Even Hackers Deserve Usability: An Expert Evaluation of Penetration Testing Tools

Michael Bingham, Adam Skillen, and Anil Somayaji
Carleton Computer Security Lab
Carleton University
Ottawa, Canada
{mbingham, askillen, soma}@ccsl.carleton.ca

***Abstract*—Penetration testing is a necessary task to prevent or mitigate network intrusion. System administrators often use various penetration testing tools to aid in testing their networks; systems administrators, however, often do not have significant security expertise. It is thus important that penetration testing tools be usable by non-security experts. Here we examine the extent to which two commonly used penetration testing tools, Nessus and Metasploit, are usable by non-experts using a *heuristic walkthrough*. We identify pitfalls in user interface design, software configuration, and user notification which may hamper a non-security expert's ability to use such tools effectively. We propose user interface improvements to address issues identified by our evaluation. We also report on the efficacy of the domain-specific heuristics we selected for penetration testing usability.**

***Keywords*—Usable Security; Expert Evaluation; Administrative Tools**

I. Introduction

Software usability evaluations typically consider the *average* computer user. In contrast, systems administrators are often regarded as a special breed of user that possesses limitless technical knowledge and skill. These idealized administrators should be able to overcome any software shortcomings that would otherwise frustrate or prevent the average user from completing required tasks. As such, IT management tools are often not designed with usability in mind.

Experienced system administrators will often be experts in the use of the tools they use on a regular basis. Infrequently performed tasks, however, will require sysadmins to use tools for which they have little to no expertise. Even worse, these tools will often encapsulate domain-specific knowledge beyond the regular experience of sysadmins. A particularly significant example of such an infrequent task is penetration testing.

With penetration testing, system administrators or security professionals simulate attacks to discover vulnerable systems and network components [11]. While full penetration testing (e.g., in the form of "red teams") is a complex, specialized skill, system administrators must often do limited penetration testing in order to determine whether their systems are vulnerable to specific high-profile attacks. If their systems are vulnerable they may have to drop everything and implement emergency remediation strategies; if not, they can ignore the issue and continue on with their normal tasks. Because penetration tools are used infrequently but in circumstances where errors can be critical, we assert that commonly used penetration testing tools should be as usable as possible by system administrators who are neither experts in these tools nor in computer security.

Building upon system management design principles in the literature (e.g., [2], [5], [3]; see Section III) we evaluate the usability of two commonly used penetration testing tools, Nessus [12] and Metasploit [8], from a systems administrator's point of view. We carry out a *heuristic walkthrough* [10] with two evaluators. The evaluators follow a typical penetration testing use case (see Section II). We use the Carleton Computer Security Lab (CCSL) [1] as our testbed. The CCSL has a small network with roughly a dozen servers. Despite the small size, the CCSL network does provide most of the components and services one would expect to find in a production network from a large organization (e.g., remote login, web applications, identity management, authoritative domain name services).

Our contribution here are as follows. To the best of our knowledge, this work is the first usability analysis of penetration testing tools as well as being the first usability evaluation of Nessus and Metasploit. We propose several changes to both tools to improve their usability. We also develop a set a heuristics for evaluating penetration testing tools based upon standard approaches to penetration testing. Our results indicate that the heuristic walkthrough is a viable technique for usability evaluation of security management tools and that our heuristics can help identify usability issues.

The rest of this paper proceeds as follows. We first describe background on penetration testing in Section II. Section III outlines our evaluation methodology including our chosen evaluation heuristics. (The specific tasks used in each walkthrough are described in the Appendix.) Section IV presents the results of our evaluation, and Section V presents our recommendations for improving the user interfaces of both tools. We discuss the utility of our approach in Section VI; Section VII concludes.

II. Background

Penetration testing [11] involves discovering exploitable vulnerabilities in a target system, service, or network. The first part of the process involves identifying vulnerabilities. Due to the high probability of false positives, the vulnerabilities should then be verified before wasting time and effort by addressing problems that do not exist.

Many papers have been published about penetration testing design and methodology (e.g., [7], [6], [11]). However, most such publications assume the audience is already familiar with the fundamentals of penetration testing, mostly discuss innovation in technique, and do not take into consideration design aspects that would affect the usability for the end user.

A number of tools exist (e.g., Nessus [12], Nexpose[1] and OpenVAS[2]) to perform the vulnerability scanning. These tools perform discovery steps (typically port scans, DNS queries, user enumeration, and OS/service fingerprinting) then check the discovered services against databases of known vulnerabilities. To verify the true positives, exploits can be researched in public databases of known exploit payloads. Alternatively, tools such as Metasploit [8] can be used to automate vulnerability verification. After verifying true positive vulnerabilities, the administrator must then determine which assets are most valuable and assess the cost of providing reasonable protection for those vulnerable systems.

While there has been extensive work in the usable security, to our knowledge there has been no previous work in evaluating the usability of penetration testing tools, and indeed there is very little work on the usability of system administration tools in general. Most work in usable security centers around user studies. System administrators do not lend themselves to user studies, however, because of the difficulties in getting a large enough and representative subject population; thus, we need alternative methodologies. While ethnographic studies are useful for evaluating more commonly used system administration tools [4], we believe there is a need for less resource-intensive methods for evaluating the usability of tools for systems administration. We address this question further in the discussion.

III. Methodology

We chose to evaluate the Nessus [12] vulnerability scanner along with the Metasploit [8] exploitation automation engine. These tools are well tested and have wide spread industry adoption. We tested the free or *community editions* of these tools with their provided web-based front-ends. In the case of Metasploit, we registered for a free *professional edition* trial, to assess features that were unavailable in the free edition.

Our testbed, the CCSL network, has 128 dedicated (publicly routable) IP addresses. The CCSL network is divided into two subnets, with unrestricted access to the public Internet through the Carleton University network backbone. The CCSL runs roughly a dozen servers and administers a dozen permanent workstations (mostly Ubuntu GNU/Linux and Apple MacOS). Several other networked devices (e.g., printers) also interface with the CCSL network. All network infrastructure devices (e.g., switches, routers, firewalls) are locally administrated. Additionally, CCSL provides a wireless access point and virtual local network access from some other Carleton University subnets. Despite the small size, the CCSL does provide all the components and services one would expect to find in a production network from a large organization. As such, we feel that the CCSL provides a realistic environment for penetration testing by systems administrators.

Our expert evaluation involves two evaluators to assess the usability of both selected software components. The evaluations take the form of a heuristic walkthrough [10]. The motivation of a heuristic walkthrough is to bridge the gap between the cognitive walkthrough and heuristic evaluation usability assessment methods.

[1]http://www.rapid7.com/products/nexpose/

[2]http://www.openvas.org/

A cognitive walkthrough follows a strict structure, guided by scenarios or use-cases. The evaluator usually assumes the persona(s) of the expected end user(s), while following the steps outlined in the scenarios. During the evaluation, the evaluator is focused on four basic questions related to user interface usability:

1) Will users know what they need to do next?
2) Will users notice that there is a control (e.g., button, menu) available that will allow them to accomplish the next part of their task?
3) Once users find the control, will they know how to use it (e.g., click on it, double click, pull-down menu)?
4) If users perform the correct action, will they see that progress is being made toward completing the task?

The main criticism of a cognitive walkthrough is that it is too tightly structured (i.e. evaluators are unlikely to find issues that arise outside of the provided scenarios).

The counterpart to the cognitive walkthrough is the heuristic evaluation. Unlike the cognitive walkthrough, the heuristic evaluation is not structured based on tasks. Evaluators are expected to explore the software freely and identify any issues they discover. Instead of standard user interface-focusing questions, evaluators are provided with a set of heuristics to identify shortcomings. These heuristics are often domain specific (and hence identify problems that are outside the scope of the cognitive walkthrough focusing questions). The main criticism of heuristic evaluations is that it is not focused enough: evaluators may spend too much time on parts of the software that the user will rarely ever see.

The heuristic walkthrough uses a two-pass approach: the first pass follows the cognitive walkthrough and the second pass follows a heuristic evaluation. It is expected that the evaluator will benefit from the first pass by learning the typical interactions between the end user and the software. The evaluator learns the expected use cases while identifying general usability issues, in the first pass. The second pass is free-form, in the spirit of the heuristic evaluation. However, at this point the evaluator is familiar the intended use-cases of the software and can focus the heuristic evaluation on these common tasks.

A. First pass—task oriented

The specific use cases we evaluate are based on the two general theoretic approaches to penetration testing: attack tree modelling [9] and flaw hypothesis [6]. From synthesizing these approaches we conclude that penetration testing generally consists of these steps:

1) Discover hosts, services, and features of the network.
2) Scan the network systems for potential vulnerabilities.
3) Test the execution of exploits for these vulnerabilities.
4) Re-evaluate what further vulnerabilities can be exploited once the system has reached a different security state.

The use-case we use follows the typical progression of a penetration test. The use case will involve two tools to automate the steps. The vulnerability scanner, Nessus [12], will perform reconnaissance, network discovery/scanning, enumeration and vulnerability identification. The exploit framework, Metasploit [8], will ensure identified vulnerabilities are true positives by attempting publicly available exploits against the target system/service. Finally, we will analyze the reported feedback to determine whether the results are comprehensive and facilitate decision making and proper action by the administrator. See Appendix A for the full list of step-by-step tasks associated with our use-cases.

The evaluators will take on the following personas during the first pass cognitive walkthrough:

- SYSTEMS ADMINISTRATOR: knowledgeable of the system but not explicitly security trained. This will help us identify the usability aspects of the security related tasks (e.g., learnability, interface usability).
- SECURITY EXPERT: proficient security knowledge but little knowledge of the system architecture. This will help us identify other aspects, such as the granularity of abstraction and capacity for producing mental models of the target infrastructure.

B. Second pass—free form

We propose the following heuristics for the evaluation of the penetration testing software. These heuristics are derived from proposed design principles outlined by Chiasson et al. [2] and Jaferian et al. [5] for usable systems management tools. We have adjusted them to reflect requirements specific to the penetration testing domain. We have divided the heuristics into three categories: those that deal with general usability principles, those that deal with the usability of the system model presented to the user, and those that deal with the usability of discovering and executing specific exploits.

C. General usability

H1: Administrators should be made aware of the required tasks (and proper ordering) that they must perform to successfully test penetration (e.g., network mapping, port scanning, user-id scraping, etc.).

H2: Administrators should be able to easily figure out how to successfully perform these tasks in the correct sequence.

H3: Administrators should be able to tell when their task has been completed, whether tests ended in success or failure, and which true-positive vulnerabilities have been identified.

D. Usability of system model

H4: Administrators should have sufficient feedback to accurately determine the current state of the environment and the consequences of their actions.

H5: Administrators should be able to form an accurate and meaningful mental model of the environment they are testing.

H6: Administrators should be able to easily examine the system from different levels of encapsulation (e.g., network, distributed services, host OS, etc.) in order to gain an overall perspective and be able to effectively plan and target attacks.

E. Usability of vulnerability identification

H7: The interface should facilitate interpretation and exploitation of potential vulnerabilities to identify true-positives.

H8: Administrators should be able to easily seek advice and take advantage of community knowledge to facilitate testing.

H9: The generated reports or feedback provided should facilitate identification of critical threats against crucial systems to aid in cost/benefit analysis for ranking corrective steps.

For the remainder of the document, we will refer to the task-based cognitive walkthrough as *pass 1* and the free-form heuristic evaluation as *pass 2*.

IV. RESULTS

This section describes our findings from the heuristic walkthrough of the two software products. We also report on the efficacy of our domain-specific heuristics.

A. Nessus

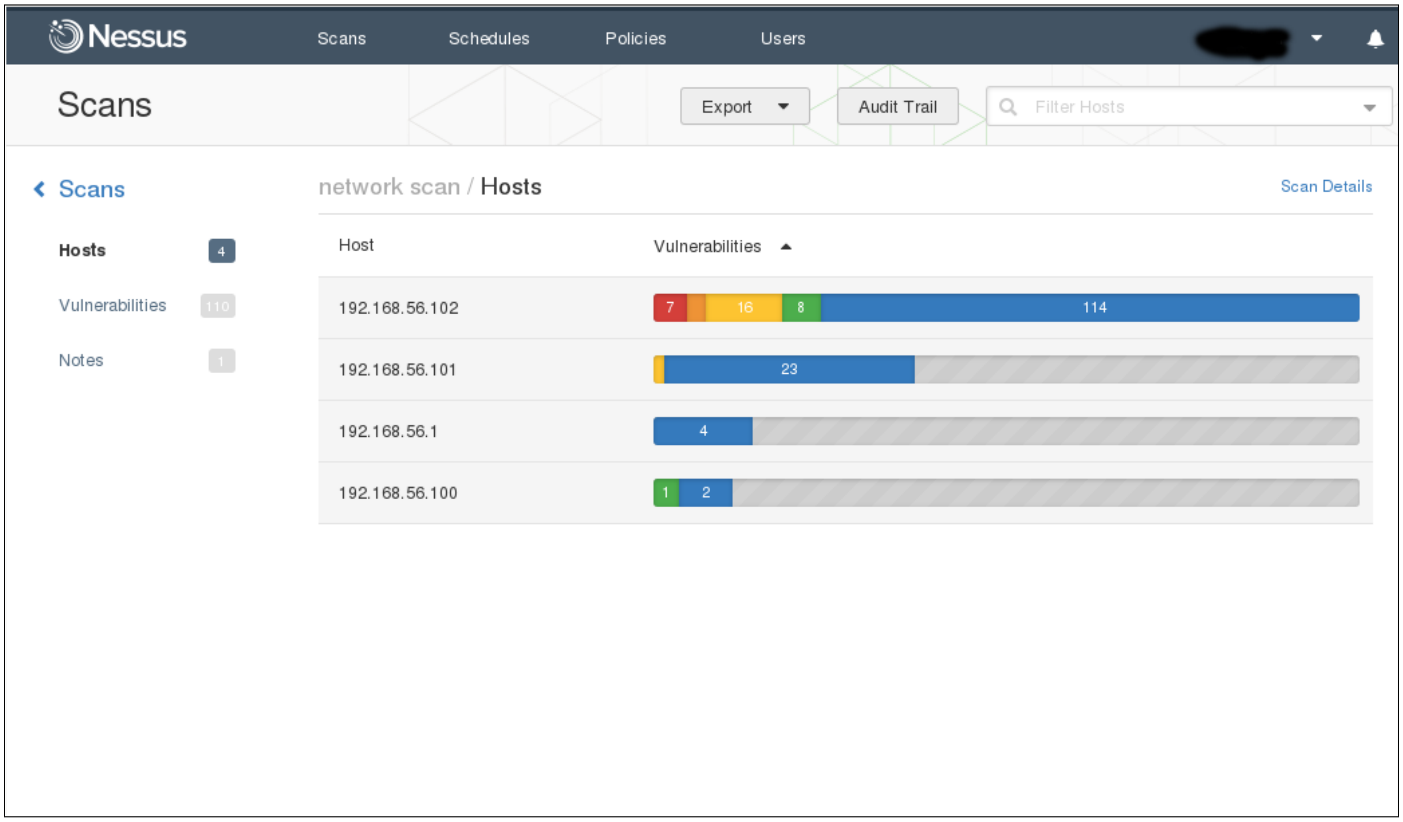
Both evaluators were relatively satisfied with the Nessus user interface. Of the two software products evaluated, Nessus had a much simpler user interface with a clear separation between performing tasks and analyzing results. This made accomplishing tasks much easier, as busy charts and tables did not clutter the screen when attempting to execute scans (i.e. all results were in the *reports* section, instead of being visible while performing other tasks).

The task of performing a scan was intuitive, and the results were displayed in such a way as to facilitate proper response from the administrator. Figure 1 shows an example of the Nessus host report. This visualization technique allows the administrator to accurately identify the most vulnerable systems, and respond accordingly.

Several user interface issues were identified, however nothing that impeded the evaluators from accomplishing their tasks. Table I shows the number of issues found during the evaluation. In total, eight issues were identified in the first pass, and ten in the second. Most of the issues found in the second pass related to specific penetration testing tasks, while the first pass revealed mostly general user interface shortcomings.

Of the discovered issues, the following were deemed the most important to correct. See Section V for our recommended remedies.

1) There is no indication that a policy must be created before a scan can be initiated. This forces the administrator to cancel the scan task to create a new policy. After creating the policy, a new scan must be created from scratch.
2) Several user interface elements have unintuitive purpose (e.g., visibility of policy). Although the system



Host	Vulnerabilities
192.168.56.102	7, 16, 8, 114
192.168.56.101	23
192.168.56.1	4
192.168.56.100	1, 2

Fig. 1. Nessus host report visualization; the left hand column enumerates scanned hosts and the right hand column displays the quantity and severity of detected vulnerabilities.

defaults are sufficient to complete tasks, input fields and controls should all have well defined purposes.

3) Exporting scan results for use with exploitation software is not clear (e.g., which file format is appropriate). The *export results* screen offers several export file formats. However there is no clear description of how the exported data formats can be used.
4) Only two abstraction levels of the system are available (either entire network or individual host). This coarse grained dichotomy may be insufficient for certain security decisions.
5) The meaning of provided vulnerability severity ratings (e.g., critical vs. high) is not clearly defined. Furthermore some screens provide numeric values to the severity (e.g., high 7.5 vs. high 8.1). There is no easily accessible definition for these ratings.
6) No guidance is provided for proper task flow (i.e. policy creation $\rightarrow$ scan execution $\rightarrow$ report generation $\rightarrow$ export results).
7) Limited support is available for searching community knowledge about identified vulnerabilities. Although a description of each vulnerability is accessible by clicking on the title, the level of detail varies for each vulnerability.

B. Metasploit

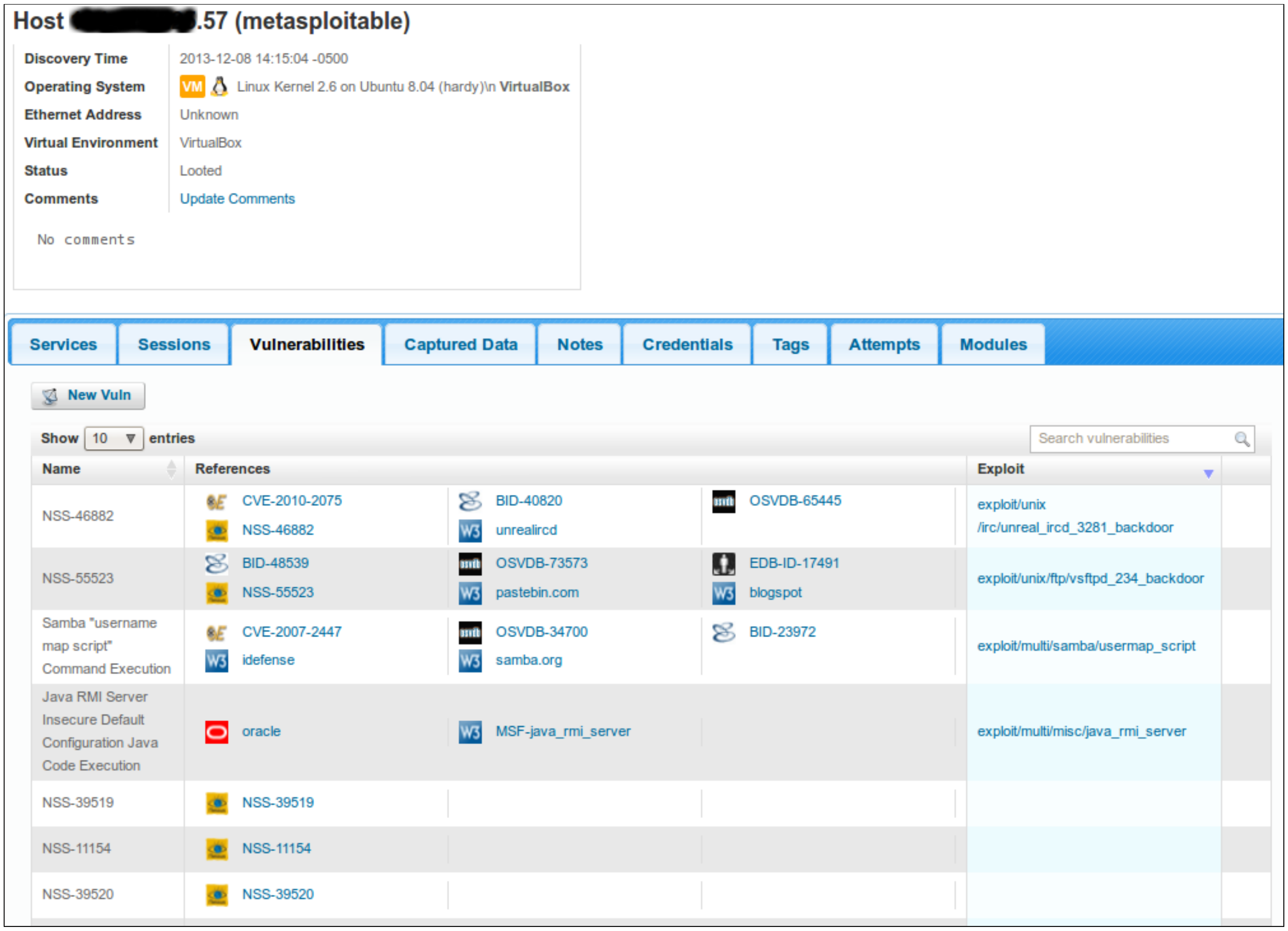
The evaluators found significantly more difficulty with the Metasploit interface. The Metasploit interface often felt cluttered, which was made even more noticeable compared to the relatively simple tasks the evaluators were trying to perform. This clutter made it difficult for the evaluators to determine how to navigate from one sub-task to another. When this issue was combined with the lack of direction from the interface, navigation became an over-arching concern.

TABLE I. NUMBER OF USABILITY ISSUES FOUND IN NESSUS. $\cup$ DENOTES THE TOTAL NUMBER OF UNIQUE ISSUES, $\cap$ DENOTES THE OVERLAP OF UNIQUE ISSUES FOUND BY BOTH METHODS/EVALUATORS.

	Pass 1	Pass 2	$\cup$	$\cap$
Evaluator 1	6	6	11	1
Evaluator 2	3	8	9	2
$\cup$	8	10	-	-
$\cap$	1	4	-	-

For example, Figure 2 shows the Metasploit host report. This visualization is much busier and less informative than the Nessus host summary. To find additional details about a specific vulnerability and known exploit payload, the administrator must navigate to a 3rd party web page. This hinders the administrator's ability to accurately identify the most important threats and respond accordingly. Additionally, information about the severity or possible consequences of an exploit are unavailable.

Table II shows a breakdown of the number of issues found during each phase of the evaluation. In total both evaluators found ten issues, with more issues being found in the second pass (13) than in the first pass (9). Combining this information, the evaluators found a total of 16 unique issues.



Host [redacted].57 (metasploitable)

Discovery Time	2013-12-08 14:15:04 -0500
Operating System	VM Linux Kernel 2.6 on Ubuntu 8.04 (hardy)\n **VirtualBox**
Ethernet Address	Unknown
Virtual Environment	VirtualBox
Status	Looted
Comments	Update Comments

No comments

Name	References			Exploit
NSS-46882	CVE-2010-2075 NSS-46882	BID-40820 unrealircd	OSVDB-65445	exploit/unix/irc/unreal_ircd_3281_backdoor
NSS-55523	BID-48539 NSS-55523	OSVDB-73573 pastebin.com	EDB-ID-17491 blogspot	exploit/unix/ftp/vsftpd_234_backdoor
Samba "username map script" Command Execution	CVE-2007-2447 idefense	OSVDB-34700 samba.org	BID-23972	exploit/multi/samba/usermap_script
Java RMI Server Insecure Default Configuration Java Code Execution	oracle	MSF-java_rmi_server		exploit/multi/misc/java_rmi_server
NSS-39519	NSS-39519			
NSS-11154	NSS-11154			
NSS-39520	NSS-39520			

Fig. 2. Metasploit host report

TABLE II. NUMBER OF USABILITY ISSUES FOUND IN METASPLOIT. $\cup$ DENOTES THE TOTAL NUMBER OF UNIQUE ISSUES, $\cap$ DENOTES THE OVERLAP OF UNIQUE ISSUES FOUND BY BOTH METHODS/EVALUATORS.

	Pass 1	Pass 2	$\cup$	$\cap$
Evaluator 1	7	7	10	4
Evaluator 2	4	8	10	2
$\cup$	9	13	-	-
$\cap$	2	2	-	-

We present here a number of the most pressing usability issues for Metasploit. Again, our suggested remedies are listed in V. These are:

1) The interface provides no direction about the ordering of tasks.
2) Once a task has completed, there is no indication of how to move on to the next stage.
3) Exploitation tasks do not directly report success or failure—the user must manually inspect the output of a console window.
4) The interface largely ignores exploit severity. For the administrator to judge the severity of a vulnerability they must navigate to a third party web page and interpret the text.
5) Though the interface provides different views of the system, some of these views leave out pertinent information—especially about vulnerabilities and current state.
6) The consequences of performing an action are not made clear. Running some exploits may cause some services to crash, for example.
7) The interface exposes many options, not all of which are available in the community edition. This leads to confusion about what options are relevant, useful or even possible for the current task.

C. Heuristics

In this section, we report specifically on the data concerning our selected heuristics. As mentioned, one of our goals is to assess the effectiveness of the proposed heuristics in aiding evaluators in finding usability problems. To this end, we have collected data on the use of the heuristics, which can be found in Tables III and IV. These tables show the number of issues found by each heuristic broken down per evaluator.

Out of the three categories, the *general usability* class of heuristics found the most usability problems (12) when counting

TABLE III. NUMBER OF USABILITY ISSUES FOUND BY EACH HEURISTIC FOR NESSUS. ∪ DENOTES THE TOTAL NUMBER OF UNIQUE ISSUES, ∩ DENOTES THE OVERLAP OF UNIQUE ISSUES FOUND BY BOTH EVALUATORS.

	General Usability			System Model			Vulnerability identification		
	H1	H2	H3	H4	H5	H6	H7	H8	H9
Evaluator 1	1	1	1	0	2	2	0	1	0
Evaluator 2	3	1	1	0	0	1	1	1	0
∪	3	1	2	0	2	2	1	1	0
∩	1	1	0	0	0	1	0	1	0

TABLE IV. NUMBER OF USABILITY ISSUES FOUND BY EACH HEURISTIC FOR METASPLOIT. ∪ DENOTES THE TOTAL NUMBER OF UNIQUE ISSUES, ∩ DENOTES THE OVERLAP OF UNIQUE ISSUES FOUND BY BOTH EVALUATORS.

	General Usability			System Model			Vulnerability identification		
	H1	H2	H3	H4	H5	H6	H7	H8	H9
Evaluator 1	1	1	2	1	1	0	2	0	1
Evaluator 2	2	1	1	1	0	0	1	1	2
∪	3	1	2	2	1	0	3	1	3
∩	0	1	1	0	0	0	0	0	0

the evaluations of both Metasploit and Nessus (six for Nessus, and six for Metasploit). As expected, this class also had the highest overlap with the first pass evaluation. This was followed by the *vulnerability identification* class with nine issues and then the *system model* class with seven issues. These classifications, *system model* and *vulnerability identification*, were the domain-specific heuristics that we expected would uncover issues that could not be captured by a cognitive walkthrough alone.

Further breaking down the numbers reveals that *vulnerability identification* found significantly more usability problems in the Metasploit evaluation than in the Nessus evaluation. The other categories had roughly equal performance across the two evaluations.

Usability problems do not seem to cluster around any individual heuristic. The only trend which stands out is that the most usability problems were found by H1. Every heuristic found at least two problems counting both evaluations, however a handful (H4, H6, and H9) did not report any usability problems in one of the two evaluations.

V. USER INTERFACE REDESIGN

Based on the analysis of the software and identified problems, we propose several improvements to the usability and data presentation of the selected tools.

A. Nessus

As discussed in Section IV-A, we were relatively satisfied with the usability of Nessus. From a non-security expert's point of view, we believe Nessus meets our requirements (i.e. Nessus has an intuitive task progression interface and provides results and visualizations that allow the administrator to respond appropriately to the vulnerabilities identified in critical assets).

We propose the following improvements to Nessus, to address the most important issues we discovered (see Section IV-A for identified problems):

1) Since there is no guidance for proper task flow, we suggest a first-use *wizard* style guide to familiarize the user with the required ordering of steps (i.e. policy creation → scan execution → report generation → export results). After first use, administrators are free to use *advanced* options for creating new custom policies/scans or re-using previously defined policies/scans in future campaigns.
2) Currently, there is no indication that a policy must be created before a scan can be initiated. We propose that policies can be created *in place* after starting the scan step. After creating the policy, the user will be returned to the scan screen (instead of re-starting the scan task, as is currently required).
3) User interface elements with unintuitive purposes (e.g., visibility of policy) should all have *tool-tip* helpers. Additionally, all **required** fields should be clearly marked as such. Since the system defaults seem reasonable for most tasks, perhaps these non-intuitive fields can be concealed in an *advanced configuration* screen. This is the current practice employed by Metasploit, and it seems to work well in their otherwise cluttered interface.
4) Exporting scan results for use with exploitation software should be made much easier, e.g., when the user clicks *export*, perhaps the first screen should ask what they intend to do with the exported data. The user would have options such as generating executive summaries, generating expert administrator reports, and exporting data for use with other software. The current interface simply lists the available export file format (e.g., HTML, NAS, PDF), with no indication of how these files can be used outside Nessus.
5) More fine-grained abstraction of the target system should be possible. E.g., reports could classify hosts based on services instead of IP addresses or names. Administrators could exercise better judgment if they knew which services were vulnerable, instead of which hosts (e.g., the web-server has 12 critical issues and the identity-provider has 3). Each site values specific services at different priorities (e.g., web hosting providers value their web servers as critical assets, while cloud storage providers value network attached storage higher than the web front-end). Administrators would benefit from an additional *service* abstraction, to facilitate easy identification of vulnerable *mission critical* assets. The administrators have the required domain knowledge to know which assets their company values most.
6) The meaning of vulnerability severity (e.g., critical vs. high) should be made abundantly clear on any

screen in which these ratings are used. E.g., *critical* severity may indicates that gaining a root shell on the host is trivial, while *high* severity might imply an attacker can progress in a given list of attacks. The meanings can be made available as tool-tip helpers on web-forms and detailed in legends on printed reports/summaries. The cryptic numeric rating system should be abandoned altogether in favour of the clearly defined categories described above.

7) Support for searching community knowledge about vulnerabilities is currently limited to certain databases (e.g., CVE[3]). In some instances, no community databases are linked against the vulnerabilities at all. Each vulnerability description has varying amounts of detail, from title only to several pages of problem description and potential exploitation attacks. All vulnerabilities should have a required minimum level of detail (e.g., type, service affected, description of exploit, consequences of successful attack). Cases in which there are no knowledge-base articles available for a given vulnerability, perhaps the system could show links to the top 5 Google results for the vulnerability.

B. Metasploit

The issues that we found with Metasploit fit into two main categories. The first of these categories is *navigation*. First, the interface does not provide direction for the ordering of tasks. After creating a project the user is dumped to a main screen where every task is possible and given equal screen space. Moreover, once the user has figured out and completed their first task, to move on to the second task they return to this main screen, again with no direction as to what comes next.

The second category is *visibility*. The interface is largely unconcerned with what information the user will find important, and does little to make that information noticeable. For example, the main goal of PT is to find and exploit vulnerabilities. However, the interface does not highlight vulnerabilities and does not give any indication of their severity.

Cutting across both of these categories is the issue of *clutter*. We discussed earlier that the interface has an abundance of elements and does little to give priority to the elements that are important. It also contains elements that seem useful for the task at hand—for example a button labelled “exploit” when viewing vulnerabilities—but which are not available in the community edition. This negatively impacts navigation, because the user sees options that would be useful but which are not possible. It also harms visibility because it increases the amount of noise in the interface, drawing less attention to the elements that are relevant.

These findings form the basis of our suggested improvements. Specifically, we suggest:

1) Removing all user interface elements which are not available in the current edition to reduce the issue of clutter. This will free up screen real estate, allowing the user interface to give prominence to more relevant interface elements.
2) Organize the main project page so that it suggests an ordering of tasks consistent with the majority of use cases (scan/import scan $\rightarrow$ vulnerabilities $\rightarrow$ exploitation $\rightarrow$ post-exploitation/data collection). Currently the tasks proceed in a semi-clockwise direction. We would suggest introducing a linear, top down ordering.
3) Provide a means to navigate from one task to the next logical task (for example, from importing a scan to viewing the vulnerabilities, or from running a data collection module to viewing the collected data). This would take the form of adding a standard forward button to the top right of each tasks view.
4) Report whether a exploitation module was successful or unsuccessful in the interface itself, rather than just in the console view.
5) When listing exploits, highlight exploits which open a session. Alternatively if a scan is imported from a vulnerability scanner, show the severity rating assigned by the scanner. We recognize that this may not be immediately possible if it requires a change in the data exchange format between the scanner and the exploit engine.
6) Highlight information about vulnerabilities in all views of the system.
7) Provide a more direct view of information about exploits. List the service they target, their severity, and possible consequences in the main interface.

VI. Discussion

Both software products seem to meet our original requirements: they both prove to be useful penetration testing tools in the hands of technically knowledgeable (non-security expert) systems administrators. Despite their obvious shortcomings, they do facilitate identification of vulnerable systems and, to some extent, aid in the prioritization of appropriate response. Based on our findings, we proposed a number of user interface improvements to the software that would further simplify penetration testing tasks for administrators. In this section, we also discuss the effectiveness of the heuristic walkthrough as a discount usability evaluation method and our selected domain-specific heuristics.

Based on the quantity, diversity, and overlap of issues identified, we believe the heuristic walkthrough is a viable *discount* assessment tactic for usability evaluation. Each evaluator was able to identify a number of unique issues during each pass. Our results indicate that many problems would not have been identified by a cognitive walkthrough or heuristic evaluation alone. In Tables I and II we can see that typically the overlap of problems found in both passes was one or two, with one instance where it was four (roughly half the usability problems). Likewise, a single evaluator would not have been sufficient to identify all critical usability pitfalls. Looking at the same tables, the overlap of problems found between the two evaluators tells a similar story—generally only a handful, at most approaching half. We believe that these observations strengthen the case for heuristic walkthroughs as a strong alternative usability evaluation method. This is especially true in situations, such as

[3]https://cve.mitre.org/

security administration, where it may difficult to find a large, representative sample of the user population.

Based on the number of issues identified by each heuristic (see Tables III and IV), we can draw certain conclusions about the usefulness of our domain-specific heuristics. The conclusions, however, should not be overstated as they are only based on one instance of use.

Our first class of heuristic (*general usability*) was most successful in identifying problems. This came as a surprise to the evaluators, as we had expected that more specific heuristics would prompt the evaluator to uncover a larger class of problems. However, closer analysis shows that the most specific heuristics (those dealing with *vulnerability identification*) actually performed slightly better than the *system model* heuristics, which we consider to be at a middle level. The most specific heuristics uncovered a significant amount of problems with the Metasploit interface. This could be for several reasons. First, the Metasploit interface may have more usability problems generally than the Nessus interface. Second, the heuristics may be targeted at a more active use case (notably H7 which deals with the active exploitation of systems). In fact the data shows that H7 found more usability problems in Metasploit (which is geared towards active exploitation) than in Nessus (which provides a more passive scanning service).

It is encouraging that problems did not have a significant bias towards or away from any individual heuristic. This suggests that no one heuristic was useless, and so we would not remove any heuristic from the set.

Overall, we believe that our proposed heuristics are a promising direction. We have increased confidence in the usefulness of higher level, more general heuristics to apply to this domain, however we also note that the specific heuristics have helped the evaluators uncover significant problems.

VII. Conclusion

IT security management tools are often overlooked, when considering software usability. Systems administrators are considered highly technical users that can cope with user interface issues which would otherwise discourage average computer users. We posit that, while highly technically trained, systems administrators are not necessarily security experts. Therefore, while poor user interface design may not hinder their ability to complete tasks, they require detailed security information and useful visual cues to make proper security decisions. We conducted an expert evaluation, in the form of a heuristic walkthrough, on the usability of penetration testing tools. Following the typical steps a systems administrator would undergo, we discover several impediments that hinder the accurate assessment of the network state. Based on our findings, we present several potential improvements of the software user interfaces which will enhance usability and decision making. Additionally, we find the heuristic walkthrough to be a valuable tool for developing security management interfaces. Our selected heuristics, although helpful in our evaluation, require further refinement for practical deployment. We hope that this evaluation will contribute to the sparse field of security management usability.

VIII. Acknowledgement

The authors would like to acknowledge the assistance of Sonia Chiasson, who provided background and feedback on this work as part of a graduate seminar on usable security.

References

[1] Carleton University. Carleton Computer Security Lab, Nov. 2013. https://www.ccsl.carleton.ca/.

[2] S. Chiasson, P. C. van Oorschot, and R. Biddle. Even experts deserve usable security: Design guidelines for security management systems. In *SOUPS Workshop on Usable IT Security Management (USM'07)*, Pittsburgh, PA, 2007.

[3] P. Duez and K. J. Vicente. Ecological interface design and computer network management: The effects of network size and fault frequency. *International Journal of Human-Computer Studies*, 63(6):565–586, December 2005.

[4] E. M. Haber and J. Bailey. Design guidelines for system administration tools developed through ethnographic field studies. In *Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology (CHIMIT)*. ACM, 2007.

[5] P. Jaferian, D. Botta, F. Raja, K. Hawkey, and K. Beznosov. Guidelines for designing it security management tools. In *Computer Human Interaction for Management of Information Technology (CHiMiT'08)*, San Diego, CA, 2008.

[6] R. R. Linde. Operating system penetration. In *National Computer Conference and Exposition (AFIPS '75)*, Anaheim, CA, 1975.

[7] J. P. McDermott. Attack net penetration testing. In *Workshop on New security paradigms (NSPW'00)*, Ballycotton, Ireland, 2000.

[8] Rapid7. Metasploit, Nov. 2013. Version 4.5.2 http://www.metasploit.com/.

[9] B. Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, December 1999.

[10] A. Sears. Heuristic walkthroughs: Finding the problems without the noise. *International Journal of Human-Computer Interaction*, 9(3):213–234, 1997.

[11] N. Shrestha. Security Assessment via Penetration Testing: A Network and System Administrator's Approach. Master's thesis, University of Oslo, Norway, 2012.

[12] Tenable Network Security. Nessus vulnerability scanner, Nov. 2013. Version 5.2.4 http://www.tenable.com/products/nessus/.

Appendix

This section enumerates the use-case tasks performed during the first pass (cognitive walkthrough) of our evaluation.

A. Nessus

The following tasks were performed during the evaluation of the Nessus vulnerability scanner.

Task 1: Login
- a) Navigate to web page
- b) Enter Username/Password
- c) Click *Submit*

Task 2: Create Policy
- a) Login (see Task 1)
- b) Navigate to *Policy* page
- c) Click *Add* to create new policy
- d) Fill in policy details (or use defaults)
- e) Click *Submit*

Task 3: Scan Network
- a) Login (see Task 1)
- b) Create policy (see Task 2)

c) Navigate to *Scan* page
d) Click *Add* to create new scan
e) Select policy to use
f) Select *Scan Now* for schedule
g) Click *Submit*

Task 4: View Report

a) Login (see Task 1)
b) Start scan (see Task 3)
c) Navigate to *Reports* page
d) Click *Browse* to view results of scan
e) Navigate through hosts to see vulnerabilities

Task 5: Export Results for use with Metasploit

a) Login (see Task 1)
b) View reports (see Task 4)
c) Click *Download* to export scan data
d) Select export file type
e) Click *Save*

B. Metasploit

The following tasks were performed during the evaluation of the Metasploit exploitation engine.

Task 1: Login

a) Navigate to web page
b) Enter Username/Password
c) Click *Submit*

Task 2: Create new project

a) From main page, click New Project
b) Enter name, description, and network range
c) Click *Create*

Task 3: Import Nessus scan results

a) Create new project (See task 2)
b) From main page, click on project name
c) Click *Import* button under Discovery heading
d) Select Nessus NBE report
e) Click *Import Data*
f) Click *Back to Task List*

Task 4: Start manual scan

a) Create new project (See task 2)
b) From main page, click on project name
c) Click *Scan* button under Discovery heading
d) Click Initiate Scan

Task 5: See vulnerabilities of a specific host

a) Create new project (See task 2)
b) Scan/import vulnerability data (See tasks 3 & 4)
c) From main page, click on project name
d) Click on *Analysis*
e) Click on the specific host
f) Click on the vulnerabilities tab
g) View vulnerability results

Task 6: Exploit Vulnerabilities

a) Create new project (See task 2)
b) Scan/import vulnerability data (See tasks 3 & 4)
c) From main page, click on project name
d) click *Exploit* under Penetration heading
e) click *Exploit*
f) View exploitation results

Task 7: Generate Report

a) Create new project (See task 2)
b) Scan/import vulnerability data (See tasks 3 & 4)
c) Exploit vulnerabilities (see task 6)
d) From main page, click on project name
e) click *Reports*
f) click *Standard Report*
g) click *Generate Audit Report*