Acta Universitatis Sapientiae
Electrical and Mechanical Engineering, 6 (2014) 68–77

Identifying Chains of Software Vulnerabilities: A Passive Non-Intrusive Methodology

Béla GENGE, Călin ENĂCHESCU

Department of Informatics, Faculty of Sciences and Letters,
Petru Maior University of Tg. Mureş, Romania
e-mail: bela.genge@ing.upm.ro, ecalin@upm.ro

Manuscript received May 4, 2015; revised August 20, 2015.

Abstract: We present a novel methodology to identify chains of software vulnerabilities in computer networks. Vulnerabilities constitute software bugs, which may enable attackers to perform malicious operations. Attacks against systems, however, embrace various software flaws on different machines interconnected by networks. As a result, vulnerability chains may give the attacker the ability to compromise a series of hosts, and to reach his goals. This paper shows that an attacker may infer software vulnerabilities by leveraging passive network monitoring tools, and may construct vulnerability chains in the attempt to penetrate security perimeters. We present an approach to automatically build vulnerability chains based an automated vulnerability reconstruction reports from National Vulnerability Database, and passive network analysis tools such as PRADS. The approach is experimentally validated in a laboratory test infrastructure.

Keywords: software vulnerability, vulnerability chain, network sniffer, National Vulnerability Database.

1. Introduction

Vulnerabilities constitute software flaws that may enable attackers to run random code sequences, to escalate privileges, and ultimately to take complete control of underlying Operating Systems (OS). Therefore, extensive effort has been invested in the description and rapid dissemination of software vulnerabilities.

However, starting with their description, the wide dissemination of software vulnerabilities faces several challenges. To begin with, vulnerability reports need to be identically formulated across various institutions and databases. Then, a structured format needs to be provided to enable automated processing

and reasoning with various tools. To facilitate the sharing of vulnerability-related information, the Common Vulnerabilities and Exposures (CVE) was introduced in 1999 by the MITRE Corporation. One of the most well-established vulnerability databases is the National Vulnerability Database (NVD), and it builds on the information provided by CVE.

Besides individual vulnerabilities, security experts need to account for the complexity of various cyber attacks embracing different vulnerabilities of software running across different host. In this context, attackers may adopt chains of software vulnerabilities, in which case exploits of vulnerabilities on one particular host may give access to running a different set of exploits on another set of hosts. Therefore, security experts need to be aware of possible vulnerability chains in order to limit the attacker's ability to reach critical assets.

In an attempt to address this challenge, this paper proposes a methodology to build vulnerability chains based on passive assessment of network traffic. The methodology embraces recent advancement in the field of passive network asset discovery [1], and automated vulnerability reconstruction [2]. Subsequently, it builds on the well-established field of attack trees [3], and their extension with probabilistic computations as proposed in the work of Nai Fovino, et al. [4]. The methodology is experimentally evaluated in the context of a simplified laboratory-scale scenario.

The remainder of this paper is organized as follows. Section 2 provides an overview of related methodologies and it emphasizes the main novelty of the technique proposed in this work. Then, Section 3 provides an overview of vulnerability reports and of the tools employed in this work, which is followed by a detailed description of the proposal. Next, Section 4 provides the results of experiments conducted with the proposed method. Finally, the paper concludes in Section 5.

2. Related work

In the field of vulnerability assessment we find various methodologies, which may be categorized in two classes. In the first class we find approaches from the field of active vulnerability assessment. Here we mention *Nessus*, an "all-in-one" vulnerability assessment tool [11]. *Nessus* actively probes services in order to test for known vulnerabilities and possible service configuration weaknesses. It relies on plug-ins which are specifically written to test for the presence of particular vulnerabilities. It generates a comprehensive report which contains descriptions on discovered assets and vulnerabilities, but also recommendations for improving system security. Next, we mention *ZMap* [12], a network scanner that provides valuable information on discovered services to vulnerability assessment tools.

In the field of passive network asset discovery we can find several tools such as *p0f* [13] and *PRADS* [1], which rely on user-specified signatures to distinguish between specific products and version numbers. These tools generate a list of discovered assets from network traffic capture files.

Finally, we mention the work of Cheminod, et al. [10], which mostly relates to ours. In their work, Cheminod, et al. extended the structure of CVE reports with entries on vulnerability causes and effects in the attempt to build an automated reasoning framework on vulnerability chains. However, the approach requires the modification of each new CVE from other open databases, i.e., NVD. Therefore, we believe that the methodology proposed in this work represents a significant improvement since it leverages open and available CVE reports to build probabilistic vulnerability chains.

3. Proposed methodology

A. Overview of vulnerability reports

As mentioned above, vulnerability reports provide a description of software flaws. Each item in CVE is identified by the year and the order they were included in the database. For example, the recently reported software vulnerability identified by CVE-2015-0699 refers to Cisco Unified Communications Manager's Interactive Voice Response (IVR) component, and the ability of an attacker to launch an SQL injection attack, which may in turn permit the execution of arbitrary SQL statements. The CVE was added to the database in 2015, and was given sequence number 0699.

One of the most well-established vulnerability databases, the National Vulnerability Database (NVD), builds on the information provided by CVE. NVD is often viewed as the "ground truth" for software vulnerability assessment [5]. At the heart of every CVE entry lies the Common Platform Enumeration (CPE), "a standardized method of describing and identifying classes of applications, operating systems, and hardware devices present among an enterprise's computing assets" [6]. CPE names provide information on software vendor, name, version, language, edition, etc.

Each CVE contains various entries among which we mention a summary of the vulnerability, a list of CPEs to which the vulnerability applies, and a scoring on its severity and exploitability. Vulnerability scores are given in the Common Vulnerability Scoring System (CVSS) format. CVSS scores range from 0 to 10, 0 being the lowest (low severity), and 10 being the highest (highest severity) vulnerability score that can be assigned to a specific CVE entry.

B. Overview of automated vulnerability reconstruction tool: ShoVAT

The vulnerability chain reconstruction methodology proposed in this work embraces the advanced features exposed by ShoVAT, as described in our previous work [2]. ShoVAT stands for *Shodan-based vulnerability assessment tool*, and it aims at automatically identifying vulnerabilities in Internet-facing services. ShoVAT uses *Shodan* search engine [7, 8] to discover services and service-specific data, e.g., service banners. Conversely, this work builds on data extracted from network traffic and on ShoVAT's ability to automatically identify software.

Besides service-specific data, ShoVAT employs NVD as a source of CVE reports. CVEs are downloaded once per-day, as they are regularly updated in NVD, and are used to reconstruct an in-memory representation of the relationship between CPEs and CVEs. The memory-resident model is based on bipartite-hypergraphs, where each CPE may be associated to several CVEs, and each CVE may be associated to several CPEs. Besides these, efficient hash functions are implemented in order to provide efficient access to various data structures.

Finally, ShoVAT's output is a comprehensive report detailing the discovered hosts, services, CPEs and vulnerabilities.

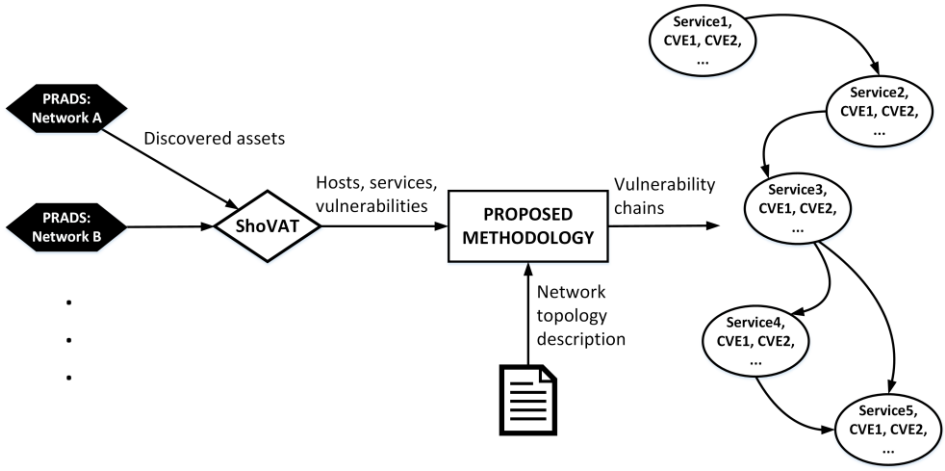

Figure 1: Architecture and components in the proposed methodology.

C. Proposed approach

The proposed methodology for vulnerability chain construction leverages various tools and well-established methodologies (see *Fig. 1*). The network asset discovery is provided by the Passive real-time asset detection system

(PRADS) [1]. Next, the data discovered by PRADS is processed by ShoVAT [2] in the attempt to identify software vulnerabilities. Its output constitutes a list of hosts, services, and CVE reports.

The proposed methodology unfolds several significant steps. At first, it takes a description of the network topology, given as a list of hosts (IP addresses), services (port numbers), and physical connections. This particular entry file also contains logical connections between services, that is, traffic flows between various software and hosts. An obvious enhancement to this data collection methodology could be the adoption of automated network topology discovery mechanisms. However, while such approaches might reduce the time needed to discover the network, their provisioning at various network locations is not trivial. Subsequently, several issues might rise along the way. In this respect, the time dimension of the network discovery phase is a significant aspect, since specific software and inadvertently, communication protocols, might be executed at various points in time.

It is noteworthy that the methodology described in this paper also embraces automated asset discovery methodologies. These are implemented with the help of PRADS, which provides a list of software and OS descriptions for each asset. An example output of PRADS is the following:

```
10.0.0.100,0,80,6,CLIENT,[http:Mozilla/5.0 (Windows
NT 6.1; WOW64; rv:30.0) Gecko/20100101 Firefox/30.0],
1,1404207546
```

In the example above PRADS identified a Firefox Web browser client, version 30.0. The detection is based on patterns described in PRAD's input files. Therefore, the accuracy of the detection depends on various factors, and it is a significant element, which influences the accuracy of results of the proposed methodology as well. An example in this sense is PRAD's ability to detect specific OS. The detection leverages the structure of network packets, specific bits, and so on. However, in many cases the procedure may produce false results due to the lack of sufficient information to accurately distinguish specific OS versions [9]. An example in this sense is the following output, as produced by PRADS:

```
10.0.2.121,0,49251,6,SYN,[8192:127:1:48:M1460,N,N,S:.
:Windows:2000 SP2+, XP SP1+ (seldom 98):link:
ethernet/modem],1,1404207849
```

According to this example, PRADS states that the OS version might be Microsoft Windows 200 SP2+, yet it may also be Microsoft Windows XP SP1+, and in rare occasions Microsoft Windows 98. Therefore, such output

poses significant challenges even in the hand of experienced security experts. Nevertheless, it should be noted that since PRADS is a passive network asset discovery tool, it is limited to such outputs, and a more accurate result may be only achieved by leveraging active detection methodologies.

In the next phase, PRAD's output is processed by ShoVAT's automated vulnerability identification modules. Since ShoVAT was intended to process data from Shodan, we extended ShoVAT with a new module written in the Python language in order to process PRAD's output. ShoVAT takes PRAD's output and reconstructs CPE names based on asset names and version numbers. Then, it extracts all known CVEs for all matching CPEs. As an example in this sense, for the previous example on Firefox ShoVAT identifies the following CPE:

```
cpe:/a:mozilla:firefox:30.0
```

Then, ShoVAT searches for known CVE and gives 52 matches. This yields a number of 52 known vulnerabilities that are associated to Mozilla Firefox version 30.0.

Based on these aspects, in the next phase the methodology at hand builds a graph-based representation of nodes, vulnerabilities, and vulnerability scores. The most significant issue with the construction of such graphs, however, is the actual exploitability of vulnerabilities. That is, for each CVE we need to establish if an attacker may exploit the vulnerability, and what are the consequences of the exploited vulnerability.

Unfortunately CVEs do not formalize these concepts, and they are summarized in an intuitive, natural language-based form. In fact, related research on building vulnerability chains [10] extended the structure of CVE with entries that specifically identify the possible exploits. However, the developed methodology is not accessible to the public, and it entails the extension of each CVE with such specific entries. Therefore, the adoption of the methodology as proposed in [10] may not be feasible in real systems due to the high maintenance requirements for keeping a separate and up-to-date database of extended CVEs. This is also easily understandable since at the moment the entire NVD contains more than 100 thousand CVEs, and their expansion with exploitability-specific information would be highly unreasonable.

Based on these facts, in this work we adopt the numerical scoring available in CVE as a measure of vulnerability exploitability. As already stated, CVEs include a set of CVSS values, among which we also find various sub-scores such as *exploitability* and *impact*. Since these have already been assigned by security experts, we adopt these numbers in the proposed methodology. More specifically, since their values are in the range from 1 to 10, we use them in the

form of probability assignments. In other words, the exploitability sub-score will represent the probability of successful exploitation (after division by 10), denoted by P_{EXP}, while the impact sub-score will represent the probability of escalating privileges (after division by 10), denoted by P_{IMP}. As a result, the probability of successful escalation of privileges by means of software vulnerability exploitation is computed as $P_{SE} = P_{EXP} \bullet P_{IMP}$.

Then, from the host's point of view, which encapsulates various software and vulnerabilities, the probability of privilege escalation includes all software and vulnerabilities found on that particular host. More specifically, the privilege escalation for a host i is defined according to [4] as $P_{Hi} = 1 - Prod_k(1 - P_{SEik})$, where $Prod_k$ computes the product of k successful privilege escalation probabilities for host i. The chain probability between two hosts is computed as $P_{Hi} \bullet P_{Hj}$.

Finally, given the above equations, we can calculate the probability for an attacker to successfully reach his target. For this purpose we use well-known graph-based algorithms, which maximize/minimize the cost of a specific path in the graph.

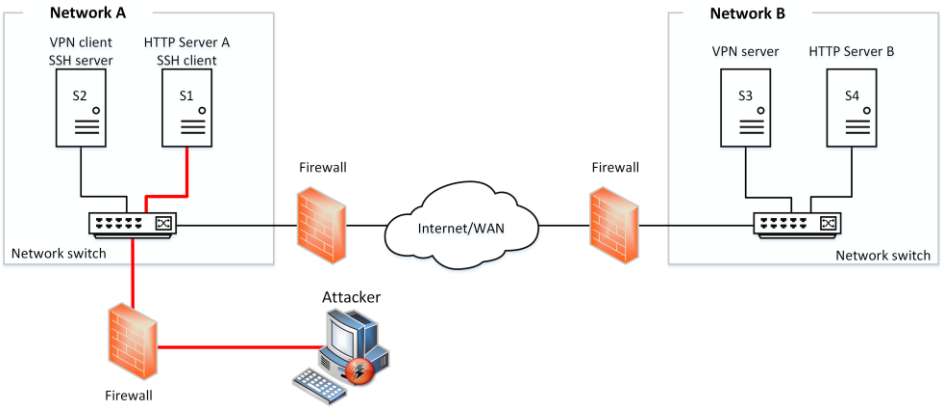

Figure 2: Experimentation setting.

4. Experimental results

The experimentation assessment assumes a simplified topology (see *Fig. 2*) consisting of two networks, i.e., *Network A* and *Network B*. Network A hosts two nodes: S1, running HTTP server A and SSH client, and S2, running a VPN client to Network B and SSH server. Network B hosts another two nodes: S3, running VPN server, and S4, running HTTP server B.

The attacker is located outside of the two networks. Both networks are protected by firewall, however, certain services are allowed through. As such, the attack is permitted to access HTTP server A on S1, and VPN client in Network A can connect to VPN server in Network B. In the case of the two HTTP servers we use Apache Web server version 2.4.4.

Based on this scenario we use PRADS to identify network assets on both networks and we feed this data to ShoVAT. Finally, we use the proposed methodology to build the vulnerability chain.

With the data discovered by PRADS, ShoVAT reconstructs the following CPEs associated to HTTP server A:

```
cpe:/a:apache:http_server:2.4.4
cpe:/a:apache:apache_http_server:2.4.4
```

Based on these CPEs ShoVAT then identifies several vulnerabilities. For the sake of the example at hand we select one particular example in order to illustrate the application of the proposed approach. More specifically, we select CVE-2013-2249, a vulnerability from 2013 affecting all Apache HTTP servers before version 2.4.5.

According to NVD, the aforementioned vulnerability is associated to `mod_session_dbd.c` in the `mod_session_dbd module`, which may allow an attacker in certain cases to run various attack vectors. The vulnerability has a CVSS score of 7.5, therefore, a *high* score. The impact sub-score is of 6.4, while the exploitability is of 10. Therefore, an attacker might exploit the vulnerability with a probability of 1, yet, the probability of successful escalation of privileges, is of 0.64.

As a result, a remotely located attacker, outside the perimeters of Network A, may successfully exploit the vulnerability with the probability of 0.64. Then, the attacker can open an SSH connection to S2, from where it gains access to S3. This entails, however, that the VPN configuration between S2 and S3 does not require further verification of credentials.

Once the attacker reaches this point, i.e., S3, it can adopt the same vulnerability in Apache HTTP server to escalate privileges on S4 with a probability of 0.64. As a result, given the two cascading, i.e., series, probabilities, we compute that the probability for an attacker to reach S4 remotely is of $0.64 * 0.64 = 0.4096$. Therefore, the attacker might have almost 50% of chances to reach his goals from outside Network A.

Based on these results, we consider that the proposed approach represent a powerful instrument in the identification of highly critical vulnerability chains. As such, the methodology may be adopted by security specialists in order to reduce the attacker's probability of success. In the scenario at hand, for

instance, by assuming that HTTP Server A has only non-critical vulnerabilities, with a probability of successful escalation of privileges of 0.15, the overall success probability for the entire vulnerability chain decreases to 0.096. Therefore, once the vulnerabilities have been identified, the methodology provides the ability to conduct what-if calculations, which can also aid network designers to identify possible solutions and security improvements.

5. Conclusion

We presented a methodology to construct software vulnerability chains. The approach builds on passive network asset discovery tools such as PRADS [1] and non-intrusive automated vulnerability identification features exposed by ShoVAT [2]. The output of these tools is processed in the attempt to construct a graph representation of network, hosts, software, and vulnerabilities. The work also proposes metrics to quantify in a probabilistic framework the successful exploitation of vulnerabilities. In this respect we adopt the impact and exploitability sub-scores as provided by CVSS. The applicability of the proposed approach is demonstrated in a laboratory-scale test scenario. Future research will focus on additional tests and measurements in various scenarios.

Acknowledgements

The research presented in this paper was supported by the European Social Fund under the responsibility of the Managing Authority for the Sectoral Operational Programme for Human Resources Development, as part of the grant POSDRU/159/1.5/S/133652.

References

[1] Fjellskal, E., "Passive real-time asset detection system (PRADS)," 2009, [Online; available at: http://gamelinux.github.io/prads/].

[2] Genge, B., Enachescu, C., "ShoVAT: Shodan-based vulnerability assessment tool for Internet-facing services," *Security and communication networks*, Wiley, 2015 (In Press), [Online; available at: http://www.ibs.ro/~bela/Papers/SCN2015.pdf].

[3] Schneier, B., "Attack trees," *Dr. Dobbs journal*, vol. 24, no. 12, pp. 21–29, 1999.

[4] Nai Fovino, I., Masera, M., and De Cian, A., "Integrating cyber attacks within fault trees", *Rel. Eng. & Sys. Safety*, vol. 94, no. 9, pp. 1394-1402, 2009.

[5] Nannen, V., "The Edit History of the National Vulnerability Database," *Master's Thesis, ETH Zurich*, Switzerland 2012.

[6] Cheikes, B., Waltermire, D., and Scarfone, K., "Common platform enumeration: Naming specification version," *Technical Report NIST Inter-agency Report 7695*, NIST August 2011.

[7] Matterly, J., "Shodan," 2015, [Online; available at: http://www.shodanhq.com].

[8] Bodenheim, R., Butts, J., Dunlap, S., and Mullins, B., "Evaluation of the ability of the Shodan search engine to identify Internet-facing industrial control devices", *International Journal of Critical Infrastructure Protection*, vol. 7, no. 2, pp. 114-123, 2014.

[9] Richardson, D.W., Gribble, S.D., Kohno, T., "The limits of automatic OS fingerprint generation", *in Proc. of the 3rd ACM Workshop on Artificial Intelligence and Security*, AISec '10, ACM: New York, NY, USA, 2010.

[10] Cheminod, M., Bertolotti, I.C., Durante, L., Maggi, P., Pozza, D., Sisto, R., and Valenzano, A., "Detecting Chains of Vulnerabilities in Industrial Networks", *IEEE Transactions on Industrial Informatics*, vol. 5, no. 2, pp. 181-193, 2009.

[11] Rogers, R., "Nessus Network Auditing", Syngress publishing, 2008.

[12] Durumeric, Z.,Wustrow, E., and Halderman, J.A., "Zmap: Fast Internet-wide scanning and its security applications", *in Proc. of the 22Nd USENIX Conference on Security*, SEC'13, USENIX Association: Berkeley, CA, USA, pp. 605–620, 2013.

[13] Zalewski, M., "p0f v3: Passive fingerprinter", 2012, [Online; available at: http://lcamtuf.coredump.cx/p0f3/.