5
ms09_072_style_object	Client	21	5
adobe_flash_otf_font	Client	10	5
java_storeimagearray	Client	25	4
ms10_090_ie_css_clip	Client	8	4
ms12_037_ie_colspan	Client	14	3
java_atomicreferencearray	Client	3	3
ms10_025_wmss_connect_funnel	Server	3	3
java_setdifficm_bof	Client	27	2
ms13_053_schlamperei	Local	12	2
ppr_flatten_rec	Local	12	2
java_trusted_chain	Client	5	2
java_rhino	Client	3	2
ms01_033_idq	Server	22	1
ms13_055_canchor	Client	15	1
ms11_081_option	Client	13	1
adobe_media_newplayer	Client	12	1
ie_execcommand_uaf	Client	12	1
ie_setmousecapture_uaf	Client	11	1
ms12_037_same_id	Client	11	1
ms13_037_svg_dashstyle	Client	11	1
ms13_081_track_popup_menu	Local	11	1
ms11_003_ie_css_import	Client	10	1
adobe_collectemailinfo	File	10	1
firefox_proto_crmfrequest	Client	6	1
java_rmi_connection_impl	Client	3	1
ms13_022_silverlight_script_object	Client	1	1
ms10_092_schelevator	Local	1	1

[4]`ftp://download.iwlab.foi.se/sved/`

3.2.4. Sample selection. Samples for machine learning were generated from the 5839 exploit attempts as follows. Each sample concerns an exploit module run against a specific machine. If any tested configuration of the exploit worked against the machine, the outcome of the sample was set as 1; in other cases the outcome was set as -1. This process amounted into 4945 samples. Each sample was then provided with a set of features as is described in section 3.3.

3.3. Feature selection

Features were selected based on the information that can be extracted from authenticated network vulnerability scans, i.e., an important subset of the information that typically is available to network administrators. Four categories of features were tested (see Table 2): features related to the vulnerable machine, features related to the vulnerable software, features related to the vulnerability itself, and features related to the exploit that was tested in different configurations to validate the vulnerability.

All categorical variables were encoded through standard one-hot-encoding, yielding a number of features corresponding to the dimensionality of each variable plus one (to account for missing values and new entries not covered by the encoding). The CVSS v2 was applied in front of CVSS v3 as only a small fraction of all CVE entries currently are scored by the CVSS v3. The machine and software concerns the CVSS Environmental Metrics, the vulnerability the CVSS Base Metrics, and the exploit the CVSS Temporal Metrics. Thus, aspects from each metric group within the CVSS was taken into account. The variables in Table 2 concern a total of 742 features.

Each of the 4945 samples were provided states of these 742 features, enabling supervised ML learning.

3.3.1. Employed features. A machine that has not been hardened, i.e, a machine that has a high overall vulnerability (*Machine:Sum of Base Scores*), many installed applications (*Machine:CPE count*), or multiple exposed common sockets (*Machine:Open ports*), is hypothesized to be less likely to have security protections and software fixes configured that would prevent successful exploitation. The installed operating system (*Machine:CPE*) is of importance as an exploit might not support it. For example, `ms08_067_netapi` does not support Windows Embedded targets, even though Windows Embedded is based on Windows XP, and thus also is vulnerable to CVE-2008-4250 (i.e., the target vulnerability).

Similarly, a more vulnerable application is hypothesized to be less likely to be configured with fixes that prevent exploitation (*Application:Sum of Base Scores*, *Application:CPE*, *Application:Exposed port*).

The CVSS Base Score of the vulnerability could impact successful exploitation in various ways (*Vulnerability:Base Score*). For example, vulnerabilities that are easier for an attacker to utilize (e.g., exploits that do not require any user interaction) might receive more development effort from exploit authors. CVSS Access Complexity (*Vulnerability:Access Complexity*) and Authentication (*Vulnerability:Authentication*) were specifically selected as they concern the difficulty of triggering a vulnerability. CVSS impact metrics were disregarded as all tested exploits concern privilege escalation, and the exploitability metric Access Vector was disregarded as all exploits were run in their designated scopes (see Section 3.2.2). CVSS environmental metrics and temporal metrics were disregarded as their states are undefined for vulnerabilities in the NVD. Older vulnerabilities are hypothesized to be easier to exploit as they often concern less hardened software (*Vulnerability:Time since disclosure*). Threat intelligence information (*Vulnerability:Snort IDS rules*[5], *Vulnerability:Symantec threats*[6] and *Vulnerability:VERIS incidents*[7]) could indicate exploit reliability as a more reliable exploit likely sees more usage in the wild. The type of vulnerability that is concerned (*Vulnerability:CWE*[8]) could influence exploit reliability as different types of vulnerabilities have received different developer focus in terms of protection mechanisms.

Each exploit in Metasploit has been assigned a reliability rank based on its potential impact to the target system[9], which goes from manual to excellent (*Exploit:Metasploit reliability*). Exploit size (*Exploit:Lines of code*), target count (*Exploit:Number of targets*) and availability of automated targeting (*Exploit:Has autotarget*) should affect an exploit's ability to handle different victim configurations. A more complex exploit in terms of customization could also influence likelihood of exploitation (*Exploit:Settings*, *Exploit:Required settings*). If the target port in a server exploit differs from the port that the application has exposed, it is possible that the application's configuration has been altered in other means that affects the likelihood of exploit success (*Exploit:Default port is open*). An older exploit has had greater opportunity to be refined and is also more likely to concern an older software with less protection mechanisms (*Exploit:Time since publishing*).

[5]`https://rules.emergingthreats.net/`

[6]`https://www.symantec.com/security-center/a-z`

[7]`http://veriscommunity.net/`

[8]`https://cwe.mitre.org/`

[9]`https://github.com/rapid7/metasploit-framework/wiki/Exploit-Ranking`

3.3.2. Feature scaling methods. Four different scaling methods were tested: none, manual min-max scaling to a 0-1 domain, standard min-max scaling to a 0-1 domain, and scaling based on means and standard deviation. Manual scaling means that each feature was manually scaled to a 0-1 domain through domain knowledge. This was done as standard ML scaling methods do not account for future changes to variable states. For example, a standard scaling of the lines of code variable would only account for the lines of code observed in the samples. This would produce issues if the model is applied to a future exploit code that is significantly larger than the previous maximum. The manual scaling is thus not custom-fit to the observed samples and more robust to future state changes.

3.3.3. SMOTE ENN. Due to the large imbalance in the dataset, SMOTE ENN[10] [22] was used to generate a second balanced dataset by artificially over-sampling the minority class (i.e., successful exploits) through SMOTE and then cleaning the space resulting from over-sampling through Edited Nearest Neighbours. SMOTE ENN was chosen as it produced the best results for imbalanced datasets in previous studies [23] and have been used in the cyber security domain, e.g., regarding intrusion detection [24].

3.4. Creation of classifiers

Three kinds of classifiers were trained through supervised learning: support vector machines (SVM), random forests (RF) and neural networks (NN). These were chosen as they are standard algorithms that have been used with success on various previous occasions [25]. A standard ML library, scikit-learn, was used for training the classifiers. Parameter tuning was performed based on k-fold cross-validation with a k of 3 on the two datasets (the original dataset and the dataset revised by SMOTE ENN). The F1 metric was used for evaluating model fit in front of the standard accuracy metric due to the imbalance in the dataset. After training, the best SVM, RF and NN model were subjected to the

[10]`https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.combine.SMOTEENN.html`

Table 2. Employed features.

Scope	Variable	Data type	Description
Machine	Sum of Base Scores	Float	The sum of all CVSSv2 Base Scores for all vulnerabilities in the machine
	CPE count	Integer	The total number of identified Common Platform Enumeration entries in the machine
	Open ports	Categorical (21 features)	The open ports in the machine according to the top 20 ports given by Nmap
	CPE	Categorical (64 features)	Common Platform Enumeration entry of the operating system
Application	Sum of Base Scores	Float	The sum of all CVSSv2 Base Scores for all vulnerabilities in the application
	CPE	Categorical (480 features)	Common Platform Enumeration entry of the vulnerable application
	Exposed port	Categorical (21 features)	The port exposed by the software mapped to the top 20 ports given by Nmap
Vulnerability	Base Score	Float	The CVSSv2 Base Score of the vulnerability
	Access Complexity	Categorical (4 features)	The CVSSv2 Access Complexity state of the vulnerability
	Authentication	Categorical (4 features)	The CVSSv2 Authentication state of the vulnerability
	Time since disclosure	Float	The elapsed time since the vulnerability was disclosed
	Snort rules	Integer	Entries in Snort network intrusion detection system Emerging Threats rulesets
	VERIS incidents	Integer	Entries in VERIS incidents
	Symantec threats	Integer	Entries in Symantec threats
	CWE	Categorical	Common Weakness Enumeration entry for the vulnerability
Exploit	Metasploit reliability	Categorical (8 features)	The Metasploit reliability index of the exploit
	Number of targets	Integer	The number of targets for the exploit
	Has autotarget	Boolean	If the exploit has an "auto" option for TARGET
	Lines of code	Integer	The number of lines of code in the exploit
	Default port is open	Boolean	If the default port for the exploit is open in the victim machine
	Settings	Integer	The number of settings of different kinds (e.g., boolean, integer, float, string)
	Required settings	Integer	The number of required settings of different kinds (e.g., boolean, integer, float, string)
	Time since publishing	Float	The elapsed time since the exploit was published

same tests. These tests involved estimating model fit over 50 datasets that were generated by ten runs of k-fold validation with a k of 5 on the overall dataset. Evaluations of models based on SMOTE ENN were made by usage of pre-computed SMOTE ENN revised versions of the training-parts of these 50 datasets.

The SVM model was trained according to the guidelines in [26, 27]. In summary, this meant training a C-SVM with an rbf kernel through a grid search for the optimal values of the parameters *C*, *gamma* and *class_weight*. A total of 8100 different parameter combinations were tested.

The RF model was trained according to the guidelines in [28, 29]. Six parameters were used to tune the model: *n_estimators*, *max_features*, *max_depth*, *min_samples_split*, *mean_samples_leaf* and *bootstrap*. Due to time constraints, instead of evaluating all 8640 relevant parameter combinations, a random grid search model was used to test 100 of these combinations. As RF algorithms are known to perform well even with their default parameters [28], 100 combinations was considered sufficient.

The NN model that is implemented in scikit-learn is a multi-layer perceptron algorithm that employs backpropagation. Three parameters were selected for tuning it: *learning_rate*, *hidden_layer_sizes* and *activation*. This resulted in 45 combinations that were tested through a grid search.

4. Results

An overview of the results for the examined classifiers is presented in Table 3. The numbers described in Table 3 concern the overall mean values as well as the lowest and highest values observed across the 50 tests. The highest mean F1 score (0.734) as well as the lowest error rate (1.8%) were observed for the RF model. This model had 400 trees in the forest, a minimum of 2 splits per node, a minimum of 1 sample to be a leaf, a max tree depth of 30, used bootstrap samples when building trees, and employed feature scaling according to mean values and standard deviation. The best SVM model had a C of 1, a gamma of 0.5, class weights according to the sample sizes, and employed standard min-max scaling to the 0-1 domain. The best NN model employed a rectified linear unit (relu) activation function, two hidden layers (with 50 and 30 neurons), inverse scaling exponent-based learning rate, and used standard min-max scaling to the 0-1 domain.

The models were best at correctly classifying non-exploitable vulnerabilities. When a detected vulnerability was not exploitable, the RF model correctly classified it 99.3% of the time; when a detected vulnerability actually was exploitable, the RF model correctly classified it 68.4% of the time. The best NN model had a similar distribution, albeit lower scores. The results differed a bit for the best SVM model, which correctly classified 78.7% of all exploitable vulnerabilities and 98.3% of the non-exploitable vulnerabilities.

SMOTE ENN did not increase the performance of any tested model. The best models when applying SMOTE ENN had mean F1 scores of 0.706 (RF, 4% worse), 0.622 (SVM, 12% worse) and 0.592 (NN, 14% worse).

The relative importance of different features was analyzed through the best model (RF). Out of the 30 most significant features, 9 concern the exploit, 8 the vulnerability, 8 the application, and 5 the machine. Thus, while there are valuable features in all categories, the results suggest that aspects related to the environment are less important than the employed exploit and the targeted vulnerability. This observation is enforced by the fact that the two by far most significant features concern the exploit (Lines of code and the Number of targets).

Metasploit's reliability index occur five times in the top 30 features: Excellent (rank 18), Good (rank 20), Average (rank 22), Medium (rank 27) and Low (rank 30). It is reasonable that these are important features, and that their relative rank reflect their influence on exploit reliability as hypothesized by their authors. It is, however, interesting that there are 17 features which are more important than these manually defined reliability ranks - including simply counting exploit's lines of code. The CWE standard provides some insight into this question. Nine CWE entries account for 82% of all samples, and seven of these (66% of all samples) concern memory corruption vulnerabilities. As even a minor alteration in the target could affect the memory layout of a vulnerable function, an exploit for this kind of vulnerability typically needs many built-in configurations to be reliable.

The fact that features related to different kinds of exploit settings are important (boolean at rank 7, string at rank 15 and list at rank 21) suggest that exploits that are more complex in terms of configuration could affect the likelihood of success.

5. Discussion, conclusions and future work

This paper presented a model for measuring the likelihood that a detected vulnerability can be exploited. ML models were trained based on data collected during

Table 3. Overview of results for best models, presented as mean (min, max).

Model	F1	Precision	Recall	Accuracy
RF	0.734 (0.567, 0.853)	0.796 (0.619, 0.974)	0.686 (0.452, 0.857)	0.982 (0.969, 0.991)
SVM	0.706 (0.577, 0.842)	0.644 (0.484, 0.800)	0.789 (0.643, 0.971)	0.976 (0.957, 0.988)
NN	0.690 (0.522, 0.800)	0.721 (0.480, 0.875)	0.670 (0.533, 0.857)	0.978 (0.969, 0.986)

a cyber security experiment involving more than 5000 exploit attempts. The best results were found for a RF model, which had a mean cross-validation accuracy of 98.2% and an F1 score of 0.73. This result is significantly better than the most similar related work [15], which merely estimates the availability of exploits, and has an accuracy of 86%.

There are other use-cases for the data and tools presented in this paper. For example, the results can be used for more realistic game theoretical [30] or attack graph [31] simulations, which often are based on completely fictitious data. Furthermore, as each activity that is carried out by SVED (e.g., an exploit or a user browsing the web) has a ground truth established with a millisecond precision through a combination of structured logs and raw system/network events (e.g., network captures), the results of an experiment enable standardized and reliable training and evaluation of intrusion detection tools and rulesets. While validity issues certainly would remain, it would arguably be significantly better than any variant of the twenty year old DARPA dataset.

The remainder of this chapter critically discusses the obtained results along with some key topics and presents future work regarding these issues.

Automated network vulnerability scanners produce false positives. Some exploits likely failed due to false positives by OpenVAS. The results should be viewed in this light: they are valid to vulnerabilities detected by authenticated network vulnerability scans, but not necessarily valid to other vulnerability detection methods.

The machines and configurations in the employed cyber range do not accurately reflect real-world operational systems. While this is true, the included machines and configurations were set up for a plethora of purposes, by a variety of personnel during the course of more than a decade. Thus, they have a high variability. Furthermore, aspects that are difficult to simulate well within a cyber range, but are of importance to security audits in the real world (e.g., the information stored in machines) are irrelevant for the experiment described in this paper. Thus, the experiment scenario should provide sufficiently reliable results.

Only exploits in Metasploit were considered. There are a plethora of exploits in the public domain that have not been implemented in Metasploit that attackers with some technical knowledge obviously can utilize. For example, there are more than 40 000 entries within the Exploit-DB. We plan to develop automated methods for translating such exploits into working Metasploit-compatible modules in future work.

Alternative features. As for any ML task, there are various alternative features that can be considered. For example, this research employed one-hot-encoding for converting categorical variables into features. While this is a standard method for this purpose, an alternate method called *entity embeddings* [32] is gaining popularity. Entity embeddings loosely means mapping each dimension of a categorical variable to a fixed-dimension space. This enables reducing the dimensionality of a categorical variable from N to P dimensions and furthermore enables mapping values that are similar in terms of output. We will explore additional features such as entity embeddings in future work.

More samples. While the dataset analyzed in this paper is larger than previous studies of the same topic, it is small in comparison to general ML tasks that typically includes millions of samples. In particular, neural networks are not very robust to low samples sizes compared to RF and SVM models. We are researching methods for enabling scaled up experiments. One promising such method is to deploy machines with configurations provided by packet managers such as Chocolatey[11] and Puppet[12] according to statistics observed from operational systems.

References

[1] H. Holm, T. Sommestad, J. Almroth, and M. Persson, "A quantitative evaluation of vulnerability scanning," *Information Management & Computer Security*, vol. 19, no. 4, pp. 231–247, 2011.

[2] M. Frigault, L. Wang, S. Jajodia, and A. Singhal, "Measuring the overall network security by combining cvss scores based on attack graphs and bayesian networks," in *Network Security Metrics*, pp. 1–23, Springer, 2017.

[3] H. Holm and K. K. Afridi, "An expert-based investigation of the common vulnerability scoring

[11]`https://chocolatey.org/`

[12]`https://puppet.com/`

system," *Computers & Security*, vol. 53, pp. 18–30, 2015.

[4] J. Nazario, T. Ptacek, and D. Song, "Wormability: A description for vulnerabilities," *Arbor Networks (October 2004)*, 2004.

[5] L. Tidy, K. Shahzad, M. A. Ahmad, and S. Woodhead, "An assessment of the contemporary threat posed by network worm malware," 2014.

[6] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer networks*, vol. 34, no. 4, pp. 579–595, 2000.

[7] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

[8] L. Su, Y. Yao, N. Li, J. Liu, Z. Lu, and B. Liu, "Hierarchical clustering based network traffic data reduction for improving suspicious flow detection," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 744–753, Aug 2018.

[9] J. Yan, D. Jin, C. W. Lee, and P. Liu, "A comparative study of off-line deep learning based network intrusion detection," in *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 299–304, IEEE, 2018.

[10] K. A. Farris, A. Shah, G. Cybenko, R. Ganesan, and S. Jajodia, "Vulcon: A system for vulnerability prioritization, mitigation, and management," *ACM Transactions on Privacy and Security (TOPS)*, vol. 21, no. 4, p. 16, 2018.

[11] T. Dumitraş and P. Efstathopoulos, "Ask wine: are we safer today? evaluating operating system security through big data analysis," in *Proceedings of the 5th USENIX conference on Large-Scale Exploits and Emergent Threats*, pp. 11–11, USENIX Association, 2012.

[12] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Experience with deter: a testbed for security research," in *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006. 2nd International Conference on*, pp. 10–pp, IEEE, 2006.

[13] H. Holm and T. Sommestad, "Sved: Scanning, vulnerabilities, exploits and detection," in *Military Communications Conference, MILCOM 2016-2016 IEEE*, pp. 976–981, IEEE, 2016.

[14] H. Holm and T. Sommestad, "So long, and thanks for only using readily available scripts," *Information & Computer Security*, vol. 25, no. 1, pp. 47–61, 2017.

[15] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 105–114, ACM, 2010.

[16] L. Allodi and F. Massacci, "Comparing vulnerability severity and exploits using case-control studies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 1, p. 1, 2014.

[17] H. Holm, M. Ekstedt, and D. Andersson, "Empirical Analysis of System-Level Vulnerability Metrics through Actual Attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, pp. 825–837, nov 2012.

[18] M. Granåsen and D. Andersson, "Measuring team effectiveness in cyber-defense exercises: a cross-disciplinary case study," *Cognition, Technology & Work*, vol. 18, no. 1, pp. 121–143, 2016.

[19] I. Takaesu, "Deep Exploit." `https://github.com/13o-bbr-bbq/machine_learning_security/tree/master/DeepExploit`, 2018. [Online; accessed 14-June-2019].

[20] A. Compton and A. Lane, "APT2 - An Automated Penetration Testing Toolkit." `https://github.com/MooseDojo/apt2`, 2018. [Online; accessed 14-June-2019].

[21] Vectorsec, "AutoSploit." `https://github.com/NullArray/AutoSploit`, 2018. [Online; accessed 14-June-2019].

[22] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge & Data Engineering*, no. 9, pp. 1263–1284, 2008.

[23] D. S. Sisodia, N. K. Reddy, and S. Bhandari, "Performance evaluation of class balancing techniques for credit card fraud detection," in *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, pp. 2747–2752, IEEE, 2017.

[24] H. H. Pajouh, G. Dastghaibyfard, and S. Hashemi, "Two-tier network anomaly detection model: a machine learning approach," *Journal of Intelligent Information Systems*, vol. 48, no. 1, pp. 61–74, 2017.

[25] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[26] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, "A practical guide to support vector classification," 2003.

[27] A. Ben-Hur and J. Weston, "A user's guide to support vector machines," in *Data mining techniques for the life sciences*, pp. 223–239, Springer, 2010.

[28] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 ed., 2009.

[29] A. Boulesteix, S. Janitza, J. Kruppa, and I. R. König, "Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 2, no. 6, pp. 493–507, 2012.

[30] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu, "A survey of game theory as applied to network security," in *2010 43rd Hawaii International Conference on System Sciences*, pp. 1–10, IEEE, 2010.

[31] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, "Dag-based attack and defense modeling: Don't miss the forest for the attack trees," *Computer science review*, vol. 13, pp. 1–38, 2014.

[32] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," *arXiv preprint arXiv:1604.06737*, 2016.