Using NASL based Superscripts to measure System Security through Analyzing and Organizing attacks

Ilias Chalvatzis, Dimitrios A. Karras and Rallis C. Papademetriou

***Abstract* — Vulnerability scanning is a very important aspect in computer network security management towards prevention of system intrusions. To this end, Nessus is a worldwide accepted such tool involving state of the art techniques. The goal of this research is to analyze the principles of vulnerability scanning using Nessus and, more importantly, to model configurations and architectures of computer networks for efficiently employing Nessus towards formalizing vulnerability scanning evaluations. Most importantly, the contribution of the herein research is to provide a superscript based framework for such a vulnerability analysis of a small to medium enterprise.**

***Keywords* — Vulnerability Scanning, Security Management, Nessus, NAS.**

I. Introduction

Vulnerability analysis (or vulnerability assessment) is the process that defines, identifies and classifies the vulnerabilities (or security holes) in a computer, network or communications infrastructure. In addition vulnerability analysis can be used to forecast the effectiveness of countermeasures to those vulnerabilities as well as evaluate their effectiveness after they are put into use [1]-[6].

Vulnerability analysis consists of several steps :

- Defining and classifying network or system resources.
- Assigning relative levels of importance to the resources.
- Identifying potential threats to each resource
- Developing a strategy to deal with the most serious potential problems first
- Defining and implementing ways to minimize the consequences if an attack occurs

Identifying potential threats can be performed by using ethical hacking techniques, where a white hat is probing a system to discover its weaknesses and develop countermeasures to prevent a genuine attack. This is how Nessus can be used for the purpose of this research.

Nessus is equipped with NASL, which is a rather straight forward scripting language and similar to C in many aspects while at the same time it is secure and specialized to perform network functions with the purpose to test a system's security against specific attacks or vulnerabilities. There are thousands of NASL scripts available in the NASL distribution for the most common systems and vulnerabilities [5], but it is also easy enough to develop custom tests for those that don't exist (like custom company software). In order to present the results to the user through Nessus NASL scripts use functions like security_warning(), while each script can use the Knowledge Base to store information retrieved from the tests it performed.

The Nessus reports provide this information for each NASL test performed, though if a cumulative report that combined all the tests reports was required (for example to provide a security overall mark for the system based on the tests), the Knowledge Base is a candidate to be used by a NASL script that retrieves information and combines them conditionally.

II. Proposed Framework for Involving Nesssus and NASL for Network Security Management

As it was described in the previous section the way to secure a system is to constantly probe it to discover and fix vulnerabilities and Nessus offers a way to probe a system or a whole network through the readily available and regularly updated CVE compatible plugins/tests and it also offers the ability to create customised Nessus tests through the NASL scripting language. The UI of Nessus is simple and user friendly, while an important enhancement to its function as a security tool would be making the feedback more specialised/user friendly and in general more useful as a metric for the total security of the system.

The way we are proposing to achieve that initially is through a superscript (written in NASL and run through Nessus) that combines the important/relevant tests for a system as well as their results and provides cumulative feedback about the security of the system.

A. Target system

For the specific purposes of the project and in order to create a demonstration a specific system will have to be specified. The first approach to that was decided to be a

Ilias Chalvatzis and Rallis C. Papademetriou are with the *University of Portsmouth, UK, School of Engineering, Faculty of Technology, Anglesea Building, Anglesea Road,* Portsmouth (UK) PO1 3DJ

Corresponding author Dimitrios Alexios Karras, is with the *EPOKA university, Computer Engineering Dept.,Tirana,* and the *National & Kapodistrian Univ., Dept. General, Psachna,* Evia, Greece e-mais:dakarras@uoa.gr,dkarras@epoka.edu.al, dimitrios.karras@gmail.com.

network demonstrated through a virtual machine (running VMware). What needs to be defined is the specifics of the systems that will be emulated (OS and running software) as well as the relevant tests that can be performed to those systems.

Initially in the Nessus demonstration presented in this report the two systems that were used were a machine running windows XP as a host and a virtual machine running Ubuntu linux as a client through VMware.

The scope of the testing will actually determine the systems that will be used as well as the content of the superscript file and the rest of the NASL scripts that will be used on the project. This could be for example a test of the security of common internet applications (Internet Explorer, MSN, Skype etc) running to a system.

B. Combining results

The purpose of this project and the way we want to move one step forward from what Nessus testing usually offers is getting an overall vulnerability analysis and testing for a system more specifically by combining various tests to determine a relative security "mark" for said system or network.

The Vulnerability database that Nessus offers could be a key element of this by offering the ability to easily share results among tests so that the superscript might not contain all the testing itself but rather combine various tests to provide a more user friendly and indicative for the total security of the system result.

It is an important design choice to decide whether it's better for the "superscript" do all the testing itself or use other scripts to get feedback as it is also key to determine how to combine these results to evaluate the total security of the system and provide collective feedback.

NASL scripts can provide feedback in the form of plain text and hyper text to the user and that's what Nessus uses itself. While text along with hyperlinks might be enough for the purpose of the project, other possible ways of presenting the results could be investigated, maybe even bypassing Nessus to display the results in another format or shell. While this is not a priority in the design and presenting the feedback through Nessus is totally acceptable a new shell written by scratch to handle the result presentation or even the input (as Nessus was shown to work with command line arguments and gives the ability to save reports in html format) could be an interesting way to enrich the project.

The motivation for doing this project is mainly focused in two areas; demonstrating the use of Nessus and NASL for the purpose of vulnerability assessment and combining different test results to get an overall "image" (or "mark") for the overall security of a system. Thus, this research aims at providing a proper framework for using Nessus for security assessment with an interactive Nessus superscript running on a simulated network environment and towards defining aspects of the theoretical research about "measuring" the overall security of a system.

The Nessus software uses "plugins" (or NASL scripts) in order to perform various security checks. These plugins are written in NASL and can vary from complex attacks to simple version checks. A Nessus superscript would combine some of those scripts (there are almost 30000 of these currently available) and their results in order to achieve some extra feedback to the user.

For instance a network specialist could create a custom "Superscript" that combines various plugins/security checks that he considers important for his network and provides custom collective feedback (rather than looking through a whole Nessus report) based on the combination of the results. In addition to the security checks packaged with Nessus there could also be custom checks made by the security specialist tailored to his own network.

Each host is associated to an internal knowledge base, which contains all the information gathered by the tests during the scan. The security tests are encouraged to read it and to contribute to it. This enables the tests to exchange info and save time and resources.

The latest part is the most important for our purpose making the knowledge base the key to combining tests through their results and not just by copying and including their code into a single script.

The way a superscript would work to achieve getting the results of various other tests is using the knowledge base.

When a test wants to save an item in the knowledge base the function "set_kb_item(name:<name>, value:<value>)" will mark the new item <name> of value <value> in the knowledge base and this item's value can be recalled from tests that are run after it using the "get_kb_item(<name>)" function.

So for a test to include the results of other tests the only thing that is required is to tell the script to perform the other test in advance by using the script_dependencies(); function in the description section and then use the function "get_kb_item()" to get the result that this test has saved in the knowledge base.

It is worth to note that a script cannot read back a knowledge base item that it has added on its own.

Apart from the use of other scripts with the Knowledge Base the superscript would act as any other normal script, with its internal variables and functions. Those can in turn be used along with conditionals to combine the Knowledge Base contents and produce results.

For example if the goal or one of the goals of a Superscript is to check for the existence of an antivirus on the system it can use already existing scripts like "nav_installed.nasl", "mcafee_installed.nasl" through the items they create on the Knowledge base like "Antivirus/Norton/installed" and "Antivirus/McAffee/Installed" to determine if an antivirus is installed on the system and have an internal variable "AVexists" that would turn true if an antivirus was found on the system.

This variable can then be used in further conditionals to modify the final report (different text if an antivirus exists or not) or be combined with other variables (possibly from different tests) for further tests (like only perform a certain test if there is no antivirus) or finals results (like change a total mark of security depending on the existence of antivirus software). This can be done through simple conditionals (if (AVexists)).

While the script itself can't use the items it will save in the knowledge base, it can do it anyway for them to be available for other scripts.

A NASL script can have two useful outputs for the user. The first one like it was mentioned above is the creation of

items in the Knowledge Base which is not something the user will immediately see at the end of the script unless they are used in another script.

The second possible output of a Nessus script is the "report" string, provided to the interpreter through the functions of the type "security_hole(port:port, data: report)".

While the description string is static the rest of the report doesn't have to be. While forming the strings variables can be used (including Knowledge Base items) also the content of the Strings can be decided through conditionals (for example if this variable is true use this string else use another one).

What makes the report string convenient to be used is the fact that it doesn't have to be created all at once. Strings can be manipulated like in C for example where strings can be added in different order. So different functions of the script can create their own strings based on their own conditionals and results and those can be summed up in the end to form the report string in the desired order.

For example a script can create different strings like string1,string2,string3 and then create the final report by combining them : "report = string1 + string2 + string 3". The same way those substrings can be updated "string1 = string1 + "\n This is an addition to String1".

The report string of the script is ultimately the end result of the script as far as the user can see presenting the results in a text form with possible hyperlinks.

C. A Nessus Superscript – Client Internet security

To demonstrate the logic described above an example superscript that tests the Internet security level of a client was created. At this time this script tests the safety of 4 different areas of a client's interaction with the Internet, the Web Browsers, the Internet Messengers, the existence of an Antivirus and the Windows Service pack version.

The Web Browsers and the Internet Messengers are the most immediate points of contact of a user with the Internet. Exploits for them are very common and are developed constantly so upgrading them to the latest version where those exploits have been fixed is a key element to the safety of a client against Internet threats.

In the case something dangerous comes through or is not directly related to the web browser or messenger software an Antivirus is key to detecting it and removing it so the existence of an Antivirus software is also important. Finally in the case of Microsoft Windows holes are discovered on the OS itself so keeping up with the latest updates and Service Packs is also very important for the security of a client.

The Superscript tests all those elements of a client by determining the existence of this software and whether the latest known (and safest) version is used. The scripts informs about the existence and the version of web browsers and web messengers and warns if any of them is not the latest version while providing a link to a location where the latest version can be downloaded. Also the existence of a known Antivirus software is determined as well as the version of Windows and the Service pack being used.

The overall security mark given to the client is determined based on those tests with different marks given for each area based on importance and marks being taken away if there is a warning.

Adding more elements (tests, ways to determine the score) to this script would not be a difficult task because of the way it is modularly designed, in fact using the Knowledge Base and adding items into it (like the total or individual scores, the warnings or even the output text) would allow for this script to be used as a part of another script (maybe a larger scope script with more tests) the way this script uses the scripts provided by Nessus.

D. Results and Output

As mentioned in the previous section the Superscript in effect combines a number of different scripts through processing their KB items and forming an output based on those along with the internal Superscript functions.

There are two possible outputs of a Nessus scripts, saving items in the Knowledge Base and the feedback in the form of text presented to the user through Nessus.

The first form of output is to be used by other scripts as the Superscript itself cannot use it. It can be very useful however when creating related scripts. The way the script can save an item in the KB is the command **set_kb_item (name : , value:**). For example saving the overall security mark determined by the Superscript in the KB is as simple as using a command like **" set_kb_item(name : " Client Internet Security mark" , value : score)"** where "score" is the numerical value that is determined at the end of the Superscript. This score can then be used by other scripts executed after the Superscript, for example there could be a different script that combined scripts like this Superscript that marked different areas of the client's security to generate an overall security mark. The items saved in the KB can be any strings so the possibilities of making the results of a script useful to another are a lot.

The second form of output is what the Nessus user will receive as feedback after running the Superscript. It is a string that is fed to Nessus at the end of the script with a command like **" security_hole(port:port, data:report**);" where the string in this case is the "report" variable. This string is formed modularly by combining various strings that are created in the subsections of the Superscript dynamically while the script is being run.

For example if it is found that a version of Internet Explorer is installed on the client the reportIE string is created with the command **"reportIE = '\n' + "*** INTERNET EXLPORER" + '\n' + "The remote host has Internet Explorer version " + IEversion + " installed." + '\n';** " which contains static text and the variable IEversion which as it was shown above comes from the KB.

Now if the script code determines that this version of Internet Explorer is unsafe the variable IEwarning is set to 1 (**"IEwarning = 1"**) which means that there is a warning about IE and the string related to IE is updated with additional info : **" reportIE = reportIE + '\n' + "-- Your version of Internet Explorer is not up to date." + '\n' + "Please visit http://www.microsoft.com/windows/internet-explorer/default.aspx" + '\n' + "to update your**

installation" + '\n' ;" which apart from warning the user that there is something wrong with Internet Explorer provides a link (clickable in the Nessus report) to update it. If the version of IE was determined to be up to date the string would be updated differently : **" reportIE = reportIE + '\n' + "-- Your version of Internet Explorer is up to date." + '\n'; "** .

This goes on consistently for every browser creating appropriate warning variables like "IEwarning" and report strings like "reportIE" until every web browser check has been performed and the script reaches to the subsection where it combines all the results related to browsers and creates the variable **"Browserwarnings"** which sums up the number of security warnings related to Web Browsers and the string "**reportBrowsers"** which combines all the individual browser reports **"reportBrowsers = '\n\n\n' +** "**+ '\n' + **"The following BROWSERS have been found on the host" + '\n' +** "***" + '\n' + **reportIE + reportFF + reportOP + '\n';** " . In similar fashion with the individual browser reports the report for the Browser can vary depending on the variable "Browserwarnings" so if any warnings have been found the report string is updated : **" reportBrowsers = reportBrowsers + '\n'+" !!! Some of your browsers seem to be outdated. There are known vulnerabilities for your internet browsers so please follow the above links to update them to the latest combatible version. !!!" + '\n\n';"** .

The rest of the individual and section reports are created similarly and then the script gets to the final part where it combines all of them to create the final output. This is also the section where the final score is determined based on the sectional scores, in this case a certain amount of points is awarded for every section that is found safe to determine a numerical value in the scale of 1-10.

The code to determine the final score is

if (Antivirusexists == 1) Finalscore = Finalscore + 3;
if (Servicepackwarning== 0)Finalscore=Finalscore + 3;
if (Browserunsafe == 0) Finalscore = Finalscore + 2;
if (Messengerunsafe == 0) Finalscore = Finalscore + 2;

The output is a single string so all the reports need to be combined into a single string called "report", **"report = desc["english"] + '\n\nPlugin Output :\n\n' + score + reportBrowsers + reportMessengers + reportAntivirus + reportServicepack;".**

All the individual reports are made in a way that they can easily be updated, used by other scripts (if saved in the KB) and are clear to see, hence the use of '\n's to change lines or "**" to show the change of sections. The way the strings are updated by adding strings together makes it easy to change the text, the conditionals or the order of text.

III.PRELIMINARY RESULTS - SCENARIOS

An example of what the Superscript's report string would look like can be seen in the following screen (Fig.1) which is the output of the Superscript when run through the NASL interpreter (using the "nasl" command).

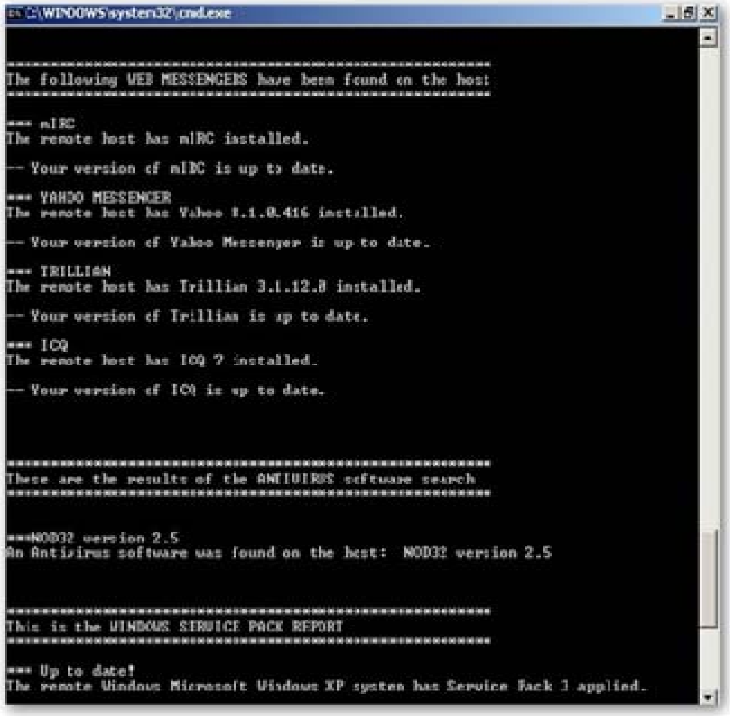

Fig. 1. The output of the superstring illustrated in the previous section running through the NASL interpreter.

This is how the script output looks like on the Nessus software and Fig. 1 shows the script on the plugin selection screen providing its info like name, ID, family, description, dependencies.

IV. CONCLUSIONS

This research report provides a framework for effectively employing NESSUS vulnerability scanner and the associated NASL scripting language as a means for efficient security management of small to medium size organizations and enterprises. This framework is based on the superscript methodology and architecture, which has been analyzed and demonstrated through a preliminary experimental analysis.

REFERENCES

[1] G. Murali M.Pranavi Y.Navateja K.Bhargavi4, "Network Security Scanner" ,M Pranavi et al, Int. J. Comp. Tech. Appl. vol 2(6)

[2] Jim Orrill, "What Is the Difference Between Active & Passive Vulnerability Scanners?", Demand Media, Available at: http://smallbusiness.chron.com/difference-between-active-passive-vulnerability-scanners-34805.html

[3] Kavita S. Kumavat, Ranjana P. Dahake, Dr. M. U. Kharat, Overview of Vulnerability Analysis, International Journal of Emerging Technology and Advanced Engineering, ISSN 2250-2459, ISO 9001:2008 Certified Journal, Vol. 3, Issue 10, 10/2013

[4] Patrick Toomey and Greg Ose,Scanning Reality: Limits of Automated Vulnerability Scanners,Strategy: Limits of Automated Vulnerability Scanners,Oct 2010

[5] Hemil Shah "Writing NASL Scripts" URL: http://www.infosecwriters.com/text_resources/pdf/NASLHShah.pdf

[6] Jae-Hong Kim et al., Cybersecurity Vulnerability Scanner for Digital Nuclear Power Plant Instrumentation and Control Systems, CSAI '18 Proceedings of the 2018 2nd International Conf. on Computer Science and Artificial Intelligence, Pages 463-467 , 2018