AutoCVSS: An Approach for Automatic Assessment of Vulnerability Severity Based on Attack Process

Deqing Zou[1,2], Ju Yang[1], Zhen Li[1,3(✉)], Hai Jin[1], and Xiaojing Ma[1]

[1] National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, Big Data Security Engineering Research Center, Huazhong University of Science and Technology, Wuhan, China
lizhen_hust@hust.edu.cn

[2] Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen, China

[3] School of Cyber Security and Computer, Hebei University, Baoding, China

Abstract. Vulnerability severity assessment is an important research problem. *Common Vulnerability Scoring System* (CVSS) has been widely used to quantitatively assess the vulnerability severity, but its assessment process relies on human experts to determine metric values, which makes the assessment process tedious and subjective. This calls for tools that can assess the vulnerability severity *automatically* and *objectively*. In this paper, we move a step forward in this direction by proposing an approach for automatic assessment of vulnerability severity based on attack process, dubbed *Automatic Common Vulnerability Scoring System* (AutoCVSS). The key insight is to leverage characteristics and rules we define to model the CVSS base metrics, and assess the vulnerability severity more automatically and objectively by capturing the attributes related to the characteristics during the attack process. In order to evaluate AutoCVSS, we reproduce the attacks for 98 vulnerabilities from Linux kernel, FTP service, and Apache service with their exploits. The experimental results show that the vulnerability severity scores automatically obtained by AutoCVSS are basically in accordance with those assessed manually by security experts in the *National Vulnerability Database* (NVD), which verifies the effectiveness of our approach.

Keywords: CVSS · Vulnerability severity assessment · Software vulnerability · Attack process

1 Introduction

Most of security incidents are caused by vulnerabilities. A variety of security vulnerabilities have brought huge economic losses around the world each year, and the situation becomes more and more serious. Prioritizing vulnerabilities

R. Miani et al. (Eds.): GPC 2019, LNCS 11484, pp. 238–253, 2019.
https://doi.org/10.1007/978-3-030-19223-5_17

that are in urgent need of patching can be used to minimize the losses [21]. Therefore, many security vendors and security agencies have done researches on the vulnerability severity assessment and put forward their own vulnerability severity assessment systems and evaluation criteria [17–19]. In order to solve the inconsistency and incompatibility problems caused by various security assessment systems, *National Infrastructure Advisory Council* (NIAC) proposes an open and common vulnerability assessment system called *Common Vulnerability Scoring System* (CVSS) [1] which uses a value between 0–10 to represent the vulnerability severity. A higher score value indicates a greater vulnerability severity [17].

However, CVSS relies on human experts to determine metric values during the process of vulnerability severity assessment, which makes the assessment process tedious and subjective [15,20–22]. In principle, the subjective problem can be alleviated by asking multiple experts, and then select the majority opinion. But this imposes even more tedious work. As a matter of fact, it is desirable to reduce, or even eliminate whenever possible, the reliance on the intense labor of human experts. This calls for tools that can *automatically* and *objectively* assess the vulnerability severity to prioritize vulnerabilities that are in urgent need of patching. The research problem can be described as follows: *When a vulnerability is discovered and its exploits or Proof of Concepts (PoCs) are submitted to the security authority, how can the vulnerability severity be assessed automatically and objectively?*

In order to answer the above question, we present the first approach for automatic assessment of vulnerability severity, dubbed *Automatic Common Vulnerability Scoring System* (AutoCVSS). The goal is to reduce the reliance on the intense labor of human experts and make the assessment process of CVSS more automatically and more objectively. Specifically, we propose a group of *characteristics* and *rules* to model each CVSS base metric according to its description. The characteristics reflect the features of each CVSS base metric, and the rules show its evaluation basis. The characteristics of each CVSS base metric are represented by a group of attributes, which can be captured during the attack process and used to evaluate the vulnerability severity according to the rules.

In order to evaluate AutoCVSS, we reproduce the attacks for 98 vulnerabilities of Linux kernel, FTP service, and Apache service with their exploits from *Exploit Database* (EDB) [2]. The experimental results show that the vulnerability severity scores automatically obtained by AutoCVSS are basically in accordance with those assessed manually by security experts in the *National Vulnerability Database* (NVD) [3].

2 Background

In this section, we briefly describe the background on CVSS, an open and common vulnerability severity assessment system provided by NIAC. CVSS has three groups of metrics: base metrics, temporal metrics, and environmental metrics [1].

In this paper, we mainly focus on base metrics reflecting the inherent characteristics of a vulnerability which are not influenced by time and users' environments. On one hand, the vulnerability severity assessment for CVSS must involve base metrics, while temporal metrics and environmental metrics are optional. On the other hand, for a vulnerability, the values of base metrics are fixed and available in the NVD, while the values of temporal metrics and environmental metrics vary with time or users' environments and are not available in the NVD. Therefore, the base metrics can be used as benchmarks for comparison. In addition, NVD uses the CVSS version 2 (i.e., CVSS v2) to evaluate vulnerabilities before CVSS version 3 (i.e., CVSS v3) was put forward in 2015, and uses both CVSS v2 and CVSS v3 for vulnerability assessment now. That is to say, almost all vulnerabilities in the NVD provide CVSS v2, and many of them do not provide CVSS v3 at the time of writing. In this paper, we select the version(s) of CVSS for each vulnerability as the NVD does.

Base metrics involve two sets of metrics: exploitability metrics and impact metrics [1]. In CVSS v2, *exploitability metrics* contain three metrics: Attack Vector (AV), Attack Complexity (AC), and Authentication (AU). These metrics are used to show how the vulnerability is accessed and whether extra conditions are required to exploit it. *Impact metrics* also contain three metrics: Confidentiality Impact (C), Integrity Impact (I), and Availability Impact (A). These metrics represent the impact of a successfully exploited vulnerability. In CVSS v3, exploitability metrics, different from those in CVSS v2, contain Attack Vector (AV), Attack Complexity (AC), Privileges Required (PR), User Interaction (UI), and Scope (S), and impact metrics are the same as those in CVSS v2. A vulnerability is assigned a CVSS base score ranging from 0 to 10. A higher score indicates a greater vulnerability severity.

3 Design of AutoCVSS

AutoCVSS has two phases: the monitoring program generation and the vulnerability severity assessment, as shown in Fig. 1. In the monitoring program generation phase, the characteristics and rules are defined to model the CVSS base metrics. For each characteristic, analyze the attributes that are captured during the attack, and then generate the monitoring program. In the vulnerability severity assessment phase, the probes of attributes involved in the monitoring program are used to instrument the exploits/PoCs, capture the attributes, and monitor the state of vulnerable software. After the process of hierarchical evaluation, the vulnerability severity is output.

3.1 Input and Output

The input of AutoCVSS consists of the description of CVSS base metrics, the *Common Vulnerabilities and Exposures IDentifier* (CVE ID), the exploits/PoCs, and the vulnerable software. The description of CVSS base metrics is used to

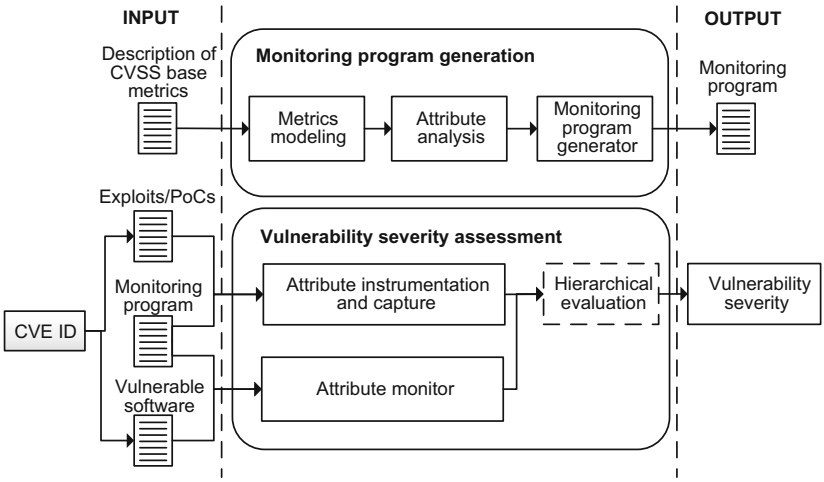

Fig. 1. Overview of AutoCVSS: the first phase generates the monitoring program and the second phase assesses the vulnerability severity. The characteristics and rules for metrics modeling in the first phase need to be defined, and the subsequent process of AutoCVSS does not involve human interaction.

model the CVSS base metrics. The CVE ID is the unique identifier of vulnerability and is used to obtain the exploits/PoCs and the vulnerable software related to the vulnerability. The exploits/PoCs for the CVE ID can be gathered from the public websites such as EDB [2]. The vulnerable software can be obtained from the relevant official websites. Besides, the monitoring program, as the output of the monitoring program generation phase, is another input of the vulnerability severity assessment phase.

The final output of AutoCVSS is the vulnerability severity, which involves the vulnerability security score and the assessment process. The vulnerability security score ranges between 0 and 10. The higher the score is, the greater the vulnerability severity is. The assessment process shows the process of hierarchical evaluation clearly, such as the level of each base metric and the evaluation basis.

It is worth noting that if the CVE ID corresponds to multiple exploits/PoCs, we use one exploit/PoC as an instance at a time to assess the vulnerability severity, and then the highest score of all instances for the CVE ID is selected as the severity of this vulnerability.

3.2 Monitoring Program Generation

In the monitoring program generation phase, there are three modules: metric modeling, attribute analysis, and monitoring program generator.

Metrics Modeling. We model the base metrics of CVSS v2 and CVSS v3 according to the description of CVSS base metrics. Base metrics are represented

as a set $B = \{EM, IM\}$, where EM represents the exploitability metrics and IM represents the impact metrics. $EM = \{AV, PR, AC, AU, UI, S\}$, where AV represents the attack vector, PR represents the privileges required, AC represents the attack complexity, AU represents the authentication, UI represents the user interaction, and S represents the scope. The exploitability metrics reflect the features of vulnerability, such as how the vulnerability is accessed and whether or not extra conditions are required to exploit it. $IM = \{C, I, A\}$, where C, I, and A represent the confidentiality impact, integrity impact, and availability impact respectively. The impact metrics represent the impact of a successfully exploited vulnerability.

Each metric in EM and IM is modeled by one or several characteristics and corresponding rules. Table 1 shows the set of characteristics for each base metric, the meanings of characteristics, and the corresponding rules. We take exploitability metric AV and impact metric C for examples to explain the characteristics and their corresponding rules in detail.

AV reflects how the vulnerability is exploited. We define the characteristic $Mode$ to represent the attack mode that the attacker could choose. The value of $Mode$ involves network attack (N), adjacent attack (A), local attack (L), and physical attack (P). The rules for evaluating the level of AV are defined as follows. AV has four levels: network, adjacent, local, and physical. If $Mode$ is N, $level(AV)$ = network, where function $level(AV)$ represents the level of base metric AV; if $Mode$ is A, $level(AV)$ = adjacent; if $Mode$ is L, $level(AV)$ = local; if $Mode$ is P, $level(AV)$ = physical. The default initial level of AV is local.

C refers to confidentiality. If the attacker illegally reads the data, the confidentiality is affected. We define the characteristic IR to represent whether the read permission of the file is modified. The rules for evaluating the level of C are defined as follows. C has three levels: high, low, and none. If the user privilege is root, $level(C)$ = high. If IR is true and the file is sensitive, $level(C)$ = high. If IR is true and the file is non-sensitive, $level(C)$ = low. Otherwise, $level(C)$ = none. The default initial level of C is none.

Attribute Analysis. After modeling the base metrics, each characteristic is depicted by several attributes which can be monitored during the attack process. Considering that these attributes are related to the system that AutoCVSS is implemented on, we will provide the attributes for base metrics in Sect. 4. In this subsection, we take an attribute of an exploitability metric AV related to IP information for example to show the process of attribute analysis before generating the monitoring program.

First of all, when the attribute t of AV is captured, the IP address in t is obtained. By comparing the IP obtained from t with the IP of server, we can get the attack mode, and then obtain the temporary level of AV. Then a separate judgment is made to obtain the temporary level of AV, since the physical attack requires the access to physical devices. Finally, the temporary level of AV returns. It should be noted that the temporary level of AV does

Table 1. Base metrics modeling involves characteristics and corresponding rules. $level(bm)$ represents the level of base metric bm.

Base metric	Characteristics	Meaning of characteristics	Rules
AV	$Mode$	$Mode$ denotes the attack mode that the attacker could choose. The value of $Mode$ involves network attack (N), adjacent attack (A), local attack (L), and physical attack (P)	If $Mode$ is N, $level(AV)$ = network. If $Mode$ is A, $level(AV)$ = adjacent. If $Mode$ is L, $level(AV)$ = local. If $Mode$ is P, $level(AV)$ = physical
AC	$Cond$, $Action$	$Cond$ denotes the conditions under which the attack occurs. $Action$ denotes the actions performed by the attack	The number of conditions and actions that occur during the attack is counted. If $\#Cond = 0$ or 1, and $\#Action \leq 3$, $level(AC)$ = low. If $\#Cond = 0$ or 1, and $3 < \#Action \leq 5$, $level(AC)$ = medium. If $\#Cond > 1$ and $\#Action > 5$, $level(AC)$ = high. The default initial level of AC is low
PR	Fp, Up	Fp denotes the file privilege, and contains three file permissions: read, write, and execution. Up denotes the user privilege, and contains three user-level permissions: root, user, and guest	If Up is root, or Fp is the privileges of sensitive files for reading and writing, $level(PR)$ = high. If Up is not root, or Fp is the privileges of non-sensitive files for reading and writing, $level(PR)$ = low. Otherwise, $level(PR)$ = none. The default initial level of PR is none
AU	Os, Sw	Os denotes the operating system authentication, and reflects the attacker needs to authenticate the operating system. Sw denotes the software authentication, and reflects the attacker needs to authenticate the software	The number of operating system authentications and software authentications is counted. If $\#Os \geq 2$ or $\#Sw \geq 2$, $level(AU)$ = multiple. If $\#Os = 1$ or $\#Sw = 1$, $level(AU)$ = single. Otherwise, $level(AU)$ = none. The default initial level of AC is none
UI	sh	sh denotes the interactive interface with the operating system. If the exploit/PoC successfully creates an interactive interface or opens an interactive interface, it indicates that the exploit/PoC needs interaction	If sh is required, $level(UI)$ = required. Otherwise, $level(UI)$ = none. The default initial level of UI is none
S	Vc, Im	Vc denotes the vulnerable component, and its privilege belongs to the authorization scope $s1$. Im denotes the affected component, and its privilege belongs to the authorization scope $s2$. The authorization scope is the collection of privileges defined by a computing authority	If $s2 - (s1 \cap s2) \neq \varnothing$, and the privilege of Vc and Im does not belong to $(s1 \cap s2)$, $level(S)$ = changed. Otherwise, $level(S)$ = unchanged. The default initial level of S is unchanged
C and I	IR, IW	IR indicates whether the read permission of the file is modified. IW indicates whether the write permission of the file is modified	If the user privilege is root, $level(C)$ = high. If IR is true and the file is sensitive, $level(C)$ = high. If IR is true and the file is non-sensitive, $level(C)$ = low. Otherwise, $level(C)$ = none. The default initial level of C is none. The rules of I is basically similar to those of C, but I concerns whether there is illegal writing
A	Nu, Mu, Du, Cu	Nu denotes the network utilization, Mu denotes the memory utilization, Du denotes the disk utilization, and Cu denotes the CPU utilization	If the user privilege is root, $level(A)$ = high. If Nu (or Mu, or Du, or Cu) $\geq 80\%$, $level(A)$ = high. If $60\% \leq Nu$ (or Mu, or Du, or Cu) $< 80\%$, $level(A)$ = low. Otherwise, $level(C)$ = none. The default initial level of A is none

not mean the final level of AV which will be obtained from the hierarchical evaluation in the vulnerability severity assessment phase (Sect. 3.3).

Monitoring Program Generator. Based on the metrics model and the attribute analysis, the monitoring program can be generated by the probes of attributes. It is used to monitor the attributes during the attack process. These attributes can reflect not only the features of the attack, but also the impact of the system or software caused by the attack. The generated monitoring program is as one input of both attribute instrumentation and attribute monitor in the vulnerability severity assessment phase.

3.3 Vulnerability Severity Assessment

In the vulnerability severity assessment phase, the probes of attributes involved in the monitoring program are used to instrument the exploits/PoCs, capture the attributes, and monitor the state of vulnerable software and its environment. Then the vulnerability severity is obtained by hierarchical evaluation. The process involves three modules: attribute instrumentation and capture, attribute monitor, and hierarchical evaluation.

Attribute Instrumentation and Capture. The exploits/PoCs are instrumented with the attributes involved in the monitoring program. These attributes are related to the exploitability metrics AV, AC, PR, UI, AU, S and impact metrics C, I. The exploitability metrics mainly reflect the features of attack behavior, and the impact metrics monitor the impact of the system caused by the exploits/PoCs. The attributes that reflect impact metrics C and I are closely related to the exploitability metric PR, therefore the impact of C and I can be obtained based on the attributes of PR. The instrumentation does not affect the execution of the exploits/PoCs, and can be used to obtain the values of attributes accurately. In this paper, the instrumentation mainly focuses on system calls.

With the aid of instrumentation tool, the probes for attributes that are monitored in the monitoring program instrument the running exploits/PoCs. If the exploits/PoCs call the attributes monitored, the information on these attributes can be intercepted. The dynamic instrumentation approach to attribute capture can reflect the features of attack behavior and the impact of the system more objectively and accurately. Finally, the captured attributes are input to the hierarchical evaluation.

Attribute Monitor. Monitoring attributes is mainly related to the characteristics of impact metric A. The purpose is to monitor the impact of the system and vulnerable software caused by exploits/PoCs. A mainly reflects the availability of the system or vulnerable software throughout the attack process. Attributes related to A need to monitor the running status of the system or vulnerable

software in real time. Finally, the monitored attributes are input to the hierarchical evaluation. It should be noticed that monitoring the attributes is significantly different from instrumenting and capturing the attributes. Capturing the attributes occurs only when the instrumented attributes are encountered, while monitoring the attributes needs to continue throughout the attack process.

Hierarchical Evaluation. There are two inputs to the hierarchical evaluation: the captured attributes from the attribute instrumentation and capture module, and the monitored attributes from the attribute monitoring module. The output is the set *Result* which involves two parts: the vulnerability severity score and the assessment process involving the captured attributes and the final level of each base metric. The process of hierarchical evaluation has three steps.

Step 1: Deal with attributes related to exploitability metrics AV, AC, AU, PR, S, and UI. Each captured attribute is processed by the attribute analysis corresponding to the exploitability metric. The temporary level of the exploitability metric is generated and compared with the level of the exploitability metric previously stored in the *Result*. If the temporary level of the exploitability metrics is greater than the level stored in *Result*, store the temporary level and other related information of the exploitability metric into *Result*, then go to Step 3. In addition, if the attributes contain read or write permission on the file, the attributes are selected and then go to Step 2.

Step 2: Deal with attributes related to impact metrics C, I, and A. The relevant path name of the file is extracted from the attribute selected in Step 1. It is compared with the path of the system sensitive files to get the levels of impact metrics C and I. If the attribute contains the read (write) permission, it is related to C (I). If the level of the impact metric is greater than the level of the impact metric previously stored in *Result*, the level and other information of impact metric override the previous information in *Result*. In addition, the evaluation method for impact metric A is similar to that for C and I. The only difference is that the level of A can be read directly from monitored attributes.

Step 3: Generate the vulnerability severity. The values of each base metric (i.e., exploitability metric and impact metric) are extracted from *Result*, and are used to generate the vulnerability severity. Finally, the vulnerability severity score and the information about assessment process are stored into *Result*.

4 Experiments and Results

In the experiments, we select the attributes to depict the characteristics related to each base metric according to the established model, and monitor the programs by using the dynamic instrumentation tool Pin [14]. We use the API given by Pin to instrument the exploits/PoCs and monitor the attributes for Linux. Since the experiments are based on the Linux and the tool Pin requires a binary executable file, the exploits/PoCs we choose are limited to those which are written by C or C++ and can run on Linux.

We divide the base metrics into three types according to the nature of their attributes. The first type contains base metrics AV, AC, AU, PR, UI, C, and I whose attributes are mainly related to system calls, the paths of sensitive files, and so on. These metrics can be evaluated by capturing related system calls and their parameters. The attributes related to this type of base metrics are shown in Table 2. The second type involves the base metric A which needs to monitor the status of system continuously. The specific attributes of A is shown in Table 3. The third type involves the base metric S, which change can be determined by the change of authority domain. It can be obtained during the attack process directly. Therefore, we do not provide the specific attributes for S.

In practice, human experts who assess the vulnerability severity can get the exploits/PoCs from the vulnerability discoverers for the first time to carry out the vulnerability assessment. In our experiments, we collect vulnerabilities and their corresponding exploits/PoCs from the public website EDB [2] to reproduce the attacks for the vulnerabilities. Figure 2 shows the number of exploits (written by C or C++) for Linux kernel, Apache service, and FTP service published by EDB from 1999 to 2016. We select vulnerabilities from Linux kernel, FTP service, and Apache service because they have more exploits and most of these software are open source. From Fig. 2, we can see that in recent years, most of the exploits are for Linux kernel and few exploits are for Apache service and FTP service. In the NVD, almost all vulnerabilities provide CVSS v2, and many of them do not provide CVSS v3 at the time of writing. We select the version(s) of CVSS for each vulnerability as the NVD does in our experiments.

Our experiments involve 98 vulnerabilities from the above three products (i.e., 74 Linux kernel vulnerabilities, 8 FTP service vulnerabilities, and 16 Apache service vulnerabilities) whose exploits provided in the EDB can be used to successfully reproduce the attacks. We adopt AutoCVSS to assess their severity. The result is that only two vulnerability severity scores assessed by AutoCVSS are obviously different from those in the NVD for CVSS v2, as shown in Fig. 3. One

Table 2. Attributes for the first type of base metrics

Base metric	Attributes
AV	$socket, connect, hub\_nport\_nconnect\_nchange, usb\_nprobe\_ninterface$
AC	$setregid, umount, mkdir, umount2, socketcall,$ $open, link, symlink, setresuid, setreuid, setuid, setfsuid,$ $setgroups, setgid, setfsgid, setfsgid, setresgid, chmod, fchmod, chown,$ $fchown, lchow$
AU	$execle, execl$
PR	$chmod, fchmod, chown, fchown, lchown, setresuid, setreuid, setuid,$ $setfsuid, setgid, setfsgid, setfsgid, setresgid, setregid$
UI	$clone, execute, fork$
C and I	$/bin, /boot, /dev, /etc, /lib, /proc, /root, /srv, /sys$

Table 3. Attributes for base metric A

Attribute	Operations
CPU utilization	Access the file */proc/stat* to extract the relevant data of CPU
Disk utilization	Use the shell command to the disk and extract the relevant data
Network bandwidth utilization	Use the shell command to obtain the network bandwidth usage and extract the relevant data
Memory utilization	Access the file */proc/meminfo* to extract the relevant data of memory

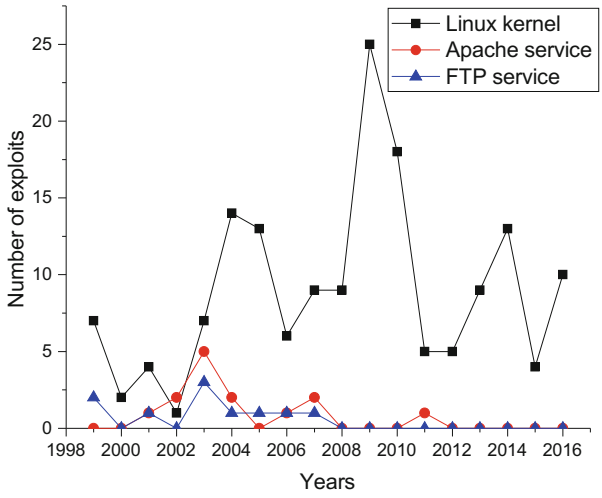

Fig. 2. The number of exploits (written by C or C++) for Linux kernel, Apache service, and FTP service published by EDB from 1999 to 2016

deviation is caused by the inaccurate level of AU. Specifically, the authentication may be occurred before the exploit. For example, the attacker may log into the system before exploiting the vulnerability. Therefore, the authentication information which the exploit does not contain cannot be captured. Another deviation is caused by the incomplete attributes we consider during the implementation, which will be improved in our future work.

In what follows, we select two vulnerabilities (CVE-2016-5195 for Linux kernel and CVE-2011-3192 for Apache HTTP Server) with three exploits from EDB (EDB-ID 40611 and 40847 for CVE-2016-5195, and EDB-ID 18221 for CVE-2011-3192) to illustrate the specific process of AutoCVSS.

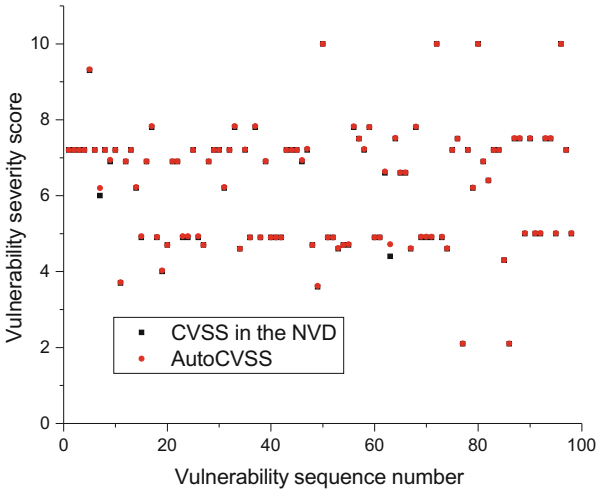

Fig. 3. Comparison between the CVSS v2 scores in the NVD and the vulnerability severity scores obtained by AutoCVSS for 98 vulnerabilities

4.1 CVE-2016-5195

CVE-2016-5195, also known as "Dirty COW", is caused by the race condition in Linux kernel 2.x through 4.x before 4.8.3. It allows local users to gain privileges by leveraging incorrect handling of a *copy-on-write* (COW) feature to write to a read-only memory mapping [3]. There are two exploits (EDB-ID 40611 and 40847). The first exploit changes the permission of a file, and the second exploit can get the root privilege.

As shown in Fig. 4, the exploit EDB-ID 40611 creates two threads: *madvise* and *procselfmemThread*. Thread *madvise* is responsible for the memory page allocation, and thread *procselfmemThread* mainly tries to write the data to memory. The exploitation process is as follows. For the first time, the write operation could cause page fault, then Linux deals with this page fault. For the second time, Linux deals with the write permission error by removing the write permission requirements and calling *madvise* to overwrite the previous cow pages. For the third time, Linux finds the page fault, but this page has no FOLL_nWRITE permission requirements, then the memory page mapped can be directly accessed, leading to permission issues.

The exploit EDB-ID 40847 is to gain the root privilege. The exploitation process mainly has three steps. First, "*bin/bas*" information is written to the file *tmp/.pwn*. Second, the permission of *pwn* file is modified, so that this file has the executable permission. Third, the shell in */etc/passwd* is modified to point to "*root*:*x*:0:0:*root*:/*root*:/*tmp/.pwn*", that is, point to *tmp/.pwn* executable file. At last, the shell can be run under the authority of root.

In both attack processes, AutoCVSS does not catch system call *connect*, which indicates that it is a local attack. The non-sensitive file *pwn* is created,

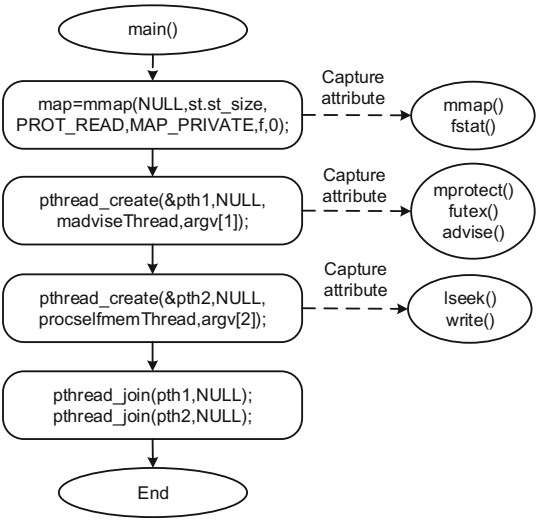

Fig. 4. The attack flow of CVE-2016-5195 (EDB-ID 40611) and the main attributes captured by AutoCVSS

and its permission is modified, thus the level of AC and PR is low. There is no interaction with Linux, no system authentication and software authentication, and the authorization scope is unchanged, which indicates the level of UI is none, the level of AU is none, and the level of S is unchanged. For the first exploit, only the permissions and contents of a non-sensitive file are changed, and the system can run properly without impact. Therefore, the level of C and I is low and the level of A is none. While for the second exploit, the root privilege is obtained after the attack, thus the level of C, I, and A is high. As the impact caused by the second exploit is more serious, the vulnerability severity assessed by the second exploit is selected as the severity of this vulnerability. As NVD provides the vulnerability with both CVSS v2 score and CVSS v3 score, we list base metrics for both CVSS v2 and CVSS v3 in Table 4. The result obtained by AutoCVSS is 7.2 for CVSS v2 and 7.8 for CVSS v3, which are the same as the results in the NVD.

4.2 CVE-2011-3192

CVE-2011-3192 is a vulnerability in the Apache HTTP Server 1.3.x, 2.0.x through 2.0.64, and 2.2.x through 2.2.19. It allows an attacker to cause a denial of service attack via a Range header that expresses multiple overlapping ranges [3].

The attack flow in function *thread_nstart*() is as follows. The request packet is send to Apache HTTP Server continuously by function *write*(). The HTTP header information in the request packet contains the range option, which defines how to request fragmented resource files. If a large number of overlapping range specification commands are set in the range option, Apache HTTP Server will

Table 4. Levels of base metrics obtained by AutoCVSS

Base metric	CVE-2016-5195		CVE-2011-3192	
	Level	Assessment basis	Level	Assessment basis
AV	Local	Default level	Network	IP is not on the same network segment as the IP of server
AC	Low	Write some information, modify file permissions	Low	Send a message
AU	None	Relevant attributes are not captured	None	Relevant attributes are not captured
PR	Low	Need to user permission	-	-
UI	None	Relevant attributes are not captured	-	-
S	Unchanged	The vulnerable component and the impacted component are the same	-	-
C	High	Get root privilege	None	Capture *read*(), but there is no file path matched in *read*()
I	High	Get root privilege	None	Capture *write*(), but there is no file path matched in *write*()
A	High	Get root privilege	High	Complete memory is exhausted

consume a lot of memory and CPU resources to construct the response data, causing the operating system to run out of resources.

In this process, AutoCVSS can intercept the main system calls *socket*, *connect*, and *write*. Since *connect* can be successfully connected, it indicates that a remote connection is made. In the server system, AutoCVSS monitors the network utilization, disk utilization, and CPU utilization, which basically remain unchanged. But the memory utilization continues to increase, basically more than 80%. From this perspective, we can see that when the memory is low, it would cause the system to deny service. At last, the vulnerability severity score obtained by AutoCVSS is 7.8, which is the same as the CVSS v2 in the NVD. Table 4 shows the level of each base metric for AutoCVSS.

5 Related Work

AutoCVSS is used to assess the vulnerability severity based on CVSS and attack process. In what follows, we review the prior works from two aspects: CVSS and attack process.

Prior Work Related to CVSS. CVSS is proposed by NIAC to solve the inconsistency and incompatibility problems caused by various security assessment systems. There are many studies about CVSS. Some studies [15,20] pointed out that the factors considered by CVSS were not comprehensive enough and the scores obtained could not truly reflect the vulnerability severity. Younis et al. [20] proposed to use the attack surface to increase the accuracy of assessment. For the assessment problems of CVSS, some approaches were presented to improve the CVSS [7,9,16]. In addition, there are also some approaches to the prediction or assessment of vulnerability [4,5,10,13]. For example, Khazaei e al. [13] proposed an automated approach to assess vulnerabilities. Their vulnerability features were generated from the vulnerability description information. However, the above studies about CVSS are basically static approaches. They do not involve the attack process which has more valuable information for vulnerability severity assessment.

Prior Work Related to Attack Process. Many researches used the attack graph to evaluate or predict the level of network security. Huang et al. [12] extracted the characteristics from the attack graph. These characteristics were combined with CVSS to statically evaluate the network security. However, our characteristics are based on attack process and our approach to vulnerability severity assessment is dynamic, which can more accurately obtain the attack data. Hu et al. [11] provided more information about the future of network attack behaviors by dynamic Bayesian attack graph. The information is limited to network attack behaviors, and the evaluation method does not apply to the severity assessment of vulnerabilities without network attack. Besides, some attack models [6,8] were also used to predict the attack behaviors.

The previous studies show that there is few concern about the combination of CVSS and the attack process to dynamically assess the vulnerability severity. Our goal is to use the attack process to make the assessment process of CVSS automatically and objectively, and the experimental results show the effectiveness of AutoCVSS.

6 Conclusion

We present AutoCVSS, an approach for automatic assessment of vulnerability severity based on attack process. It leverages the characteristics and rules we define to model the CVSS base metrics, and assesses the vulnerability severity automatically and objectively by capturing the attributes related to the characteristics during the attack process. Our results show that the vulnerability severity scores automatically obtained by AutoCVSS are basically in accordance with those assessed manually by security experts in the NVD, which verifies the effectiveness of AutoCVSS. For future research, we will improve the characteristics and rules of AutoCVSS for more comprehensive vulnerability severity assessment and strive to assess the vulnerability severity through multiple exploits/PoCs more effectively.

Acknowledgments. This paper is supported by the National Key Research & Development (R&D) Plan of China under grant No. 2017YFB0802205, the National Science Foundation of China under grant No. 61672249, and the Shenzhen Fundamental Research Program under grant No. JCYJ20170413114215614.

References

1. Common Vulnerability Scoring System. https://www.first.org/cvss/
2. Exploit database. https://www.exploit-db.com/
3. National Vulnerability Database. https://nvd.nist.gov/
4. Allodi, L., Banescu, S., Femmer, H., Beckers, K.: Identifying relevant information cues for vulnerability assessment using CVSS. In: Proceedings of the 8th ACM Conference on Data and Application Security and Privacy (CODASPY), pp. 119–126. ACM (2018)
5. Allodi, L., Biagioni, S., Crispo, B., Labunets, K., Massacci, F., Santos, W.: Estimating the assessment difficulty of CVSS environmental metrics: an experiment. In: Dang, T.K., Wagner, R., Küng, J., Thoai, N., Takizawa, M., Neuhold, E.J. (eds.) FDSE 2017. LNCS, vol. 10646, pp. 23–39. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70004-5_2
6. Almasizadeh, J., Azgomi, M.A.: A stochastic model of attack process for the evaluation of security metrics. Comput. Netw. **57**(10), 2159–2180 (2013)
7. Cheng, P., Wang, L., Jajodia, S., Singhal, A.: Aggregating CVSS base scores for semantics-rich network security metrics. In: Proceedings of the 31st Symposium on Reliable Distributed Systems (SRDS), pp. 31–40. IEEE (2012)
8. Del Valle, S., Hethcote, H., Hyman, J.M., Castillo-Chavez, C.: Effects of behavioral changes in a smallpox attack model. Math. Biosci. **195**(2), 228–251 (2005)
9. Gallon, L.: On the impact of environmental metrics on CVSS scores. In: Proceedings of the 2nd International Conference on Social Computing (SocialCom), pp. 987–992. IEEE (2010)
10. Ghani, H., Luna, J., Khelil, A., Alkadri, N., Suri, N.: Predictive vulnerability scoring in the context of insufficient information availability. In: Proceedings of 2013 International Conference on Risks and Security of Internet and Systems (CRiSIS), pp. 1–8. IEEE (2013)
11. Hu, H., Zhang, H., Liu, Y., Wang, Y.: Quantitative method for network security situation based on attack prediction. Secur. Commun. Netw. **2017**, 1–19 (2017)
12. Huang, H., Zhao, F., Ye, M.: Estimate the influential level of vulnerability instance based on hybrid ranking for dynamic network attacking scenarios. In: Proceedings of the 10th International Conference on Information Sciences Signal Processing and their Applications (ISSPA), pp. 586–589. IEEE (2010)
13. Khazaei, A., Ghasemzadeh, M., Derhami, V.: An automatic method for CVSS score prediction using vulnerabilities description. J. Intell. Fuzzy Syst. **30**(1), 89–96 (2016)
14. Luk, C., et al.: Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of Conference on Programming Language Design and Implementation, pp. 190–200. ACM (2005)
15. Luo, J., Lo, K., Qu, H.: A software vulnerability rating approach based on the vulnerability database. J. Appl. Math. **2014**, 932397:1–932397:9 (2014)
16. Ross, D.M., Wollaber, A.B., Trepagnier, P.C.: Latent feature vulnerability ranking of CVSS vectors. In: Proceedings of the Summer Simulation Multi-Conference, pp. 19:1–19:12. Society for Computer Simulation International (2017)

17. Spanos, G., Sioziou, A., Angelis, L.: WIVSS: a new methodology for scoring information systems vulnerabilities. In: Proceedings of the 17th Panhellenic Conference on Informatics, pp. 83–90. ACM (2013)
18. Tripathi, A., Singh, U.K.: Estimating risk levels for vulnerability categories using CVSS. Int. J. Internet Technol. Secured Trans. **4**(4), 272–289 (2012)
19. Younis, A.A., Malaiya, Y.K.: Comparing and evaluating CVSS base metrics and Microsoft rating system. In: Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 252–261. IEEE (2015)
20. Younis, A.A., Malaiya, Y.K., Ray, I.: Using attack surface entry points and reachability analysis to assess the risk of software vulnerability exploitability. In: Proceedings of the 15th International Symposium on High-Assurance Systems Engineering (HASE), pp. 1–8. IEEE (2014)
21. Younis, A.A., Malaiya, Y.K., Ray, I.: Assessing vulnerability exploitability risk using software properties. Software Qual. J. **24**(1), 159–202 (2016)
22. Younis, A., Malaiya, Y.K., Ray, I.: Evaluating CVSS base score using vulnerability rewards programs. In: Hoepman, J.-H., Katzenbeisser, S. (eds.) SEC 2016. IAICT, vol. 471, pp. 62–75. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33630-5_5