**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **07/August/2020** | **Name:** | **Prashantha naik** |
| **Course:** | **Coursera** | **USN:** | **4al17ec074** |
| **Topic:** | **Network Security &**  **Database Vulnerabilities** | **Semester & Section:** | **6th b** |
| **GitHub Repository:** | **prashanth\_course** |  |  |

|  |
| --- |
| **SESSION DETAILS** |
| **Image of session** |
| **Report – Report can be typed or hand written for up to two pages.**  **Describe the nature of various injection attacks and their prevalence on the threat landscape. My name is Dmitriy Bryozoa. I am a member of X-force Ethical Hacking Team. A little bit about what we do; we do penetration testing of security products. We test products before they're released to customers. For those who don't know, penetration testing is a type of security testing, where a tester or QA person acts as an attacker, as a hacker.**  **So we use very same techniques and tools as the bad guys out there use to attack customers and products, and that helps us find security holes and we report them to development. They get fixed before the products are released. In these presentations that we do, we show you a lot of examples that we see in real case scenarios out there. Hopefully, with the recommendations that we give, you would be able to both address and prevent security bugs in your software.**  **So let's get started. Injection flaws, if we give a definition, they usually allow attackers relay malicious code through the vulnerable application to another system. Could be operating system, could be a database server, LDAP server, and just pretty much any component that accepts scripting as input. As you can see from this chart, they're fairly common. But what makes them special is that usually they're rated as high issues, top issues, and they're extremely dangerous. In the worst case scenario, they may allow full takeover of the vulnerable system. You may be familiar with OWASP Top 10 list, it's open web application security project. It gives you a list of most common security vulnerabilities that afflict web applications. As you can see from year to year, from previous version in 2013 to the current version in 2017 of the list, injection vulnerabilities are at the very top of the list. They are considered as the most dangerous type of vulnerability out there. There's also SANS Top 25, a list that you may be familiar with.**  **Here you can see it's the same picture. Positions one and two are taken by SQL injection and OS Command injection. There's agreement throughout the industry that these are the most dangerous types of vulnerabilities out there. Injection vulnerabilities, we hear about them in the news constantly. They made possible some of the most dramatic hacks in recent history. The one that you probably heard of last year was Equifax hack, where hackers use this type of vulnerability to leak data of a 150 million US and Canadian citizens.**  **That was truly massive, probably even the fact that some people on this call. Another example is the hack of TalkTalk, which is a British telecom company, records of 157,000 customers were exposed through SQL injection. If you read through news, you see these types of vulnerabilities come up very often and the end result of these types of leaks are lots of customer data being leaked, personal user information being leaked. They're really, really dangerous.**  **additional OS command injection attacks, explain why commands should not be run through the shell interpreters, explain why it is important to use explicit paths when running executables. Next recommendation is to not run commands through shell interpreters.**  **You may be familiar with shell interpreters like SSH, bash, that's on Unix, cmd and PowerShell on Windows. When you supply your command, lets say your business logic requires you to run an OS commands, so you have no choice, you have to do it. If you run it and supply it as a parameter to the shell interpreter then that also gives attacker extra power because, for example, they could chain commands. They could as syntax showed earlier, with the semicolon, you could chain a number of commands in there and the attacker could download malicious code, could delete files, could create web shells, do all kinds of damage. But if you run the command directly without passing it to the shell interpreter, then a whole set of shell syntax such as semicolon, pi, ampersand, Beck quotes, dollar sign, all that syntax that's introduced and understood by the shell interpreter is no longer valid, because you're just running a single command, in this case, a remove command, and you're passing a file to remove. So this command that you see here, the second command on the slide, will actually fail. It will not work, because the semicolon will have no meaning to the RM command. Note that even if you do that there's still danger of doing additional damage. You could not only manipulate the single parameter, but you could add additional parameters to a command that you're running.**  **So here, in this example, we're running an Nmap application to scan a particular IP address. However, if the attacker has control of the input they could specify additional Nmap parameters. In this particular case, it will cause Nmap to write the log file to a particular file that you specify. However, if it's running as root command, there's a possibility that this would actually overwrite an important system library. So bad things still could happen, it's just the attack surface is reduced.**  **We're no longer dealing with a full set of syntax that show interpreter supports, but rather concentrate on the single command. Also, something to keep in mind is that another example that you often see is that you may execute a single command, let's say a shell script, but inside that shell script there could be something in those commands that could trigger OS command execution on the parameters that are passed in. So even though you're running something directly not involving shell interpreter, there's still a chance that something could happen inside the script that you need to watch out for, and you have to scrutinize how you operate on the input that user supplies so that a malicious command is not executed and recommended. The next recommendation is to use explicit paths. Applications are found by the operating system and executed based on system path settings.**  **There is a type of attack where if there's a folder on the system that's writeable to a general user, and have multiple users using the machine, that folder also is referenced in the path before the folder containing the executable you're trying to run. What the attacker could do is it have a low privilege account on the system, they could log in and they could drop a malicious version of the application, in this case, Nmap, in the folder that's under their control and that appears in the path before the regular folder where Nmap lives. When we execute commands without the full path, just say Nmap and then the IP address, that malicious executable will be run instead of the one that we intend. So to protect against it, it's best to reference all the executables that you run by their full path, that way there is no ambiguity, and the right executable will be loaded. By the way, the same recommendation applies to another type of attack called DLL hijacking, because the same thing applies to DLLs and shared libraries, attacker could drop a malicious version of a shared library or a DLL into the folder that they control that appears in the path and that will be loaded before the valid version of that same binary file.** |