International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:14, No:1, 2020

SVID: Structured Vulnerability Intelligence for Building Deliberated Vulnerable Environment

Wenqing Fan, Yixuan Cheng, Wei Huang

***Abstract*—**The diversity and complexity of modern IT systems make it almost impossible for internal teams to find vulnerabilities in all software before the software is officially released. The emergence of threat intelligence and vulnerability reporting policy has greatly reduced the burden on software vendors and organizations to find vulnerabilities. However, to prove the existence of the reported vulnerability, it is necessary but difficult for security incident response team to build a deliberated vulnerable environment from the vulnerability report with limited and incomplete information. This paper presents a structured, standardized, machine-oriented vulnerability intelligence format, that can be used to automate the orchestration of Deliberated Vulnerable Environment (DVE). This paper highlights the important role of software configuration and proof of vulnerable specifications in vulnerability intelligence, and proposes a triad model, which is called DIR (Dependency Configuration, Installation Configuration, Runtime Configuration), to define software configuration. Finally, this paper has also implemented a prototype system to demonstrate that the orchestration of DVE can be automated with the intelligence.

***Keywords*—**DIR Triad Model, DVE, vulnerability intelligence, vulnerability recurrence.

I. Introduction

IN the past few years, the number and complexity of IT systems has led to a steady increase in the number of vulnerabilities in these systems. According to National Vulnerability Database (NVD) official statistics, the number of vulnerabilities received by NVD in 2016, 2017, and 2018 were 6447, 14645, and 16516, respectively [1]. These vulnerabilities that continue to occur every year have had a serious impact on individuals, companies, and organizations. In September 2018, a Facebook vulnerability allowed hackers to take over user accounts directly, and the vulnerability caused at least 50 million user account information to be leaked [2]. In 2017, WannaCry ransomware used unrepaired vulnerabilities in less than 24 hours, causing more than 300,000 computers in more than 150 countries around the world to stop working [3].

The ubiquity of vulnerabilities has led more and more software vendors and organizations use crowdsourcing to exploit vulnerabilities. Anyone on the Internet can find a vulnerability and report this issue. The US Department of Defense launched the first federal vulnerability award program "Hack the Pentagon" in 2016. The project has received nearly 3,000 vulnerability reports from more than 600 security researchers worldwide, of which more than 100 are considered high-risk vulnerabilities, including remote code execution and methods to bypass the DoD website authentication [4]. Companies such as Google and Microsoft are spending millions of dollars on their "bug bounty" projects to reward vulnerability reporters [5], [6]. Some submitted vulnerability reporters can obtain a Common Vulnerabilities and Exposures (CVE) number after the vulnerability is approved. As of March 2019, there have been more than 110,000 vulnerabilities on the CVE website [24]. So, vulnerability information can effectively help software vendors and organizations reduce the burden of find a vulnerability.

Although there are so many vulnerabilities reported, both software vendors and third-party vulnerability platforms need to spend a lot of manpower and resources to verify the accuracy of these vulnerability reports, and then to fix these vulnerabilities. According to researchers' tests, it takes 5 hours to verify the existence of a vulnerability on average based on vulnerability reports for vulnerabilities on CVE. According to vulnerability reports for vulnerabilities on non-CVE, it takes 3 hours to verify the existence of a vulnerability on average, and personal vulnerability reports from popular security forums have a minimum of 4.5% success rate [7]. Such a low success rate will have some serious consequences. For example, a Facebook user once found a vulnerability that allowed an attacker to post a message to anyone's timeline. However, due to “lack of sufficient detail to reproduce the vulnerability”, Facebook engineers ignored the initial report until the Facebook CEO’s schedule was hacked [8]. If a security researcher can automatically construct a DVE with a target vulnerability based on vulnerability information, then it can effectively increase the success rate of vulnerability recurrence, greatly reduce resources such as manpower and material resources, and reduce the risk of vulnerability report review and reduce the further harm from reported vulnerabilities. DVE is a standard proposed to facilitate the "security emergency response team" to build a vulnerability recurring environment, because for the author of the software, the task of setting up a recurring environment is a breeze: there are ready-made development and testing environments in the local area. The code repository can be built back to the historical version to build the specified version of the software at any time. Only for third-party teams, setting up a vulnerability recurring environment is indeed a dirty work that cannot be ignored and cannot be avoided.

Vulnerability intelligence, as a more targeted category of threat intelligence, is closely related to the research of threat intelligence. In the past 10 years, researchers have spent a lot of time working on threat intelligence. Some researchers are

Yixuan Cheng, Wei Huang, and Wenqing Fan are with the School of Computer Science and Cybersecurity, Communication University of China. (e-mail: cucfitz@gmail.com, huangwei.me@cuc.edu.cn, fanwenqing@cuc.edu.cn).

working on the format of threat intelligence and the process of incident reporting. Menges et al. [9] and Asgarli et al. [10] conducted a comparative analysis of some of the most important event reporting formats, including STIX and IODEF, and designed the process of incident reporting. Their works have a good reference in the field of data exchange, but do not have a clearer description of the important factors in the security incident - the vulnerability and its recurring process.

Other researchers focus on comparing the differences between different vulnerability reports. Dong et al. [11] compared the quality and consistency of information between CVE and NVD vulnerability reports and conducted large-scale measurements. Their results indicate that inconsistencies between the two are common. The community needs to systematically correct inaccurate claims in the vulnerability report. Their research confirms that not only third-party vulnerability reports, but also some official vulnerability reports have information inaccuracies, such as there are many errors in the affected software version information; this inaccurate information also has an impact on the recurrence of vulnerabilities. So, if we can automate the generation of a vulnerability recurring environment based on the vulnerability report, and then determine whether the environment has the vulnerability, then we can efficiently and conveniently verify the authenticity and accuracy of the vulnerability report.

Based on the above background, the two problems to be solved in this paper are: 1. What information is used to construct the DVE environment in vulnerability information is critical and mandatory information; 2. How to design a new standardized, structured, machine-oriented vulnerability intelligence standard make it easier to build DVEs.

The rest of this article is organized as follows. Section II describes the work parallel to our vulnerability intelligence standard SVID. Section III explains the design of our vulnerability intelligence. Section IV describes the design of the prototype system. Section V describes the implementation of prototype system. Section VI uses a representative vulnerability as an example to prove the validity of SVID. Section VII concludes the paper and proposes future research directions.

II. Related Work

Some researchers focus on classifying and comparing threat intelligence. Tounsi and Rais [12] differentiated and classified existing threat intelligence types and provided an intelligence-sharing strategy based on trust and anonymity to help organizations eliminate the risk of business disclosure in the process of sharing threat intelligence. According to the traffic light agreement [13], threat intelligence should be shared when sharing. Therefore, our vulnerability information allows detailed vulnerability reconstruction environment construction information and PoC; the premise should be "trust-based" intelligence sharing strategy. That is, when a vulnerability reporter reports a vulnerability to a vendor affected by a vulnerability, the trusted vulnerability researchers share the vulnerability research results with each other. The reason for this is to avoid the misuse of information in DVE and vulnerability intelligence to reduce the time it takes to expose vulnerabilities in cyber-attacks. The existing threat intelligence, vulnerability announcement information, and third-party vulnerability research reports have excessive disclosure of details of the vulnerability, but the more common situation is the lack of key information for the recurrence of the vulnerability. The reasonable reason for missing this part of the information is to avoid the details of the abuse of the vulnerability.

Steinberger et al. [14] analyzed different exchange standards and introduced several comparison criteria. Their work provides valuable advice for comparing reporting formats, although most of these standards are now almost non-relevant and work does not specifically focus on reporting formats. Mavroeidis and Bromander [15] introduced the CTI model to enable cyber defenders to explore their threat intelligence capabilities. They also used their models to analyze and evaluate several existing threat intelligence classification methods, intelligence sharing standards, and ontology related to threat intelligence. They also suggested that knowledge from domain expertise should be collected in a structured way and presented in the ontology of threat intelligence.

Some researchers explore some of the basic elements of threat intelligence. Cichonski et al. [16] have done a great deal of work in describing events and their components, and have greatly assisted in the description of events in threat intelligence. However, the acquisition of new vulnerability information depends on the briefings of some third-party organizations, website publishing content, mailing lists, etc., and the information about the vulnerability information obtained is less related to the recurrence of the vulnerability. Therefore, organizational security researchers who obtain threat intelligence cannot quickly reproduce vulnerabilities based on threat intelligence, thereby patching vulnerabilities and preventing further losses.

Other researchers jumped out of the vulnerability intelligence itself and innovated from the perspective of vulnerability recurrence. Mu Mu et al. [8] focused on the recurrence of vulnerabilities and tried to select vulnerability reports from the perspective of loophole recurrence. For the first time, they manually reproduced a large number of real vulnerabilities to further analyze the actual problems encountered in the process of recurring vulnerabilities. Based on their research results, they suggest that if the software name and affected version are already defined, the software installation part can be automated; if the Proof of Vulnerable (PoV) script, the vulnerability trigger method, and the vulnerability verification method are given out, the vulnerability reproducer can automate the recurrence of the vulnerability, in which the PoV script can reuse the existing Proof of Concept (PoC) script. The work of Mu et al. provided invaluable advice for the study of vulnerability recurrence, but did not provide a concrete and feasible solution to achieve automation.

III. Vulnerability Intelligence

In this section, we will introduce an overview of the SVID

methodology. We will then understand the overall intelligence structure of SVID and detail the design of each field and the basis for our selection of these fields.

A. Methodology Overview

Our goal is to design a standardized, structured, machine-oriented vulnerability intelligence standard that makes it possible to automate the development of a DVE. But there are many challenges in this process.

The first challenge is how to determine which fields are necessary to automate the building of DVE intelligence. Vulnerabilities based on existing vulnerability intelligence are almost entirely manual, and in order to reproduce vulnerability, it may be necessary to integrate information from multiple vulnerability reporting sources and some are not valid. It is very cumbersome and complicated to determine the information necessary to reproduce the vulnerability in this complicated vulnerability information.

The second challenge is that successful vulnerability recurrence may also depend on the knowledge and skills of security analysts [7]. The variety of vulnerabilities has led to a diversity of vulnerabilities in recurring environments, and the types of vulnerabilities required for vulnerabilities to replicate are also different. So, for different types of vulnerabilities, many domain experts may be needed to analyze them together.

Based on the above challenges, it is difficult for our work to achieve depth and breadth at the same time. So, we decided to prioritize the depth issue while maintaining a reasonable scale to achieve scalable results. More specifically, we have chosen the vulnerability in the software that satisfies the twelve-factor application [17] as our research object (the application software mentioned in the subsequent chapters of this article refers to the software that satisfies the application of the twelve-factor). So, we can prepare a special expert group for components in this field to conduct our research. We combine the information needed in the process of manually recreating the vulnerability, the existing information in the existing vulnerability report, and the configuration software used to define the network node in some professional software that sets up the network environment (such as the applications in GNS3 [18]). The most critical information used to automate the construction of the DVE environment was filtered out to design vulnerability information SVID. In addition, we have designed a prototype system to verify the effectiveness of SVID. Finally, we selected a representative set of software and verified the high-risk vulnerabilities in recent years.

B. Intelligence Design

For the vulnerability in the software that satisfies the twelve-factor application, we define the vulnerability information as shown in Table I.

C. Intelligence Element

The following is a detailed explanation of the fields in the vulnerability information and explains why we chose this information.

TABLE I
VULNERABILITY INTELLIGENCE FORMAT

Classification	Field	Description	Options
Basic Element	*os-s*	Software operating system	Required
	os-e	Exploit operating system	Required
Software Information	*sn*	Software name	Required
	sv	Software version	Required
Software Configuration	*sdc*	Software dependency configuration	Required
	sic	Software installation configuration	Required
	src	Software runtime configuration	Required
PoV	*edc*	Exploit script dependency configuration	Required
	eic	Exploit script installation configuration	Required
	erc	Exploit script runtime configuration	Required
Vulnerability Verification	*vvm*	Vulnerability verification method	Required
Vulnerability Description	*vd1*	Vulnerability database	Optional
	id1	Vulnerability identification	Optional
	…	……	…
	vt	Vulnerability type	Optional
	em	Exploit mode	Optional
	vtm	Vulnerability triggering method	Optional
	st	Software type	Optional
	ve	Vulnerability effect	Optional
	p	Port	Optional
	vde	Vulnerability description	Optional
	sve	Software vendor	Optional
	svu	Software vendor URL	Optional
	ex	Extended fields	Optional

1) Basic Element

Software Operating system: It is the operating system on which the affected vulnerability of the vulnerability is dependent. An application depends on its operating system, and operating system information is also the basic information used to reproduce the vulnerability environment. The format specification of the operating system can refer to the format specification of CPE [19]. In CPE, a typical operating system information description is cpe:2.3:o:debian:debian_linux:7.0:*:*:*:*:*:*:*.

Exploit Operating system: It is the operating system on which PoV depends. Exploit host and target software may not be on an operating system, so we need to additionally define the operating system on which the exploit depends to complete the system configuration of the exploit host.

2) Software Information

Software name: The software name is used to declare the name of the software affected by the vulnerability. Every vulnerability depends on a particular piece of software. So if we want to build a vulnerability recurring environment, we need to build the affected software environment first. Affected software names can help identify software quickly.

Software version: The software version is used to declare the version information of the software affected by the vulnerability. There is a scope for each version of the software affected by the vulnerability. Vulnerabilities in the A version of the software may not exist in the B version of the same software. So, we need precise, unambiguous software version definitions to uniquely identify different versions of the software. For the

declaration of version information, refer to the declaration specification of the semantic version 2.0.0 [20]. In addition, one can also refer to the format specification of CPE.

3) Software Configuration

In the process of researching software configuration, PoV and related description information, we proposed the DIR triad model to describe software configuration related information and PoV related information. Software dependency configuration is used to describe the dependency environment of the software and how it is configured. The software installation configuration is used to describe how the software itself is installed and configured. The software running configuration is used to describe the relevant configuration of the software at runtime, and defines the operating rules of the software configuration script and PoV script. Within our knowledge, for software that meets the twelve-factor application, almost all software configuration script information used to build a vulnerability recurring environment and PoV information for verification purposes can be described using the DIR triad model. In the following we will introduce how the DIR triad model is applied in software configuration, PoV and its associated descriptions, and why we chose them.

Software dependency configuration: It refers to the list of environment dependencies required to define all dependencies in the software configuration script, corresponding to the Dependency Configuration in the DIR triad model. Software configuration scripts often use some third-party function modules and tools when implementing their configuration functions. If these function modules and tools are missing, the configuration scripts will not be executed as expected. In addition, the environment required for the software configuration script itself to run needs to be met, such as the environment in which the script's development language is configured. So, software configuration dependencies are needed to precisely define all the modules, tools, development languages, etc., and their version information.

Software installation configuration: It refers to the software configuration script used to install and configure the target software to enable it to have a target vulnerability and its related configuration, corresponding to the Installation Configuration in the DIR triad model. Software installation configuration is a very important part of the vulnerability recurring work, but it is often overlooked. According to researchers, more than 87% of vulnerability reports do not include information such as configuration and options for software installation [7]. A vulnerability may exist in different versions of the software, but even if the software is installed for one of the affected versions, the successful installation does not mean that the vulnerability must exist in the software and may require additional configuration. Let's take the CVE-2019-6340 vulnerability as an example. The software version affected by this vulnerability includes drupal:8.5.0 version, but the vulnerability cannot be triggered directly after installing drupal: 8.5.0, because one of the trigger conditions for this vulnerability is to use the RESTful web service module (rest module) enabled by the drupal 8 core site and allow PATCH or POST requests. Therefore, additional software installation configuration is required for the installed drupal software to successfully trigger the vulnerability. Other trigger conditions for this vulnerability also require additional software configuration. So, in order to automate the construction of the DVE environment, software installation configuration is necessary.

Software runtime configuration: It refers to the way required to run the software configuration script to define the execution method of the software configuration script, corresponding to the Runtime Configuration in the DIR triad model. There may be many development languages for software configuration scripts, or they may be defined by an ansible playbook or an yml file in CI/CD. Different implementation methods of different types of scripts depend on the expert's experience. So, if we need to automate the configuration of DVE for the machine, we need the software running configuration to define how the software configuration script works.

4) PoV

Exploit script dependency configuration: It refers to the list of environment dependencies required to define dependencies and their configuration in the exploit script, which corresponds to the Dependency Configuration in the DIR triad model. Exploiting scripts also rely on some third-party modules and tools when implementing their exploits. A typical tool is Metasploit [21]. So, without these dependent modules or tools, an exploit may fail and it is impossible to verify that the environment being built is a DVE. Similar to the software configuration, the environment required for the exploit script itself to run needs to be met, such as the development language environment of the configuration script. Therefore, exploit script dependency configuration are needed to precisely define all the modules, tools, development languages, etc. and their version information.

Exploit script installation configuration: It refers to the PoV code and related configuration that exploits the vulnerability to trigger a specific vulnerability for the target environment, corresponding to the Installation Configuration in the DIR triad model. An important step in automating the construction of DVE is to verify that the existing environment does have a target vulnerability, so automation of the exploit needs to be implemented. PoV is the most important part of the loopback process. If there is no PoV script to reproduce the loophole, it will take a lot of manpower and resources, and it needs the expertise of experts in the field. The worst part is that it takes extra time to manually analyze. So, if we want to automate the exploitation of the vulnerability, we need to provide PoV in the intelligence.

Exploit script runtime configuration: It is used to define how the exploit script is run, corresponding to the Runtime Configuration in the DIR triad model. There are many possible development languages for exploit scripts, and even a lot of exploit code can be done with a single bash command. So, if we need to automate the exploitation of the machine, we need the exploit script runtime configuration to define how the exploit

script works.

5) Vulnerability Verification

Vulnerability verification method: It is used to define whether the verification exploit script successfully triggered the specified vulnerability. In the exploit phase, although there are already PoV scripts to help us trigger the vulnerability, the trigger will not necessarily succeed, so we also need to check if the vulnerability really triggers the success. Taking the remote arbitrary code execution class vulnerability as an example, the PoV script can create a representative specific file in the specified directory after successfully triggering the vulnerability, and then monitor the read and write status of the file in the specified directory to verify whether the arbitrary code execution vulnerability is successfully triggered. Only if the verification is successful, prove that the vulnerability does exist and the exploit is successful, it can be said that the DVE is built. So, it is necessary to define how to verify the successful exploitation of the vulnerability.

6) Vulnerability Description

All of the above field information is mandatory. The optional field information is described below. Although the following field information is not directly related to the automatic construction of the DVE environment, it is very important for describing the information about the vulnerability and software and the application scenario for broadening the vulnerability information, so it is an optional field. The meaning of each field and its role are described in detail below.

Vulnerability database: It defines the dependent database for the vulnerability number. Because the vulnerability numbers of different vulnerability databases are different, a field is needed to clarify which database the vulnerability number belongs to prevent confusion. Therefore, there may be multiple vulnerabilities database names, and each vulnerability database name corresponds to an identifier.

Vulnerability identifier: It is an identifier that uniquely represents the vulnerability and is used to distinguish between different vulnerabilities and is a necessary field. Typical numberings include the CVE official CVE serial number and the CNNVD serial number of the China National Vulnerability Database of Information Security (CNNVD). The same vulnerability may have different IDs, but the ID of each vulnerability should be able to uniquely identify a vulnerability. For example, the CVE-2018-7600 vulnerability ID is different from the CNNVD-201803-1136 vulnerability ID, but they represent the same vulnerability. So, there can be more than one ID field, and each number corresponds to a vulnerability database name.

Vulnerability type: It defines the type of vulnerability. The vulnerability type information is the basic information necessary to describe the vulnerability, which can broaden the application scenarios of vulnerability information. For example, when the application is in the field of automated build range, it is possible to automate the generation of a series of DVE environments for range training for different vulnerabilities of the same type. The way to declare the vulnerability type can be found in CWE [22].

Exploit mode. It refers to the attack mode of an attacker when exploiting the vulnerability. Vulnerability analysts, testers can use this information to increase understanding of attack behavior and enhance defense capabilities. This field can broaden the application scenarios of vulnerability intelligence. For example, when the application is in the automated build range field, a series of DVE environments can be automatically generated for range training for different vulnerabilities of the same attack mode. The way to declare the exploit pattern can be found in CAPEC [23].

Vulnerability triggering method: It is used to describe the method that triggered the vulnerability and is the basic information necessary to describe the vulnerability information. The vulnerability analyst, the tester can choose the solution to reproduce the vulnerability based on the vulnerability triggering method.

Software type. It refers to the subordinate type of the vulnerability affecting software. Software types can help people understand the impact of software business logic and vulnerabilities. In addition, software types can link different software of the same class, allowing for correlations between vulnerabilities.

Vulnerability effect: It refers to the effect that can be exploited by the vulnerability. For example, a vulnerability can be used to implement remote arbitrary code execution. The vulnerability effect can help understand the hazards of the vulnerability, help determine the vulnerability verification method, and can also be applied to automate the construction of the target field to generate the same vulnerability in the target environment.

Port: A vulnerability-aware port is the listening port of the software affected by the vulnerability. According to the twelve-factor application, application software that satisfies the twelve-factor application can provide services through port binding. Therefore, by giving the port information, it can help determine the software listening port, and then apply it when building the DVE environment. A typical application is honeypot technology, which can help security practitioners automate the construction of honeypots and bind the software's default port to increase the authenticity of honeypots for intrusion fraud. Because the listening port of the software is mostly only on a certain port, but it can also be specified by humans, this information is not mandatory and optional.

Vulnerability description: Vulnerability description information is the basic information necessary to describe a vulnerability and is used to describe the entire vulnerability and its affected software, environment-dependent information, etc. in a natural language. And it can help people better understand the vulnerability.

Software vendor: The affected software vendor refers to the application from GNS3, which is used to identify the affected software vendor, so that you can consult the relevant information of the manufacturer when you encounter related problems.

Software vendor URL: The affected vendors' URLs are referenced from the GANS3's appliances, which are used to

give the software vendor's official website to prevent vendor information acquisition errors due to factors such as software duplicate names and keyword conflicts.

Extended fields: Extended fields are reserved to describe other unknown information, making vulnerability intelligence scalable.

IV. Prototype Design

The prototype system is designed as shown in Fig. 1, which includes two types of roles, Manager and Worker. The workers include Porter, Builder, Configurator, Exploiter, and Checker.

The flowchart of the prototype system is shown in Fig. 2. After receiving the SVID, the manager will parse the SVID and extract the useful information to distribute to other workers. The manager distributes (id, os-s, os-e, sn, sv) to the carrier and distributes (id, os-s, os-e, sn, sv, p) to the Builder, and distributes (sdc, Sic, src, p) to the Configurator, distributing (edc, eic, erc, p) to the attacker and distributing (vvm, em, p) to the verifier.

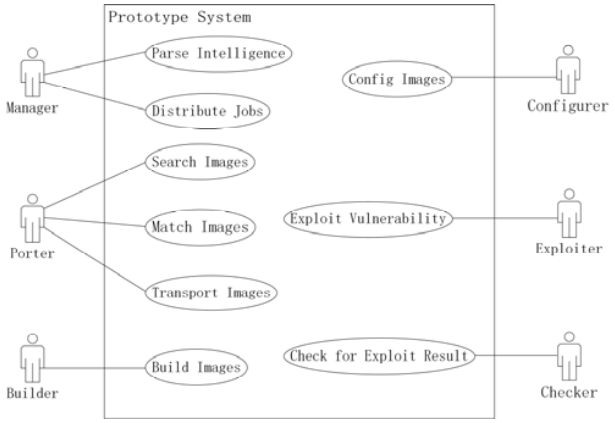

Fig. 1 Prototype system use case diagram

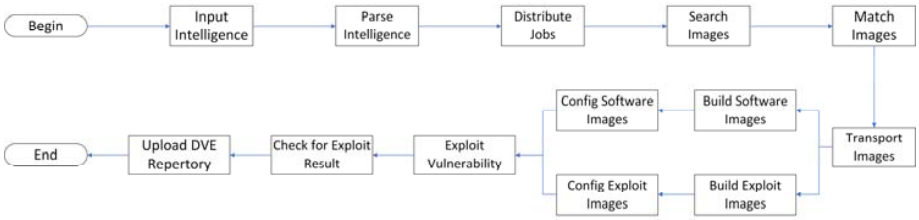

Fig. 2 Prototype system flow chart

The entire system process is similar to the factory's pipeline operations. After receiving the information distributed by the manager, the Porter searches for and matches the available original image, including the software image and the exploit image, from the mirror repository based on the information (id, os-s, os-e, sn, sv). The original image is then moved to the Builder. After the original image is obtained, the Builder configures the original image according to the information (id, os-s, os-e, sn, sv, p), and runs the image to become the running image. We will name it Image-R. After the previous step is completed, the Builder hands the Image-R to the Configurator. The Configurator configures the Image-R based on the (sdc, sic, src, p) information to make the target image has the target vulnerability. We will name the configured image as Image-RC. After the configuration is completed, the Configurator hands the Image-RC to the Exploiter, and the Exploiter completes the configuration of the exploit host according to the (edc, eic, erc, p) information, and uses the PoV to attack the target image. We named the image after the exploit was completed as Image-RCA. After that, the Checker obtains the Image-RCA from the Exploiter and completes the detection according to the (vvm, em, p) information, verifying that the previous exploit has been successful, proving that the image has become a DVE, and handing it over to Porter. Finally, the Porter moves the validated DVE to the DVE repertory to complete a complete production process.

V. Implementation

This section will introduce our specific implementation of the above prototype design.

The initial implementation of SVID was to use the JSON format as a ubiquitous, portable, and structured mechanism for later collaboration and improvement.

Our prototype system uses docker technology as the key technology to support the preservation, operation and exchange of images, docker hub as the image repository, harbor as the DVE repository, and Gitlab CI/CD pipelines to define and control Manager and Workers. Because software that satisfies the twelve element applications can provide services through ports, port mapping can be used to implement mirroring between builders, configurators, attackers, and verifiers.

GitLab CI/CD pipelines use runners to define different roles. The Porter searches for matching available software images from the docker hub with the mirrored TAG value based on the information (id, os-s, os-e, sn, sv). For example, you can search for available images in the sn:sv format. sn:sv itself is the official Docker-recommended image naming convention, so this design approach ensures that a large number of candidate based images are available. The original image that the Porter searched for was Docker Image, which needed to pull the Docker Image from the docker hub and hand it to the builder. The builder configures the original image based on the information (id, os-s, os-e, sn, sv, p) and runs it to generate the running Docker container Image-R. After above steps are

completed, the configurator installs and configures the Docker container Image-R according to the (sdc, sic, src, p) information. The attacker then uses the PoV script to complete the exploit process based on the (edc, eic, erc, p) information and creates a unique identifier to prove that the exploit process was successful. For example, for exploits of remote arbitrary code execution classes, PoV scripts can create specified unpredictable files in a specified directory. After the attack is completed, the verifier completes the verification based on the (vvm, em, p) information. For example, for exploits of remote arbitrary code execution classes, the verifier can monitor the read and write of all files in the specified directory. If the specified unpredictable file is found to be successful, the attacker's exploit process is successful, and the image Image-RCA has become DVE. Finally, the Porter packages the generated DVE into a Docker image and uploads it to the DVE repository harbor to complete a complete production process.

VI. A Typical Case

This section will take a specific vulnerability CVE-2018-7600 as an example to illustrate how SVID should be applied.

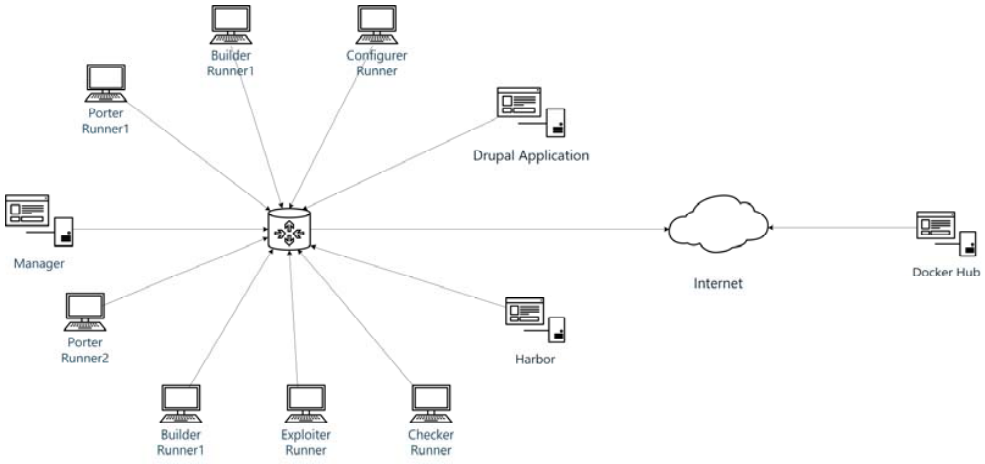

Fig. 3 Case network topology

The network topology diagram of the experiment is shown in Fig. 3. After obtaining the SVID of CVE-2018-7600, the Manager parses it. The vulnerability number is CVE-2018-7600, and the affected software name is Drupal, and a typical version number can be 8.5.0 after cleaning. Using this information, we can search for available images in the existing image repositories. Taking the public docker hub as an example, we can use the software name drupal to search for the official designated repository. There will be all different TAG values in the official repository for identifying different images. According to the software version information, we can search and match among all TAGs and select the appropriate image, such as the drupal:8.5.0 image. After finding the available image, we can use the Docker API to complete the image pull and container operation on the server that provides the docker service. Because the software that satisfies the twelve-element application can provide services through the port, the service can be mapped to the configurator runner and the attacker runner and the checker runner through port mapping. In addition, we need to complete the image acquisition and operation of the exploit runner according to os-e. The steps are similar to the way the drupal service is built.

The drupal service provided in the drupal:8.5.0 image is not installed and configured by default, so we need to complete the software configuration. On the configurator runner, the environment configuration is completed according to the software configuration dependency, and then the software configuration script is completed according to the software running dependency, so that the drupal can successfully complete the installation and the vulnerability configuration.

After the software configuration is complete, an exploit is required. On the exploit runner, we need to first configure the attack environment based on the exploit script dependency, and then use the PoV script to complete a complete exploit process based on the exploit script running dependency.

Once the exploit is complete, it needs to be verified. The exploit effect of the CVE-2018-7600 vulnerability is remote arbitrary code execution. Therefore, when exploiting the vulnerability, the PoV script can be used to create a specific name file in the specified directory to verify the effect when exploiting the completion vulnerability and performing arbitrary code execution. Monitoring of files is achieved by monitoring file changes in this directory. When verifying, we only need to check the monitoring log file to see if the specified name file is created to check the exploit result.

After the check is successful, the validated DVE container can be packaged as an image and uploaded to the harbor repository for persistent storage of the DVE environment.

VII. Conclusion

In this article, we summarize the information needed in the manual recurring vulnerability process, the existing information in the existing vulnerability report, and the configuration information used to define the network node in

some professional software that sets up the network environment. The most critical information for automating the construction of the DVE environment is used to define structured, standardized, machine-oriented vulnerability intelligence SVIDs and to expand new application scenarios: automate the construction of DVE. Compared with the threat information of the traditional STIX2 format, the SVID proposed in this paper focuses on the automated construction of the vulnerability recurrence environment, making the information more targeted. In addition, we highlight the important role of software pre-configuration and PoV files in vulnerability intelligence, and abstract the DIR triad model to describe software configuration and PoV usage specifications, reducing the technical requirements for vulnerability intelligence users. It enables the vulnerability intelligence users to focus on building the vulnerability recurring environment without spending a lot of time learning the basic principles of different types of vulnerabilities, effectively reducing the time cost of the vulnerability intelligence users. Finally, we also designed a prototype system to verify the validity of the SVID. And a detailed analysis of a specific vulnerability using SVID automation to build DVE process proved its feasibility.

In the following work, we will continue to study in the following aspects: 1. How to construct the vulnerability information SVID designed according to the existing threat information. 2. How to use the natural language processing method to obtain a variety of intelligence information Sources (such as blogs, vulnerability announcements, etc.) automatically identify, understand, and construct threat information/vulnerability information/PoV. 3. How to apply SVID to scenarios that require the construction of a target environment, such as honeypots, honeynet, ranges environment, etc. 4. How to implement a custom DVE environment combination.

ACKNOWLEDGMENT

This work was supported by the Fundamental Research Funds for the Central Universities (Grant No. 3132018XNG1815 and 3132018XNG1814).

REFERENCES

[1] NIST. National vulnerability database. https://nvd.nist.gov/. Retrieved: April 26, 2019.
[2] Lily Hay Newman. Everything we know about Facebook's massive security breach. https://www.wired.com/story/facebook-security-breach-50-million-accounts/. Retrieved: April 26, 2019.
[3] McAffee Corporation, McAfee Labs - Threat-Report, In: 2017, https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-sept-2017.pdf.
[4] DOD. Hacking the pentagon. https://www.usds.gov/report-to-congress/2017/fall/hack-the-pentagon/. Retrieved: April 26, 2019.
[5] Taylor Hatmaker. Google's bug bounty program pays out $3 million, mostly for Android and Chrome exploits. https://techcrunch.com/2017/01/31/googles-bug-bounty-2016/. Retrieved: April 26, 2019.
[6] Tom Warren. Microsoft will now pay up to $250,000 for Windows 10 security bugs. https://www.theverge.com/2017/7/26/16044842/microsoft-windows-bug-bounty-security-flaws-bugs-250k. Retrieved: April 26, 2019.
[7] Mu, Dongliang, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. "Understanding the reproducibility of crowd-reported security vulnerabilities." In 27th {USENIX} Security Symposium ({USENIX} Security 18), pp. 919-936. 2018.
[8] Steven Musil. Researcher posts Facebook bug report to Mark Zuckerberg's wall. https://www.cnet.com/news/researcher-posts-facebook-bug-report-to-mark-zuckerbergs-wall/. Retrieved: April 26, 2019.
[9] Menges, Florian, and Günther Pernul. "A comparative analysis of incident reporting formats." Computers & Security 73 (2018): 87-101.
[10] Asgarli, Elchin, and Eric Burger. "Semantic ontologies for cyber threat sharing standards." In 2016 IEEE Symposium on Technologies for Homeland Security (HST), pp. 1-6. IEEE, 2016.
[11] Dong, Y., Guo, W., Chen, Y., Xing, X., Zhang, Y., & Wang, G. Towards the Detection of Inconsistencies in Public Security Vulnerability Reports.
[12] Tounsi, W., & Rais, H. (2018). A survey on technical threat intelligence in the age of sophisticated cyber attacks. Computers & security, 72, 212-233.
[13] CISA. Traffic Light Protocol (TLP) definitions and usage. https://www.us-cert.gov/tlp. Retrieved: April 26, 2019.
[14] Steinberger, J., Sperotto, A., Golling, M., & Baier, H. (2015, May). How to exchange security events? overview and evaluation of formats and protocols. In 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM) (pp. 261-269). IEEE.
[15] Mavroeidis, V., & Bromander, S. (2017, September). Cyber threat intelligence model: an evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence. In 2017 European Intelligence and Security Informatics Conference (EISIC) (pp. 91-98). IEEE.
[16] Cichonski, P., Millar, T., Grance, T., & Scarfone, K. (2012). Computer security incident handling guide. NIST Special Publication, 800(61), 1-147.
[17] The twelve-factor app. https://12factor.net/. Retrieved: April 26, 2019.
[18] GNS3. https://www.gns3.com/. Retrieved: April 26, 2019.
[19] NIST. Official Common Platform Enumeration (CPE) dictionary. https://nvd.nist.gov/Products/CPE. Retrieved: April 26, 2019.
[20] Semantic versioning 2.0.0. https://semver.org/. Retrieved: April 26, 2019.
[21] Rapid7 Corporation. Metasploit. https://www.metasploit.com/. Retrieved: April 26, 2019.
[22] MITRE Corporation. Common weakness enumeration. https://cwe.mitre.org/index.html. Retrieved: April 26, 2019.
[23] MITRE Corporation. Common attack pattern enumeration and classification. https://capec.mitre.org/. Retrieved: April 26, 2019.
[24] CISA. Common Vulnerabilities and Exposures. https://cve.mitre.org/. Retrieved: May 13, 2019.