IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 23, Issue 5, Ser. II (Sep. –Oct. 2021), PP 27-31
www.iosrjournals.org

Advanced Cyber Security

Harshitkumar R Panwala[1]
[1]*School of Computer Science Engineering, VIT-AP University, Amaravati, India*

Abstract: *In today's fast-paced world, application security is critical. A backdoor is a way for attackers to get access to a system in order to carry out illegal activities or infringe on users' rights. It has created a severe security risk to networks. As a result, it is critical to implement proper defensive measures against such attacks. In the present study, the problem of web backdoor detection and prevention has been accomplished. The investigation of the various installation methods and concealment techniques employed by web backdoors is performed. The three areas of web backdoor detection and protection: server configuration and reinforcement, intrusion detection systems, and static analysis is covered in the current study. In addition, a detection and prevention strategy for PHP backdoors that combines a whitelist along with dynamic analysis is demonstrated. By detecting sensitive functions, this method can entirely block web backdoors and has no false positives.*

Date of Submission: 28-10-2021 Date of Acceptance: 12-11-2021

I. Introduction

A backdoor is any means by which authorized and unauthorized individuals can get around typical security protections and get high-level user access (root access) on a computer system, network, or software application in the area of cybersecurity. Malicious hackers can use a backdoor to steal personal and financial data, install further malware, and commandeer devices after they've gained access. Backdoors, on the other hand, aren't solely for evil individuals. Backdoors can also be deliberately inserted by software or hardware developers to get access to their technology after the fact. Non-criminal backdoors are beneficial for assisting clients who have been locked out of their devices indefinitely, as well as troubleshooting and resolving software bugs [1, 2].

Backdoors, unlike all the other cyber threats that make themselves known to the user (think ransomware), are noted for being stealthy. Backdoors allow a small number of insiders to obtain quick access to a system or program. Backdoors aren't going away anytime soon as a threat. Backdoors were the fourth most prevalent threat detection in 2018 for both consumers and enterprises, according to the Malwarebytes Labs State of Malware report, with increases of 34 and 173 % over the previous year [3].

Backdoor malware is commonly referred to as a Trojan. A Trojan horse is malicious computer software that masquerades as something it isn't in order to spread malware, steal data, or open a backdoor on your system. Computer Trojans, like the Trojan horse from Greek mythology, always come with a terrible surprise. Trojans are a highly adaptable tool in the arsenal of cybercriminals. They can disguise themselves as an email attachment or a file download, and they can convey a variety of malware threats. To make matters worse, Trojans can occasionally reproduce themselves and spread to other systems without the need for additional commands from the crooks who built them [4, 5].

Backdoors are built-in or proprietary backdoors that are installed by hardware and software manufacturers. Built-in backdoors, unlike backdoor viruses, aren't always designed with malicious intent in mind. Built-in backdoors are frequently found as byproducts of the software development process. These backdoor accounts are created so that software engineers can swiftly move in and out of applications while they're being coded, test their apps, and solve software issues or errors without having to register a "genuine" account. Backdoors were not supposed to come with final software that's given to the public, but they occur occasionally. It's hardly the end of the world, but there's always the possibility that a proprietary backdoor will slip into the wrong hands. The current work presents a method or the way through advanced cyber security medium that prevents such backdoor cyber-attacks [6, 7].

II. Literature Review

The backdoor, according to Thomas and Francillon [8], is a purposeful structure in the system that degrades the system's original security by allowing easy access to additional privileged functions or information. Backdoors, also known as webshells, C/S backdoors, thread insertion backdoors, and extended backdoors, are program methods that circumvent security restrictions such as authentication to access system permissions. The backdoor program in the online application is referred to as Webshell. PHP, ASP, JSP, Python, and other programming languages are among the file formats supported.

The webshell is installed in the server file directory when the attacker leverages the vulnerability to infiltrate the website for future remote control, execution of malicious commands, and other operations. To obtain control, the C/S backdoor employs a client/server mechanism, which shares several concepts with traditional Trojan programs. The client launches the backdoor to control the server once the attacker implants it into the target machine. When a thread insertion backdoor operates, it does not run as a separate process but instead inserts itself into a certain service or thread of the system, which is currently a common sort of backdoor. It is, however, difficult to detect or eliminate, and typical firewalls are unable to adequately guard against it. System user detection, port opening, opening/stopping services, HTTP access, and file upload/download are all typical features that are frequently concentrated in the extended backdoor. It has a lot of capabilities; however, its concealment isn't very good [9].

In the subject of security, there is currently limited research on Python harmful code. The majority of backdoor detection publications concentrate on webshell. Simultaneously, JavaScript is utilized as an interpretable language, and some of the approaches for detecting harmful statements can be applied to other languages. This section will review previous research on webshell and malicious JavaScript from both a static and dynamic detection standpoint [10].

NeoPi [11] is a well-known open-source webshell detection program. It looks at the document's statistical characteristics to see if it's obfuscated, then compares it to a feature function to see if it's obfuscated. The feature database, on the other hand, is quite ancient. Based on malicious signatures and malicious functions, Tu et al. [12] discovered webshells in apps. At the same time, it was suggested that the header tag simply evaluate the beginning and conclusion of the longest word. Although this method lowered the number of false positives from NeoPi's 24.5 percent to 6.7 percent, the detection principle remained the same: basic character matching.

Lawrence et al. [2] created a firewall tool that intercepted and alerted calls to system functions that weren't whitelisted. Due to the narrow whitelist, there were a lot of false positives, and the webshell, which was encrypted and obfuscated using complex methods, couldn't be found. AL-Taharwa et al. [13] designed and implemented the JSOD (JavaScript Obfuscation Detector), a JavaScript obfuscation detector that focused on obfuscated scripts. It used the AST to do anti-obfuscation processing and retrieve the code's contextual semantic information (Abstract Syntax Tree). The Bayesian classifier then identified the malicious JavaScript.

To discover malicious code, dynamic detection involves dynamically executing sample files, monitoring network traffic, or calling sensitive routines. This approach is frequently used to investigate unique file behavior, such as extracting HTTP requesting or responding to payload features and hooking sensitive functionalities. To detect malicious JavaScript, Kim et al. [14] created the JsSandbox framework. Functions that could not be run by API hooking could be extracted using IFH (internal function hooking) monitoring and analysis of sample code behavior. This method was extensively utilized in various malicious code categorization jobs, and its detection efficiency was unaffected by procedures like code deformation and obfuscation. Canali et al. [15] employed honeypot technology to collect and analyze the attacker's behavioral features related to the target's destruction. HTTP request logs and files updated or produced after the attacker gained permission from the target host were among the data sources. It also looked at some of the most prevalent webshell behaviors.

As a result, to get superior classification results with limited resources, research has merged static and dynamic detection. Cujo, a malicious JavaScript automated detection method proposed by Rieck et al. [16], was assessed by integrating static lexical information and dynamic runtime features. The dynamic analysis obtained web page code via sandbox, generated analysis reports, and then mapped the report results to vector space. SVM machine learning was used to create the final detection model. The JavaScript malware detection tool JSDC was developed by Wang et al. [17]. To begin, it used machine learning to recognize the retrieved text, program metadata, and risky function call attributes. The malware was classified into eight attack categories based on the attack feature vector and dynamic execution trajectory. A false-positive rate of 0.2123 percent and a false negative rate of 0.8492 percent were attained using the dynamic and static combination.

Starov et al. [3] used two dimensions of static and dynamic analysis to undertake an in-depth investigation of common webshell behaviors. The majority of the sample attacks, according to static analysis, involved file browsing and uploading, system information viewing, command execution, and so on, but dynamic analysis revealed that the majority of them attempted to access links or directories such as http:// and /etc/. Dynamic detection is more effective at detecting malicious code behavior, but its inherent flaws necessitate a lot of resources and samples, and its practical applicability is limited. To choose a good detection method, a variety of criteria must be taken into account.

III. Security Measures

The following security measures must be adopted in order to prevent your device to get into any cyber backdoor kind of trouble.

3.1 Change default password

The user has unknowingly built a backdoor by using the default password. To reduce risk, change the default password as soon as possible and use Multi-Factor Authentication (MFA). It can be difficult to remember a distinct password for each program. According to a Malware Bit Labs survey on data privacy, 29 percent of respondents used the same password across several apps and devices.

3.2 Monitor network activity

If the user sees any unusual data spikes, it's possible that someone is using a backdoor on the user's device. Use security measures such as firewalls to track inbound and outgoing traffic from the many apps installed on the user's computer to avoid this.

3.3 Carefully selection of applications and plugins

Backdoors are hidden inside seemingly benign free programs and plugins by cybercriminals. The greatest and most straightforward defense is to ensure that all programs and plugins users install are from a reliable source.

3.4 Make use of a reliable cyber security solution

Any decent anti-malware solution can prevent cybercriminals from deploying the trojans and rootkits used to open those troublesome backdoors.

IV. Methodology and Result

Figure 1 depicts the flow diagram of the present work. The step-by-step methodological process is explained in this section. The first step of the process is to generate a PHP file. Weevilly can be used to create a PHP shell. The shell should be created with a password and the name of a file. Subsequently, a website should be searched that is to be targeted. That website must have a file uploading vulnerability. The next step is to upload a shell file produced by weevely. Then upload the PHP backdoor file into the website.

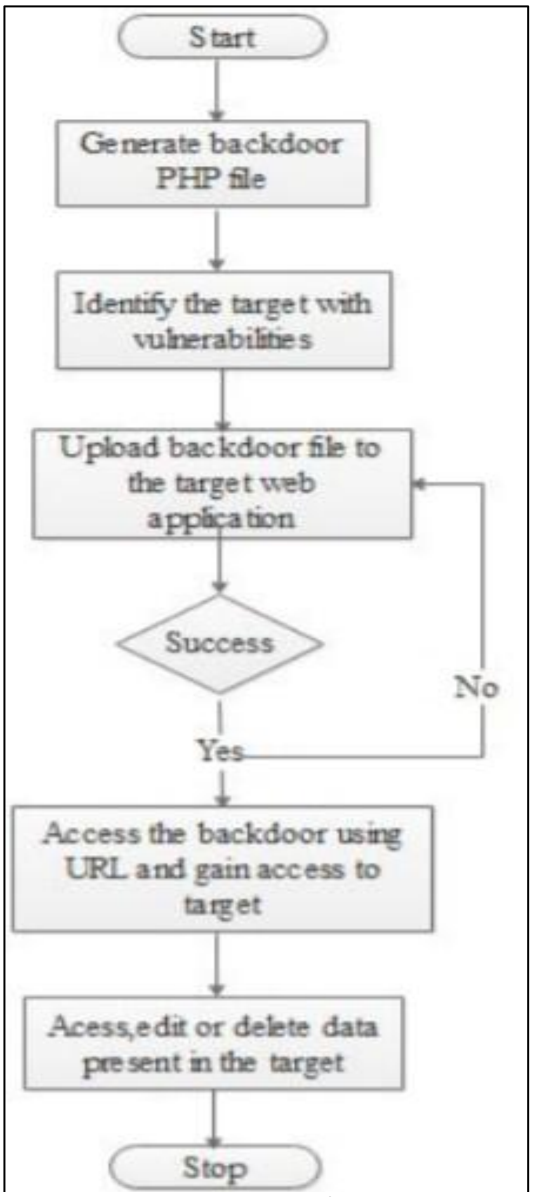
Start
Generate backdoor PHP file
Identify the target with vulnerabilities
Upload backdoor file to the target web application
Success
No
Yes
Access the backdoor using URL and gain access to target
Acess,edit or delete data present in the target
Stop

Figure 1: Flow diagram of present work

Following, using the kali console, gain access to the server. Using the password-protected backdoor file, gain access to the website server and files. Using the "ls" command, you can see all the files in the current directory. Afterward, Using the "cd" command, navigate through the server's directories. The shell file is in the upload's directory; use the "cd" command to move to the home directory. Subsequently, access the server's files. Using basic Linux commands, anyone can now edit, update, add, or delete any file on the website server. Using gedit, open the index.php file. The index.php file is now open in geditor, and any changes made there will be reflected on the server. You can now make any changes to the website and gain access to the database files, as well as all the data included within them.

V. Conclusion

The attacker will obfuscate and encrypt the code in order to keep the backdoor hidden. Concurrently, the detection impact will be influenced by parts of the text that are unrelated to the function. The current paper explains the static and dynamic analysis of backdoors along with their differences. Moreover, security measures are also given, so that the user can prevent the device from backdoor cyber-attacks. The current research focuses on determining how the Backdoor is abused by penetrators who use threats to force their way into and out of the Web application. It's tough to spot built-in backdoors and defend oneself against them. The majority of the time, the manufacturers are completely unaware of the backdoor's existence. As technology advances, so does the risk of being attacked. Users should be more aware of vulnerabilities to lessen the risk of them being harmed. As a result, it's critical to utilize technology carefully and to make use of security features in Web apps to reduce the danger of hurting our machine to the greatest extent feasible.

Reference

[1]. Malwarebytes Labs, “2020 state of malware report,” 2020, avaible at: https://resources.malwarebytes.com/files/2020/02/ 2020_State-of-Malwarebytes.Report.pdf.
[2]. Y. D. Lawrence, D. Liang Lee, Y.-H. Chen, and L. Xiang Yann, Lexical analysis for the webshell attacks,” in Proceedings of the 2016 International Symposium On Computer, Consumer And Control (IS3C), pp. 579–582, IEEE, Xi’an, China, July 2016.
[3]. O. Starov, J. Dahse, S. S. Ahmad, T. Holz, and N. Nikiforakis, “No honor among thieves: a large-scale analysis of malicious web shells,” in Proceedings of the 25th International Confer- ence on World Wide Web, pp. 1021–1032, Montréal, Canada, April 2016.
[4]. C. Wang, H. Yang, Z. Zhao, L. Gong, and Z. Li, “The research and improvement in the detection of PHP variable webshell based on information entropy,” Journal of Computers, vol. 28, pp. 62–68, 2016.
[5]. M. Peter and B. V. W. Irwin, “Towards a PHP webshell taxonomy using deobfuscation-assisted similarity analysis,” IEEE, in Proceedings of the 2015 Information Security for South Africa (ISSA), pp. 1–8, Johannesburg, South Africa, August 2015.
[6]. H. Zhang, M. Liu, Z. Yue, Z. Xue, Y. Shi, and X. He, “A PHP and JSP web shell detection system with text processing based on machine learning,” in Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1584–1591, IEEE, Guangzhou, China, January 2020.
[7]. Z. Zhang, M. Li, L. Zhu, and X. Li, “Smartdetect: a smart detection scheme for malicious web shell codes via ensemble learning,” in Proceedings of the International Conference onSmart Computing and Communication, Tokyo, Japan, De- cember 2018.
[8]. S. L. Thomas and A. Francillon, “Backdoors: definition, de- niability and detection,” in Proceedings of the 25th Interna- tional Symposium on Research in Attacks, Intrusions, and Defenses, pp. 92–113, Springer, Heraklion, Greece, September 2018.
[9]. T. K. Ho, “Random decision forests,” in Proceedings of the 3rd international conference on document analysis and recogni- tion, vol. 1, pp. 278–282, Montreal, Canada, August 1995.
[10]. Y. Guo, H. Marco-Gisbert, and K. Paul, “Mitigating webshell attacks through machine learning techniques,” Future In- ternet, vol. 12, p. 1, 2020.
[11]. B. Scott and B. Hagen, “Web shell detection using NeoPI,”2011, https://resources.infosecinstitute.com/topic/web-shell- detection/.
[12]. T. D. Tu, G. Cheng, X. Guo, and W. Pan, “Webshell detection techniques in web applications,” in Proceedings of the 5th International Conference on Computing, Communications and Networking Technologies (ICCCNT), pp. 1–7, IEEE, Hefei, China, July 2014.
[13]. I. A. Al-Taharwa, H.-M. Lee, A. Jeng, K. Wu, C. Ho, and
[14]. S. Chen, “JSOD: javascript obfuscation detector,” Security and Communication Networks, vol. 8, no. 6, pp. 1092–1107, 2015. H. C. Kim, Y. H. Choi and H. L. Dong, JsSandbox: a framework for analyzing the behavior of malicious JavaScriptcode using internal function hooking,” KSII Transactions onInternet & Information Systems, vol. 6, p. 2, 2012.
[15]. D. Canali and D. Balzarotti, “Behind the scenes of online attacks: an analysis of exploitation behaviors on the web,” in Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS 2013), San Diego, CA, USA,February 2013.
[16]. K. Rieck, T. Krueger, and A. Dewald, “Cujo: efficient detectionand prevention of drive-by-download attacks,” in Proceedingsof the 26th Annual Computer Security Applications Confer- ence, pp. 31–39, Austin, TX, USA, December 2010.
[17]. J. Wang, Y. Xue, Y. Liu, and H. Tian, “Jsdc: a hybrid approach for javascript malware detection and classification,” in Pro- ceedings of the 10th ACM Symposium on Information, Computer and Communications Security, pp. 109–120, Sin-gapore, April 2015.

Harshitkumar R Panwala. “Advanced Cyber Security.” *IOSR Journal of Computer Engineering (IOSR-JCE)*, 23(5), 2021, pp. 27-31.