Towards Improving Productivity in NMap Security Audits

Jose Manuel Redondo[1,*] and Daniel Cuesta[2]

[1]*Computational Reflection Research Group, Department of Computer Science, University of Oviedo, Science Faculty, Office 240, C/Federico Garcia Lorca S/N, 33007, Oviedo, Spain*
[2]*Computer Network Attack (CNA), S2Grupo, Valencia, Spain*
E-mail: redondojose@uniovi.es
[*]*Corresponding Author*

Received 19 February 2019; Accepted 18 September 2019;
Publication 04 October 2019

Abstract

Maintaining an adequate security level in computer infrastructures, like Internet-facing web servers, requires periodic assessment of their vulnerabilities with specialized security tools. `nmap` is arguably the most popular one, due to its versatility, powerful features, and low resource usage. However, this versatility can turn its usage difficult and error-prone, as it implements a lot of features and reports errors at runtime. This can lead to suboptimal results while designing auditing tasks. This research aims to decrease this complexity by developing a web GUI that favors experimentation, on-demand scans, and provides solutions to several shortcomings detected in the official one. We complemented it with a Domain Specific Language that implements early detection and reporting of syntax, type, and semantic errors when creating audit tasks. Both expand `nmap` possibilities, creating robust, schedulable, distributable, and portable auditing tasks able to find anomalies analyzing their output. Our initial release shows that the web GUI has been well received by several security related media and professionals. The language can detect and report a wide range of potential errors, substantially increasing the robustness of the created tasks. Therefore,

Journal of Web Engineering, Vol. 18_7, 539–578.
doi: 10.13052/jwe1540-9589.1871

Domain Specific Languages with early detection of type errors turned to be suitable to lower the complexity and expand the usage possibilities of complex tools like `nmap`.

Keywords: `nmap`, web GUI, advanced features, productivity, Domain Specific Language, static type checking.

1 Introduction

Attacks to different types of computer system infrastructures [1] are critical problems. Their number increases each year [2]. These attacks usually take advantage of subtle implementation details of protocols and services [3]. They may be performed over private companies, public institutions, or military infrastructures [4]. Consequences may be varied: steal user information [5], alter the normal behavior of services [6–8], data hijacking [9], or taking control of an infrastructure to perform malicious activities [10]. Both web applications and their hosting servers are very common attack targets [11].

The security status of computer systems can be evaluated through audits performed by professionals (called *pentesters*). They find vulnerabilities applying the same techniques that malicious attackers may use, but on controlled environments and agreeing to certain limitations. Found problems are written to a report, detailing their causes and potential solutions.

Pentesting activities require specialized tools. The `nmap` network analysis tool [12] is arguably the most popular and widely used one. It was created in 1997 to discover hosts and services on computer networks (thus building a "Network Map"). Its user base and popularity greatly increased when it was ported to all major operating systems. It also won the *Linux Journal Editor's Choice Award* in 2001 [13] and appeared in numerous news media. It is also part of widely used automatic vulnerability discovery tools.

`nmap` capabilities have increased over the years. Nowadays, it is a very flexible and powerful security tool that performs a wide variety of security tests, beyond typical service type and version discovery. It has a scripting engine (*Nmap Scripting Engine* or NSE) and an extensive and customizable library of scripts [12]. These enable an ample set of advanced and specialized security tests in a wide range of scenarios. For example, it is possible to establish the geographical location of scanned targets, locate malware-spreading hosts via the *Google Safe Browsing* API, locate active e-mail accounts, or perform web server specific testing. It also supports a wide range of scanning techniques and integrates firewall/IDS evasion features, like the popular *zombie scan* technique.

Unfortunately, this turns using its full potential more difficult, especially for students or users with little experience. It has 4 ways of specifying targets, more than 110 options (divided in 10 categories), multiple option combinations (some mutually incompatible) that achieve different effects, and more than 600 standard NSE scripts [12] (it also supports adding third-party ones). Detailed information about `nmap` options is not provided when running the tool from the command-line. Additionally, the official GUI (*Zenmap*) lacks information about options (and their correct parameter values) to facilitate performing quick on-demand scans (see Figure 1, part 1). More information is provided if designing a custom scan profile, but this requires time, advanced knowledge about the tool options, correct parameter values, and possibilities. This could lead to missing or misusing some important options when creating non-basic on-demand scans, leading to suboptimal results.

Solutions to simplify and automate complex operations with infrastructures not only comprise improving GUIs. The rise of the cloud as a deployment platform for public HTTP-based services has also popularized tools that allow deploying infrastructure as code, such as *Ansible*[1] or *Puppet*[2]. These products define a Domain-Specific programming Language (DSL), so their programs can be translated to specific actions in the target infrastructure, installing, updating, configuring or managing software in any way the

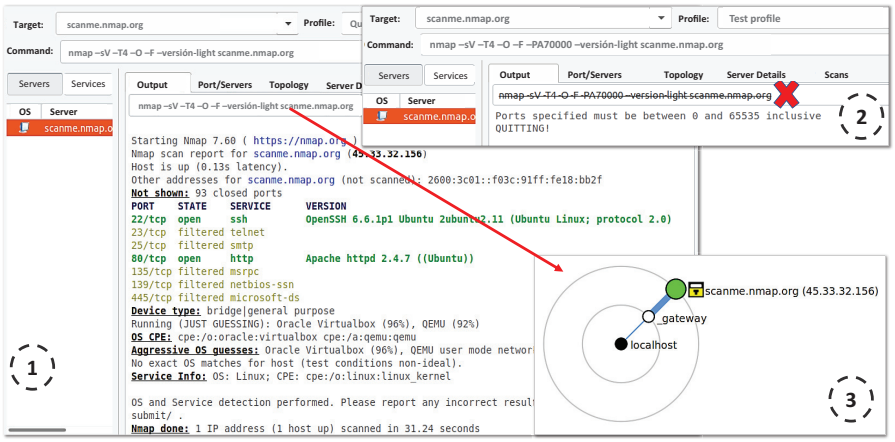


Figure 1 Different *Zenmap* scan outputs.

[1]https://www.ansible.com/
[2]https://puppet.com/

user will specify. The same approach is followed by the popular container management software *Docker*[3] with its `Dockerfiles`.

One of the main advantages of this approach is that these programs can be easily distributed and used in multiple OS, as they are small plain text files. This way, a single program can be applied to multiple machine instances fulfilling the same role provided by modern cloud-based services. This substantially simplifies infrastructure configuration. However, the lack of early validation of these programs is arguably the main drawback of this approach: these DSLs are typically dynamic, so validation is performed mostly at runtime, while the infrastructure is being deployed by executing the program commands. This means that if one instruction has errors (for example, by using parameters of the wrong type), the user could be forced to run the program multiple times, until all errors are fixed.

`nmap` scans also suffers this lack of early validation problem. The syntax and parameters of its options is usually very different from each other, and therefore prone to mistakes. Errors are always reported when the scan process is run (see Figure 1, part 2). This way, users might as well be forced to run a scan several times until its configuration is correct, decreasing their productivity. The contribution of our research is focused on facilitating `nmap` usage by:

1. Designing a new GUI (*NMapGUI*) that answers the following research questions: can a GUI offer detailed information about all `nmap` features in on-demand scans, and facilitate using its full potential? Can it also facilitate the usage of advanced `nmap` options, or implement additional features (such as scan scheduling or output analysis), without requiring deep `nmap` knowledge?
2. Creating a new DSL (*NMapDSL*) that models and facilitates creating and managing `nmap` scan tasks with the same advantages of popular infrastructure-management DSLs: distribution, portability, and easy application to multiple targets. Our research also aims to substantially decrease lack of early validation problems, identifying and reporting task errors prior to its execution. Therefore, the final research question is if modern language processors design techniques can be used to provide early validation and debugging of `nmap` option syntax, parameter types, and semantics, decreasing runtime errors.

`nmap` scan tasks designed with *NMapGUI* can generate *NMapDSL* code. Both products can run independently. Their features have been created

[3] https://www.docker.com/

focusing on teaching scenarios, encouraging knowledge transfer from experienced users to beginners, and providing verbose information about their operations. We will review the design principles, features, usage scenarios and planned future functionality, and how a part of our research was received by security-related media and users.

The rest of this paper is structured as follows. Section 2 describes *NMapGUI*. First, it details the shortcomings of the official GUI (Subsection 2.1); Then, it details *NMapGUI* main design features (Subsection 2.2), and UI design patterns (Subsection 2.3). The web GUI architecture and implementation characteristics will be reviewed in Subsection 2.4, along with sample usage scenarios (Subsection 2.5). Section 3 describes *NMapDSL*, describing its syntax (Subsection 3.1), validation (Subsection 3.2), architecture (Subsection 3.3), and execution results (3.4). Section 4 describes the reviews made by security-related media and users to the current release of our research prototype, along with a proposed systematic validation procedure. Finally, Sections 5, and 6 detail the conclusions, future, and related work.

2 NMapGUI

2.1 Shortcomings of the Official GUI

Performing predefined on-demand scans with the `nmap` official GUI (*Zenmap*) is simple. Users must choose a *scan profile* and provide scan targets. Default *scan profiles* are a list of 10 predefined combinations of `nmap` options, determining the amount or type of information that will be obtained when a scan completes. However, unless we decide to edit a profile, *Zenmap* does not display a description about the profile goals. Its concrete options are just shown as a `nmap` command line. This way, users with little experience might not fully understand the performed operations or the type of information they obtain, as it requires a substantial knowledge of the tool. *Zenmap* shows the following information after a scan finishes (see Figure 1, parts 1 and 3):

1. The `nmap` raw output.
2. Services running in each port of every target.
3. Extended information about each discovered service and target details (such as predicted operating system (OS) type and version), depending on the chosen profile.
4. A diagram of the subnets passed through to reach the targets.

However, *Zenmap* does not facilitate the usage of the full `nmap` auditing potential in several use cases, as we will outline in the following subsections.

2.1.1 No information about individual options for on-demand scans

Zenmap predefined *scan profiles* do not cover all possible options, combinations, or scan types, just a few of common usages. So, customizing on-demand auditing tasks requires manually editing the options to be used, which is error-prone due to the lack of information about them. Each option and parameter must be manually written in the *Zenmap* command box (see Figure 1), as if running from a system console. *Zenmap* displays more detailed information if a new scan profile is created, but this requires more time, and do not fit well in quick on-demand scan scenarios. So, users frequently rely on third-party support materials to consult the effects and syntax of each option. Finding the correct option combination to create an on-demand audit task can be complex and time-consuming, decreasing productivity.

2.1.2 Limited support of NSE scripts

The NSE script library gives `nmap` a lot of flexibility, greatly enhancing its usefulness in multiple scenarios. *Zenmap* only provides a list of the installed NSE scripts (plus individual descriptions) when a new scan profile is created. The script list does not have filtering or search capabilities, so finding adequate scripts may be more difficult.

For example, a script like `http-affiliate-id`, grabs affiliate network IDs from a web page, so pages with the same owner can be identified and studied together. Crawling and analysis of web error pages is offered by the `http-errors` script, which can be used to guess important information about an audited system, or the technologies used in their hosted webs [14]. Several scripts also target specific hardware types or vendors: `broadcast-bjnp-discover` discovers *Canon* printers in a network. Others detect backdoors in certain devices, as the `http-dlink-backdoor` script does with certain *D-Link* router models.

Not using adequate scripts could drastically change the outcome of a security audit, especially when users have limited experience. For example, not using the `http-dlink-backdoor` NSE script may leave some important vulnerabilities undetected, throwing false negatives, and not taking advantage of all `nmap` possibilities.

2.1.3 Limited support of parallel scan tasks

The execution state of simultaneous scan tasks cannot be viewed on real time unless multiple *Zenmap* instances are run. Active scan tasks can be accessed through the “Scan” tab (see Figure 1, label 2), but only one can be viewed at

a time in a single window. This potentially decreases productivity, forcing the user to navigate through different windows to know the status of each scan.

2.1.4 No scan scheduling or result analysis support

`nmap` does not implement scheduling or output analysis with reporting features. These have to be performed by third-party tools. This lowers its usefulness in some pentesting scenarios, forcing the user to search or create tools when, for example, a low-cost surveillance method is needed, such as in IoT networks.

2.2 Main *NMapGUI* Design Features

One of the major problems we faced when designing the GUI is that we could not follow a systematic approach involving users. Unfortunately, we could not find potential target users that have adequate knowledge of `nmap` and consistently commit to the project development during the GUI creation phase. So, we were unable to create questionnaires for different potential user groups regarding functionality and UI design, to obtain feedback before creation. Shortcomings of the official GUI (Section 2.1) were identified using our own experience with `nmap` and reading anonymous opinions in specialized forms. These information sources were also used to define the features of *NMapGUI* we will describe in the following subsections.

Having no initial feedback from potential user profiles increases the risk of creating an unsuccessful GUI design. We verified that this was not the case using the feedback provided by expert reviewers and users after its initial public release (see Section 4). However, we admit that using a systematic approach instead would have been more adequate.

2.2.1 Favor on-demand scans and experimentation

NMapGUI was designed to prioritize experimentation and creating on-demand scans that can be optionally saved later. Opposite, *Zenmap* is more focused in creating custom scan profiles. *Zenmap* provides much more complete `nmap` usage information when creating/editing profiles (see Section 2.1), which in turn requires a deep knowledge of `nmap` options. *NMapGUI* displays the full range of `nmap` options divided in four areas for immediate use (see Figure 2):

1. *Host discovery*: related to finding different types of targets.
2. *Port scanning* (shown in Figure 2): enabling users to find services running at concrete ports in the targets.

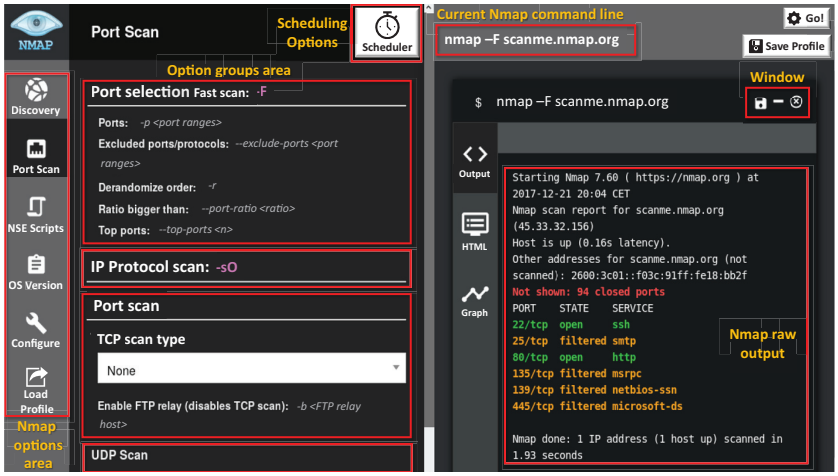


Figure 2 `nmap` UI design and `nmap` output.

3. *Usage of NSE scripts* (shown in Figure 3): that facilitates using NSE scripts in scanning tasks.
4. *OS and service version detection*: related to the detection of OS/service types and versions running on the targets.

A fifth option area ("Global scanning options") contains those that change the behavior of every scan. Inside each area, *NMapGUI* groups options by purpose (Figure 2). Each option has a short description and a syntax example to decrease the probability of accidental errors and needing external information sources.

NMapGUI favors interactivity allowing users to create scan tasks only with GUI elements: clicking on an option automatically adds it to the command line to be executed. If the option is already included, it is removed. However, user feedback (see Section 4) revealed that manually customizing the command to be run was requested, even if part of it was specified using GUI elements. Consequently, *NMapGUI* allows writing `nmap` options directly in the command section, enabling combined manual- and click-based command construction. Pressing the "Go!" button runs the current command as a background task, so a new scan can be immediately built and run (see Figure 2).

2.2.2 NSE scripts support

NMapGUI organizes installed NSE scripts in a category tree, depending on their main purpose (see Figure 3). A keyword search engine facilitates finding

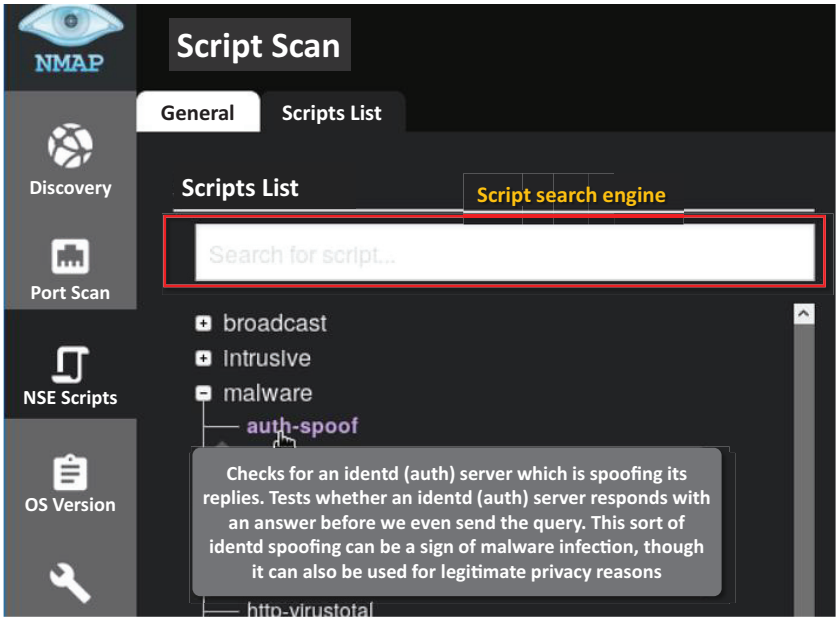


Figure 3 NSE script information in *NMapGUI*.

them by description, obtained from its official documentation and shown as a tooltip. This way, users have information to decide if a script is suitable or not, decreasing the possibility of consulting external information sources. This is aimed to facilitate the work in learning environments.

2.2.3 Parallel execution of scanning processes

Users can run multiple instances of `nmap` with different option sets simultaneously within the same GUI instance. Every scan task runs in a different thread, and its output is displayed on an independent minimizable window at real-time (see Figure 2). This way, users may increase their productivity by easily running and supervising multiple scans at the same time, especially if the scans take a long time to complete (as those using IDS evasion techniques).

2.2.4 Scan scheduling and anomaly reporting

Periodically running `nmap` auditing tasks over machine infrastructures, analyzing anomalies in their output, can be a very useful and low resource consumption surveillance practice. An anomaly can be defined as a condition that target machines fulfill when they should not to, or vice versa. For example, if a web server is expected to have the ports 22 (`SSH`), 80 (`HTTP`)

and 443 (`HTTPS`) open only, presence of additional open ports is anomalous. This might indicate a simple misconfiguration, but also an ongoing data exfiltration process from a compromised machine or web site. Also, finding these ports closed or unresponsive may indicate that a machine is down, or its services have been interrupted. This allows users to detect problems earlier, when the damage may not be very high, or quickly develop corrective responses without further compromising the infrastructure. Using `nmap` for these tasks enhances the security level of an infrastructure with lower resource usage that more complex (albeit more versatile) threat detection software [15]. Therefore, it is easier to deploy in infrastructures with potentially low resources and/or bandwidth more easily, such as IoT networks.

The first step to create scan scheduling support is to give configured scans its own entity. To do this, we provide the ability to save scan task configurations created in *NMapGUI* as *named scan profiles* (see Figure 2), with an optional long description. This way, *NMapGUI* allows users to create a custom profile from their experience of working and testing several option combinations first. These named scan profiles are aimed to be distributed and reused, capturing the acquired usage experience. This was also specially thought to target learning environments.

NMapGUI named scan profiles are simple JSON files composed by key-value pairs. They contain the `nmap` command line to be run (except the targets) and a description of its effects, so they can be easily shared and understood. *Named scan profiles* were designed with a different design approach to *Zenmap* custom profiles: users can design them to be very granular, so they don't contain a great range of different options, modeling just particular features or usage types without requiring a deep knowledge of the tool. This way, more complex scans can be composed aggregating smaller *named scan profiles*. Using multiple *named scan profiles* in a scan task combines their options. *Named scan profiles* can also be used in *NMapDSL* programs (see Section 3).

The current release of *NMapGUI* and *NMapDSL* includes 37 different `named scan profiles`, covering a wide range of `nmap` options. Some of them use common features, like different TCP detection techniques (`TCP_IP_scan`, `TCP_SYN_scan`, ...), but others also have more advanced options. For example, there are profiles with firewall/IDS detection and evasion techniques (`detect_firewall`, `avoid_firewall_detection_packet_fragments`, ...), detection of `NETB IOS` services (`netbios_find`), or even testing specific vulnerabilities (`netbios_vuln_MS08067`).

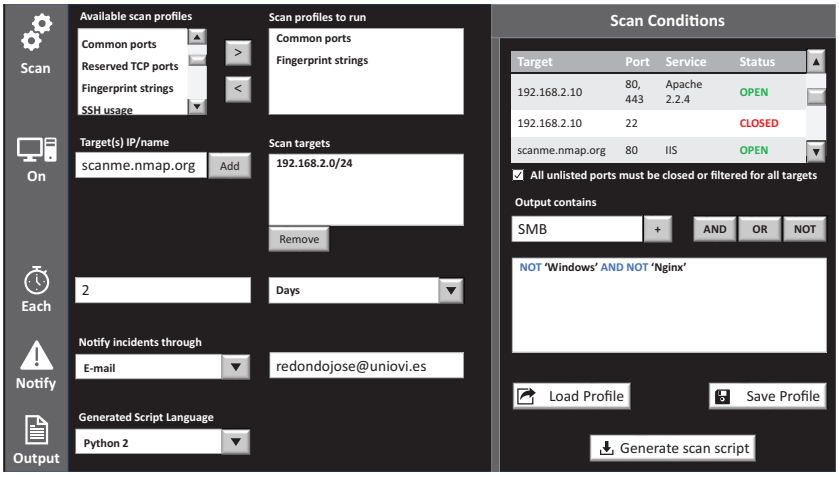


Figure 4 Current prototype of the scheduling GUI proposed to be added to *NMapGUI*.

Figure 4 shows the *NMapGUI* window that supports scheduling and surveillance features using *named scan profiles*. Several can be chosen to be run over selected targets at a certain rate. Periodicity of tasks may be specified from minutes to months, depending on the surveillance frequency requirements. Detected anomalies may be reported through email or written to a local or remote file to be consulted later. Support for anomaly detection is divided in two categories:

1. *Unwanted port states or running services*: A table specifies expected target IP/DNS names, ports, running services, and port status. The example in Figure 4 shows that the service `Apache 2.2.4` is expected to be running in ports 80 and 443 of the target IP, all of them `open` to traffic. Also, the port 22 (SSH) needs to be explicitly `closed` on the same machine. Additionally, results of the scheduled scan must also indicate that `IIS` is running in the port 80 of `scanme.nmap.org`. Service name comparison is case-insensitive, and exact substring matching will be used once trimmed. Finally, by default all unlisted ports in the scan conditions table must either not appear in the task output, be `closed`, or `filtered` to not to generate an anomaly alert. This works as a blacklist, when everything is denied unless it is explicitly granted. The user may lift this restriction.
2. *Unwanted strings in scan results*: This allows users to create simple logical expressions with strings, specifying texts that may (or may not)

appear in the scan output. The purpose of this functionality is to provide basic support to detect certain services running in environments when they should not, machines with unwanted software (OS, service types, or service versions), or just check that some services are really visible from the outside. In Figure 4, no machine should have a `Windows` operating system detected, and no `Nginx` web server must be running. Detecting these strings in the scan output will be reported as errors to the users. `OR` and `OR NOT` operators have similar semantics but reported as standard information messages.

Once the scheduling behavior is specified, *NMapGUI* can generate the corresponding *NMapDSL* program, which could be translated to scripting languages supported by the scanning machines OS (see Subsection 3.3). This also opens the possibility of running surveillance tasks with technologies like *Docker* (creating ad-hoc containers with `nmap` installed), or from automated infrastructure deployment products, such as *Puppet*. These configurations may be also saved or loaded to be modified at any time.

2.3 UI Design Patterns

As we said in Section 2.2, we also relied on our own experience and anonymous comments when designing the layout of the different GUI windows. We used expert advice to choose adequate UI patterns for each part of the GUI, but UI mockups couldn't be systematically validated by users prior to building the application.

GUI feedback was again obtained after the application was released (see Section 4). We know that we took a substantial risk, as this is not the optimal way of creating a GUI, but its high specialization restricted the potential user community we can reach, to the point that we were forced to take a different approach. As we will see, the requested UI changes were not major, and feedback from `nmap` experts has been good so far.

NMapGUI has been initially designed considering several UI design patterns [16] to facilitate its usability. As we can see in Figure 2, main UI abstraction is the *Module Tabs* pattern. This is because there is a limited visual space, and content needs to be separated into just 5 sections that need a flat navigation mode. As the content of each tab can be viewed separate from each other, section names are short, and content do not depend of the context of each other, this pattern adapts to our needs. We also prevent unnecessary page refreshing.

Regarding scan outputs, we used a *Navigation Tabs* design pattern but using the window abstraction with minimize, maximize and close buttons (so they are disposable) instead of typical tabs. This is because we will rarely have more than 9 scans running at the same time, scan output needs to fill the entire width of a page, and there is a need to single-out the currently selected scan. Regarding displaying single elements, the script category tree (see Figure 3) has been conceived to follow the *Progressive Disclosure* UI pattern while maintaining the classification of the scripts. This allows users to progressively display information about the script they are inspecting.

Finally, the `nmap` options presented in each section are based in the *Input Prompt* UI pattern. Labels of options do not fully explain what should be filled into it. Therefore, they are accompanied by sample text using a different font style so, in combination with the label, both further explain what kind of input is needed. Although we have a single editable input field (the `nmap` command line to be executed), using this approach clarifies what kind of input is expected in each option. In any case, more work in the usability is needed before the final *NMapGUI* release. We will detail part of this additional work in the future work section.

2.4 Architecture

Current version of *NMapGUI* is an open source project composed by several packages (see Figure 5, part 1) designed to be easy to understand and extend.

Executing the *NMapGUI* `.jar` file initially runs a small startup *Java* 1.8 *Swing* application. Its classes belong to the `localGUI` package and ensure that `nmap` is installed using the features of the `Executor` package. Then, it allows users to start/stop an instance of the *Spring* application implementing

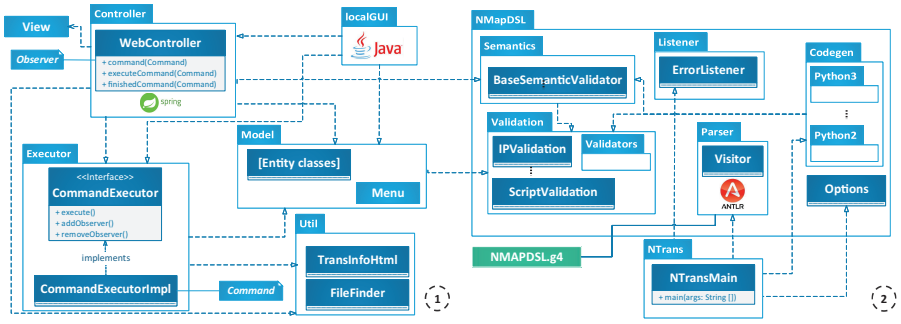


Figure 5 Architecture of *NMapGUI* (1) and *NMapDSL* (2).

the *NMapGUI* web interface. *NMapGUI* can only be used from `localhost` to prevent potential security problems via remote command execution or using the machine as a pivot to explore other networks. In our tests we used the version 1.5.3 of the *Spring* framework.

The architecture of the web application follows the MVC design pattern. The `View` is responsible of processing user inputs, passing the corresponding commands to the web server via HTTP requests. Data visualization is enhanced using the *jQuery* 3.2.1[4] and the *D3* V3[5] libraries.

Requests are managed by the `WebController` class until they are executed. When the user runs a command, the controller calls `CommandExecutorImpl`. This class is the default implementation of the command execution processes of the application, and therefore runs each `nmap` instance with the provided options. Command execution is asynchronous, to enable running `nmap` instances in parallel and to write each command output in its own window. For that reason, the `WebController` uses an *Observer* design pattern, so it gets notified every time a command finishes to update the corresponding view.

The `Model` package contains all classes representing system and scan result entities: `Address`, `Command`, `Hop`, `Host`, `Hostname`, `Port`, `Scan` or `Script`. Additionally, the `Menu` submodule contains entities representing different interface elements, like `Menu`, `Submenu`, `Category` or `Option`. Its configuration is read from an XML file so it can be easily updated.

Finally, the `Util` package contains utility classes to convert from XML to HTML through an XSLT file (`TransInfoHtml`), and to locate files that store command execution results (`FileFinder`). Code coverage of the application tests is controlled using the *jacoco* 0.7.9[6] library.

2.5 Usage Scenarios

This section describes how to use *NMapGUI* to run typical `nmap` on-demand scans as part of an auditing process. On-demand scans are more flexible that creating custom scans profiles. The user can quickly react to the information obtained by a scan, so the next will can be customized depending on the information obtained by the previous one. For example, if the first scan

[4] https://jquery.com/
[5] https://d3js.org/
[6] https://www.eclemma.org/jacoco/

detects a web server, the next ones may try different web-related NSE scripts (like the ones mentioned in Section 1) to try to discover different vulnerabilities.

The first scenario requires to check the TCP reserved ports of a remote system. To avoid potential filtering, the `TCP SYN` scan type will be used. To do that, the user goes to the "Port Scan" area and, inside the "TCP Scan Type" group, locate and click the appropriate `nmap` option for this type of TCP scan (`-sS`). This way, the user should only remember the type of port scan, and *NMapGUI* provides the necessary information about the corresponding option. Then, the command can be enhanced guessing the OS type of the target system. To do that the user goes to the "OS Version" area and click on the "OS detection" option (`-O`). As described on Section 2.2, the interface automatically adds this option to the existing `nmap` command line. Once the user finishes adding all the desired options (using the GUI or manually editing the command), clicking in the "Go!" button shows the scanning progress at real time. All these steps are illustrated on Figure 6.

When the scanning process finishes, the user may decide to further explore the remote machine. This begins the second scenario, that uses the features of the `nmap` NSE script library to enhance the results of a previous scan. In this case, the command will incorporate a script that outputs the *fingerprints* (service and versions description strings) of any service not directly recognized by `nmap`. If the name of this script is not known,

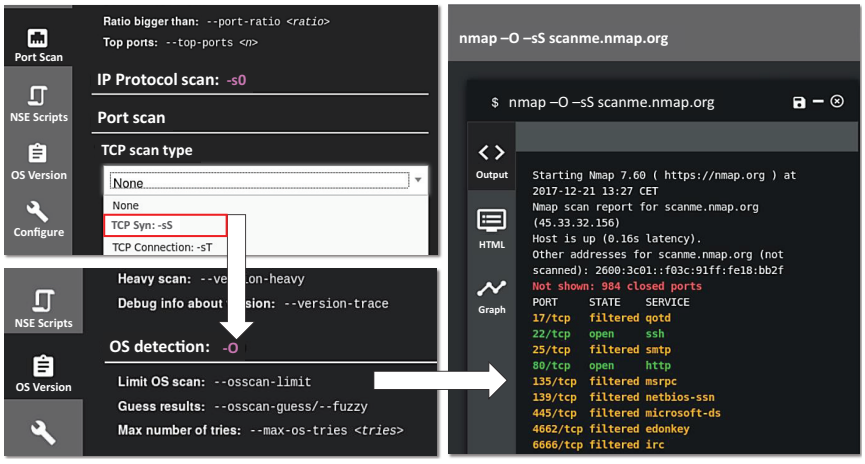


Figure 6 Typical basic scan process over `scanme.nmap.org`.

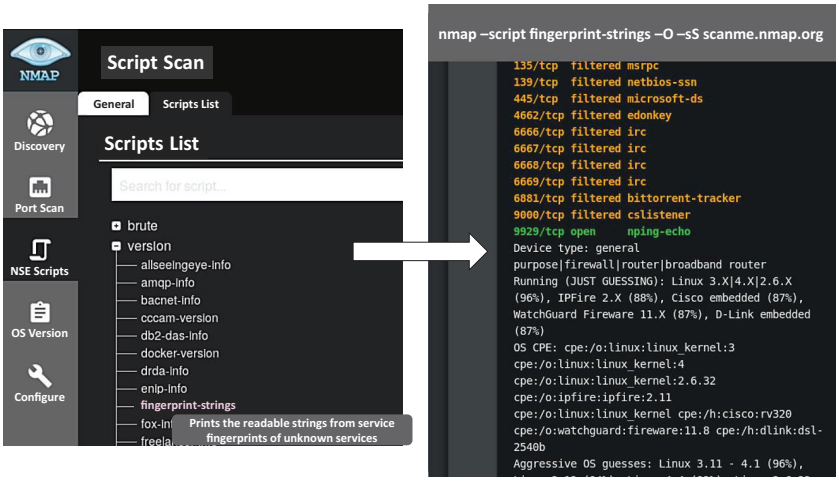


Figure 7 Scanning `scanme.nmap.org` using NSE scripts.

NMapGUI can be used to search it and perform the scan. Figure 7 shows the "NSE Scripts" area and its "Script list" tab. The desired script can be found under the "version" category (`fingerprint-strings`). Once located, the script information can be checked to ensure it is the correct one. Finally, clicking on the script adds it to the current command line, so the enhanced scan can be run again. Although *NMapGUI* preserves the previously run `nmap` command line, this is a new scan, so its output is displayed in a new independent window. Therefore, it is easy to study and compare both outputs later. Additionally, if the previous scan was still running, both scans will be executed in parallel. Figure 7 illustrates this scanning process and the final `nmap` output when executed over the same target machine as the previous one.

3 NMapDSL

As we said on Section 1, *NMapDSL* has two main design goals:

1. Provide an early validation and debugging tool for scan tasks, so errors can be detected, precisely reported, and corrected during the task design phase instead of at runtime (see Figure 1, label 2).
2. To be able to facilitate `nmap` usage on environments that typically cannot display web pages or GUIs, like servers and IoT devices.

The language is structured in blocks, each modeling different parts of a scanning process. Blocks include the scan scheduling, result notification,

and output analysis features described on Section 2.2.4. Another software product created by this research is the *NTrans* validation and translation tool, responsible of translating valid *NMapDSL* programs to popular scripting languages. These translated scripts may be moved to any machine in the infrastructure with `nmap` and the target scripting language runtime environment installed. They are also able to run stand-alone, without dependencies of the GUI components. We plan to support *Python 2*, *Python 3* (including its fastest implementation [17], *PyPy*), *Bash script*, and *Powershell* as target scripting languages to be able to run on any major operating systems. As we said, *NMapGUI* is also aimed to generate *NMapDSL* code, so it can also work as a *NMapDSL* web IDE for external machines not able to run it.

3.1 Syntax

To fulfill the requirements of our research, *NMapDSL* tries to increase user productivity being a simple and easy to understand language that facilitates performing advanced scan tasks without requiring deep knowledge of `nmap`. This is also especially important in computer security courses. The language has six different blocks, although several are optional. Line comments are supported with `#`. The main elements of each block are defined using the ANTLR[7] specification syntax in Figure 8. Sample programs are shown in Figure 9.

3.1.1 SCAN block

This block is the most important, as it specifies a list of *named scan profiles*, or a single list of `nmap` command-line options, telling `nmap` what to do. If nothing is specified, `nmap` runs with all-default parameters. Additionally, a series of *scan customizing keywords* may be optionally placed before `SCAN`, to facilitate running common advanced scan configurations without using options or profiles. These keywords do not cover the full range of possibilities of `nmap`, but facilitates users with limited experience to access advanced scan types:

- *Timing options* (`timing_scan` ANTLR rule): `PARANOID` (0), `SNEAKY` (1), `POLITE` (2), `NORMAL` (3), `AGGRESSIVE` (4), and `INSANE` (5), representing values from 0 to 5 of the `-T<number>` option.
- *Scan purposes* (`purpose_scan` ANTLR rule): keywords representing certain typical scan options, such as `STANDARD` (runs standard

[7] https://www.antlr.org/

```
main: scan_profile destination periodicity? report? with? output?;
// SCAN block
scan_profile: timing_scan? purpose_scan? evasive_scan?
SCAN_KW scan_type NS;
…
scan_type: profile_list | nmap_options;
profile_list: profile_name (',' profile_name)*;
…
nmap_options: nmap_command_line;
nmap_command_line: nmap_command_options
target_specification?;
target_specification: (input_file | random_targets | host_excluded
| host_excluded_file)+;
nmap_command_options: (host_discovery |scan_techniques
|port_spec|service_detect|script_scan|os_detect|timing|evasion
|scan_output|misc)*;
… (103 additional rules, one per each nmap option)
// EACH block
periodicity: EACH_KW period time_unit NS;
…
// ON block
destination: TARGET_KW targets port_list? NS;
targets: target (',' target)*;
target: ip_v4 | ip_v6 | cidr | dns_name;
…
// NOTIFY block
report: NOTIFYNG_KW (file_notify | email_notify) NS;
…
// WITH block
with: WITH_KW '{' (target_block)+ '}';
target_block: target '{' target_features '}';
target_features: target_feature (target_feature)*;
target_feature: (host_status | port_list_state | service_list) NS;
…
// OUTPUT block
output: OUTPUT_KW '{' logical_expression NS'}';
logical_expression: (not|text_to_search) (and | or| and_not|
or_not)*;
… (keywords and lexer rules)
```

Figure 8 ANTLR specification of NMapDSL blocks.

```
1
SCAN 'common_ports', 'fingerprint_strings'; #Named profiles
ON 192.168.2.0/24; #Entire subnet
EACH 2 DAYS;
NOTIFY EMAIL redondojose@uniovi.es;
WITH ONLY{
      192.168.2.10 { #Expected ports/services for this host only
        PORT 80, 443 OPEN;
        SERVICE 'Apache 2.2.4' PORT 80, 443;
        PORT  22 CLOSED;
      }
      … #Ports/services of additional hosts
}
OUTPUT CONTAINS {
      NOT 'Windows' AND NOT 'Nginx' ;
}

2
INSANE SCAN -O -sS ; #NMap options
ON 'scanme.nmap.org'; #Machine DNS name
EACH 2 WEEKS;
NOTIFY FILE /tmp/test.txt ;
WITH ONLY {
      'scanme.nmap.org' {
            PORT 22, 80 OPEN;
            PORT 17, 25, 135, 139,445, 4662, 6666 FILTERED;
            SERVICE 'qotd' PORT 17;
            SERVICE 'ssh' PORT 22;
            SERVICE 'smtp' PORT 25;
            SERVICE 'http' PORT 80;
            SERVICE 'msrpc' PORT 135;
            SERVICE 'netbios-ssn' PORT 139;
            SERVICE 'microsoft-ds' PORT 445;
            SERVICE 'edonkey' PORT 4662;
            SERVICE 'irc' PORT 6666;
      }
}

3
ACTIVEONLY SCAN -v;
ON 192.168.2.0/24;
EACH 2 MINUTES;
NOTIFY EMAIL redondojose@uniovi.es;
WITH ONLY {
      192.168.23.10 { UP; } #Hosts are alive
      192.168.23.25 { UP; }
      #This host must not be running
      obsolete.frontend.local { DOWN; }
      …
}
```

Figure 9 Samples of *NMapGUI* DSL programs.

scan techniques, `-sC` option), `ACTIVEONLY` (only lists hosts, `-Sp` option), `AGGRESIVE` (runs more common scan options, `-A` option), or `VERSION` (obtain versions of the operating system and running services, `-O` option). Other keywords also use popular NSE scripts to achieve useful effects, like `GEO` (geolocalizes targets using the *Maxmind* database, `ip-geolocation-maxmind` script), `WHOIS` (obtain DNS information about targets, `whois` script), `MALWARE` (uses the *Google Safe Browsing* API [18] to determine if the target is identified as a malware

distributor, `http-google-malware` script), `EMAIL_ADDRESS` (locates active email accounts, `http-google-email` script), or `WEB` (runs a series of scripts that specifically target web servers and CMSs, such as `http-methods` and `http-enum` (`nmap -sV --script=http-enum <scanned target>`)).

- *IDS Evasion* (`evasive_scan` ANTLR rule): `EVASIVE`, automatically uses multiple `nmap` options and techniques to evade firewalls or IDSs (`-f`, `--randomize_hosts`, `--badsum`, `-D RND:10`, `--spoof-mac 0`).

This way, a valid `SCAN` block may be like this: `INSANE ACTIVEONLY SCAN <profiles or NMap options>`. Scan profiles or options have more priority in case of overlapping with the keywords.

3.1.2 Other blocks

The `ON` block is used to specify targets. It accepts lists of IPs, networks (CIDR format) or DNS names. An optional `PORT` keyword allows to restrict which ports are going to be scanned.

The `EACH` block enables task scheduling functionality (see Section 2.2.4), accepting a number and a time unit (`MINUTES`, `HOURS`, `DAYS`, `WEEKS`, or `MONTHS`). If not specified, the task is run just once. The also optional `NOTIFY` block deals with scan notifications, accepting email addresses to send scan results or a file path to write them to.

The two final optional blocks deal with anomaly detection. `WITH` block describes the expected port/service distribution for targets within the range specified in the `ON` block. The `WITH ONLY` variant is similar but treats as anomalies every port found `OPEN` outside the ones specified in the script source code.

Finally, the `OUTPUT CONTAINS` block allows creating simple `AND`/`OR` logical expressions (optionally prepending `NOT`) with strings, to check `nmap` output against them. Both blocks behave as described in Section 2.2.4.

3.2 Program Validation

Unlike statically typed languages like *Java*, *C#* or *C++*, dynamically typed languages do not perform type-checking at compile time. Therefore, the probability of finding errors at runtime increases in exchange for a higher degree of flexibility. Static typing offers the programmer early detection of type errors, making possible to fix them immediately rather than discovering them at runtime [19]. All `nmap` usage errors are reported at runtime (see Figure 1, part 2). Therefore, a programming language created to model `nmap`

Table 1 Summary of validations and additional features provided by NMapDSL

Validation Types	
Script syntax validation	Detects misspelled keywords and unwanted characters
Usage of existing options	Checks that every used `nmap` option exists
Types of option parameters	Checks the type of the parameters passed to every `nmap` option
Usage of existing NSE scripts	Notifies if attempting to use a non-installed NSE script
Usage of documented script parameters	Warns if using non-documented script parameters
Validate NSE script parameter types	Provides an extension mechanism to enforce robust validation of NSE script parameter types without modifying the script source code
Semantic validation	Checks multiple conditions that can go wrong during a scan: target IPs within declared network ranges, usage of existing DNS, detection of incompatible options...
Precise error location	Locates each detected error or warning (line and column) and provides a detailed explanation of its cause
Additional Features	
Scheduling (`EACH`)	Schedules scan tasks to be repeated after a predefined amount of time
Output analysis (`WITH`)	Port, service banner detection, basic keyword search
Special keywords	Usage of advanced `nmap` options with special keywords without requiring further configuration

auditing tasks will also have this drawback, as potential type errors will be detected at runtime.

To counteract the lack of robustness due to no early type error detection, there are several research lines that developed solutions to statically detect type errors in dynamic languages [20], or when using dynamic metaprogramming features of statically-typed languages [21,22], without losing flexibility. All of them increased program robustness by enabling early detection of type errors in dynamically typed scenarios. We followed the same approach, enabling static type checking of the parameter types of `nmap` options, along with a series of static semantic validations of the values of these types. Table 1 summarizes the types of validations *NMapDSL* performs and enumerates the described additional features.

This way, the current implementation of *NMapDSL* validates the syntax of all `nmap` options: usage of malformed, misspelled or non-existing options will be reported. It also detects wrong types in their parameter values. Semantic errors, like incompatible options and IPs outside scanning ranges, are also detected and reported (see Figure 10). Warnings are also used; for

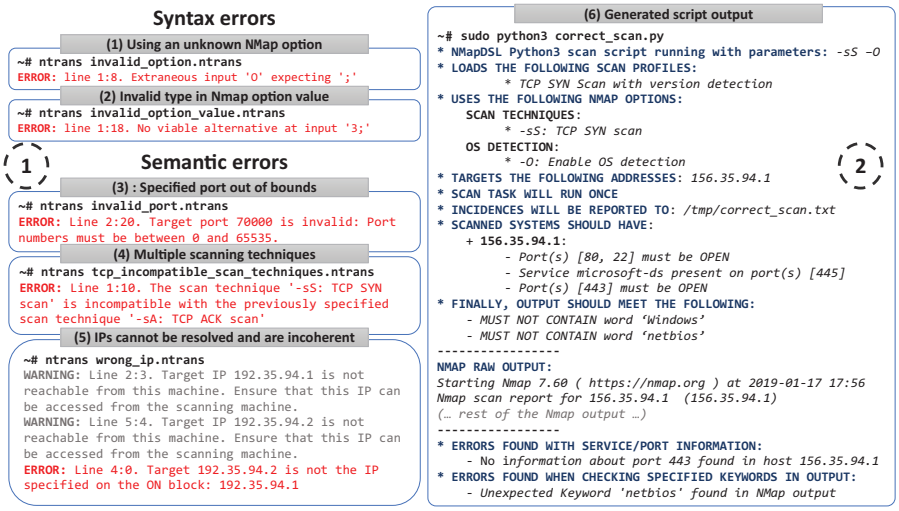


Figure 10 Output examples of script generated with *NMapDSL*.

example, when specifying unreachable IPs or DNS names, as the machine used to create the script may not be the same running it. Each error is reported providing a detailed explanation and its location (line and column). *NMapDSL* will only translate to a target scripting language those programs without errors. This way, the resulting scan tasks are less prone to have errors than manually configured ones, as *NMapDSL* offers its users a great range of early error validations. This is also useful in security courses, as the detected error information will be valuable to learn how to use `nmap` correctly.

Regarding NSE scripts validation, every script has its own parameters and acceptable value types. Unfortunately, this information is not properly documented for our needs. Script descriptions may be extracted from its source code (parsing `description` entries), and parameter names are usually documented via `@args` or `@arg` tags. However, parameter documentation is frequently incomplete, and there is no syntax to specify which are mandatory or its expected type. For this reason, *NTrans* includes a special folder `script_info` that contains JSON files specifying this information. This way, if we use the `dns-zone-transfer` script and a `dns-zone-transfer.json` file is present in this directory, *NTrans* will load the file to read what parameters are `mandatory` and its expected type (`integer`, `string`, `url`...). If the corresponding `.json` file do not exist, the script source code will be parsed, and a warning will be thrown if an undocumented parameter is used (scripts

could leave some of their accepted parameters undocumented). This way, we use the available information without compromising scan flexibility, also enabling a mechanism to provide robust script validation without modifying the script source code. The initial version of *NTrans* is supplied with some `.json` files corresponding to standard NSE scripts as examples.

As shown in Table 1, additionally to syntax, multiple semantic conditions are also checked. These are the most important ones:

- `SCAN`: `nmap` option syntax (either manually written or extracted from *named scan profiles*) is checked, throwing errors if they are malformed or the parameter types are invalid (see Figure 10).
- `ON`: an error is thrown if an invalid IP, DNS name or CIDR is specified. Additionally, a warning is reported if their syntax is correct, but it cannot be resolved through a DNS query during the translation process. This could be caused by a temporary network outage or because only the scanning machine has access to these targets (hence issuing a warning) but helps to mitigate potential script creation errors. The same type of validation is done every time an IP, CIDR or DNS is used, such as in the `nmap` command-line options of the `SCAN` block.
- `EACH`: warns when using time units higher than 1000, considering that it is way too much time for a periodic task regardless its time unit. Only unsigned integer types are accepted.
- `NOTIFY`: if an email is specified, an error is thrown if the address does not comply with the email *Internet Message Format RFC2822 Standard* [23], using the validators of the *Apache Commons* project[8]. The existence of the email domain is also checked, although only a warning is reported if not found for the same reasons we did with IPs or DNS names. If a file is specified, its path is checked, and a warning is issued if the specified file will be overwritten, again considering that the scanning machine might be different than the script creation one.
- `WITH`: validation of IPs and DNS names is identical to the described in the `ON` block. Port numbers are forced to be unsigned integers in the 1..65536 range. Additionally, coherency is checked between the targets specified in the `ON` block and the ones present in the `WITH` block. For example, if a CIDR is specified in the `ON` part, and IP addresses are used in the `WITH` block, every IP is checked to be within the CIDR range. Moreover, if the `ON` block uses a DNS name and the `WITH` block uses an IP (or vice versa), both are checked to see it they match.

[8] https://commons.apache.org/proper/commons-validator/

Some of these conditions can only be checked if a net connection is available in the machine running `NTrans`, giving a warning otherwise. Validation result examples can be shown on Figure 10 (part 1).

3.3 NMapDSL Architecture

NMapDSL architecture is also represented on Figure 5 (part 2). The first step is to automatically generate the language parser by creating an ANTLR grammar [24] (`NMAPDSL.g4` file in Figure 5) composed by several sections (see Figure 8), using the ANTLR features to automatically generate an adequate parser:

- The six described language blocks.
- Special data types of our problem domain (IPv4, IPv6, MAC address...).
- An entry for each `nmap` command-line option, grouped according to the classification used in *NMapGUI* and the `nmap` help description.
- Language keywords.
- Lexer rules.

Once the parser is generated (`Parser` package), all the translation process is controlled by the *NTrans* program (`NTransMain` class), that validates its options and reads the DSL program source file. If all is correct, it parses the script using the generated parser, configuring an `ErrorListener` (`Listener` package) to report possible errors. If program parsing is correct, it selects a target scripting language (depending on the one specified in `NTransMain` options) and runs its corresponding code generator (`Codegen` package, see Figure 5). Code generation triggers two validation processes:

1. *Additional validations* of program entities (DNS names, files, IPs, emails, network connections, NSE scripts, program options, and named scan profiles) that could not be performed by the parser. These include ensuring correct IPv4 byte ranges (`IPValidation` class), or locating and enforcing the parameter type rules associated to a NSE script (`ScriptValidation` class, see Section 3.2) among others. All the classes modeling these validation rules belong to the `Validation` package, and use the features present in the subpackage `Validators`, as shown in Figure 5, part 2.
2. *Semantic validation*. The `Semantics` package contains a series of validators, all deriving from the `BaseSemanticValidator` class. These check the semantic rules of the program entities shown in Table 1. For

example, the `ValidScanTechnique` class checks that no incompatible options are used in a configured scan like, for example, more than one TCP scan technique. Reachability of CIDRs, DNS, and IPs, and compliance of the specified targets with the network range of targets to be scanned are also checked with classes of this package. Other validations performed by classes in this package include scheduling time limits, email domain, and provider configuration (if notifications via email are used), paths of input and output files, valid port numbers, ranges, named scan profiles and NSE scripts. All these classes give the precise location (line and column) of the located errors or warnings within the source code (see Figure 10). The `Semantics` and `Validation` packages can also be accessed from *NMapGUI* to reuse the same validations.

Once this validation phase is complete, the detected errors are displayed (see Figure 10). If no error was detected, `NTrans` proceeds to write the output program with the modeled scan task in the desired target scripting language. To do that, we use several classes belonging to the `Codegen` package (see Figure 5), taking advantage that every `nmap` option has its own entry in the language *Abstract Syntax Tree* (AST) and, therefore, they can be treated individually if used in a program.

First, two additional visitors are used to extract information from the parser-generated AST tree. The first, `VisitorScanData`, collects information about the values of the different `nmap` options that are going to be used in the scan task. The second, `VisitorOptionsInfo`, gathers detailed information about the used options. The purpose of this second visitor is to provide users with precise details about the activities of the scan task if it is run in verbose mode, so they can be useful in learning environments or to have a precise log of the performed activities. Both visitors are the same to all target languages, as they gather information in a language-agnostic way, placing all the information in the `SourceGenerationData` class. An example of this detailed information is shown in Figure 10, part 2.

Finally, to generate code in the target scripting language, a class derived from the `CodeGenerator` interface is loaded in the corresponding target language package, and its `generateCode` method is called. This method loads a *script model* with the code of common functions to any generated task that is created for each target language (in Python it is called `script_model.py`). Then, joins it with the data gathered from the two mentioned visitors. Finally, code templates of different processing functions in the target scripting language syntax are added to finish the generated scan

task, thanks to helper classes placed in the same package, and loaded from this main code generation class.

3.4 Scripts Samples and Output

Figure 9 shows sample programs of the current specification of *NMapDSL*. Program 1 contains most of the scheduling shown on Figure 4: a subnet is monitored every two days to check opened ports on certain machines and ensuring that no machine reports *Windows* or *Nginx* installed. In a real scenario, this may detect unwanted servers or rogue machines. Program 2 scans the machine *scanme.nmap.org* and ensures that a list of expected port and service names is strictly observed. In a real scenario, not fulfilling this might mean that the machine has been compromised. Finally, program 3 just checks a network for a series of machines. Extra ones in the list may warn about rogue or unwanted machines (an old server is accidentally brought up) in a network. Less machines may also indicate that a server has failed or have been misconfigured and unable to run, detecting outages.

Figure 10 (part 2) shows the typical output of scripts translated by `NTrans` and examples of errors detected by `NMapDSL` (part 1). Apart from the syntax errors, the error output of using illegal port numbers (3) or incoherent/unreachable IPs (5) we mentioned in Subsection 3.2 are shown. Additionally, usage of incompatible options is also detected (4), as `nmap` cannot use multiple scan techniques in a single scan. Figure 10 (part 2) also shows the step by step detailed information provided by translated scripts about the purpose of every option and NSE script we mentioned, apart from the `nmap` raw output. All these features answer research questions that were described in Section 1.

4 Reviews and Proposed Validation Procedure

We released a preview version of *NMapGUI* implementing most of the described functionality to obtain feedback from the community. This preview version still lacks support to create or load *named scan profiles*, although the file format is defined and supported by the current release of *NMapDSL*. It does not integrate yet the DSL validation capabilities for `nmap` options and values we saw in the previous section, although the DSL do implement them. Finally, the scheduling interface is under construction, and therefore not available in the main scan screen yet.

Several security-related websites made articles reviewing this initial release of *NMapGUI* [25–30]. Reviews praise the interface design and capabilities, especially the ability to supervise parallel commands and the integrated information about `nmap` options and NSE scripts. Main concern about *NMapGUI* has been the potential security problems that may occur if a malicious user accesses the interface from an external machine, using the scanning machine as a pivot to access other networks. This is the reason why we decided to restrict connections to the GUI to `localhost` only.

User feedback obtained from social networks also shows a positive reaction, praising *NMapGUI* features. However, some users state that `nmap` should only be used manually as a command-line tool due to their complexity and number of options, as a GUI could not capture its complexity. However, mastering `nmap` is difficult precisely because its flexibility and complexity, hence the motivation of creating *NMapGUI* to lower it. Advanced users can manually modify `nmap` commands in *NMapGUI* if they wish to do so. This way, they can take advantage of the GUI features while being in full control of the command structure, as they requested. Additionally, for users that just use command-line tools, using only *NMapGUI* to create *NMapDSL* auditing tasks is a valid alternative.

A group of users that have reviewed the product also questions the need of a new GUI, as an official one exists. *NMapGUI* is an alternative to the official one, designed to try to solve its deficiencies and increase user productivity in on-demand scan scenarios. When creating *NMapGUI*, we primarily target occasional users, students or users with limited `nmap` experience. We think that this GUI facilitates learning how to use `nmap` and provides adequate and more efficient ways to explore the different possibilities of the tool. *Rawsec's Cybersecurity Inventory* has added *NMapGUI* as one of their recommended tools, so it can be easily found by more security professionals [31].

Finally, we also used the issue tracker of the *NMapGUI GitHub* repository to collect feedback and feature requests from users. For example, there is a feature request to load a scan configuration from a text file that will be implemented once *NMapDSL* and *NMapGUI* are fully integrated, loading DSL programs into the GUI. A couple of minor visual enhancements have also been requested and are being implemented, like animation and menu improvements, scrollable titles, internationalization, and further information in the tooltips. Another two very interesting requested features are command cancellation and option incompatibility detection. The latter will be available when both products are finally integrated, as this kind of early validation is implemented in *NMapDSL* (see Section 3.2).

4.1 Systematic Validation

We are planning to use these tools in higher education security-related courses, along with other complementary tools [32, 33]. First users will be students from different courses of the *School of Computer Engineering* of the *University of Oviedo* [34]. The *Computer Security* (*BsC of Software Engineering*), the *Web Security Systems*, and *Web Server Administration* [35, 36] (*MsC and Doctorate in Web Engineering*) courses are the first candidates. Next, courses of the infrastructure module of the future *Master on Cybersecurity in Software Engineering* (also from the same school) will also use them to perform infrastructure auditing tasks.

The purpose of applying our tools to education is twofold: facilitate the usage of `nmap` as part of their contents and perform an initial *systematic validation* of our research. We have generally positive evaluation of experts in the field and incorporated our experience on increasing the robustness of dynamically typed languages to improve error detection on audit task creation. However, we did not perform a systematic validation of the tools yet. For this reason, we will describe the plan we established to evaluate our research in the future through the mentioned courses, so we can study its application in one of its intended usage contexts.

As the first author of this paper is the head teacher of several security-related courses, using their students will facilitate the implementation of the systematic validation we are going to describe. This way, we can easily create groups of students, control the interaction they have with different tools, and facilitate data collection and analysis using the online platform that our university provides. Doing this with security experts is not currently possible in our scenario. However, as described in Section 4, feedback from these users was used in a previous stage to improve the tool once released.

As our goal is to improve the productivity of `nmap` security audits, we will systematically validate our research with *use case studies* [37]. They were chosen because the audit processes could only be assessed at a high level of abstraction, as they can be very complex and variable depending on its goal. To do a use-case driven systematic evaluation, we will ask several student groups to perform certain audit tasks of the same difficulty level both with `nmap` and with our tools. Users with no previous experience with `nmap` will be chosen among BsC students (most of them should be), while students with average audit experience will be selected upon the MsC courses, as they received a previous introductory security course. In the first case, an introductory lesson about `nmap` goals, basic operation, and examples will be

given, to provide the students enough materials to a successful start. Once all is set up, the proposed systematic validation procedure will follow these steps:

- *Design*: the objective is to check if the mentioned user groups can successfully finish several audit tasks faster by using our tools than when done via plain `nmap`/`Zenmap` usage. Use cases will be divided in two main groups: creating audit tasks with realistic purposes from scratch and solving problems on supplied audit tasks with known errors. For example, we can ask students to guess the web server versions installed into machines present in a custom local network (this can be easily simulated with *Docker* using little resources).

  In this example, as the users have no clue about how many or where the machines are, they must first locate active endpoints in the network. Then, filter those that indeed are web servers and, finally, correctly guess the web server version. Version guessing could also be made more difficult by hiding from `nmap` default guessing techniques. This way, the proper NSE scripts have to be used to obtain more information from targets, and therefore a correct answer. This audit task example is realistic and require the execution of multiple steps. Therefore, students must check different sets of `nmap` parameters and scan modes, apply them, analyze its output and use it as the input of the next step. This requires practicing or acquiring more knowledge of the tool while they work towards the result, and some amount of experimentation using on-demand scans, so it fits with *NMapGUI* design approach.
- *Data collection*: BsC and MsC students will be separated in two groups. We expect an average of 50 BsC students and 20 MsC. Half of the users in each group will be instructed to do the proposed audit tasks with `nmap`, and the other half will do the same with our tools. This is to prevent that the acquired experience with a tool influences the results of doing the same things with the other: users in a certain group will always work with the same tools to finish an audit task set.

  We will measure the time spent doing the corresponding work for every user by using a questionnaire in the online resource website of the course. Tasks will be proposed as questions, and time will be measured until the user is able to provide an answer to each one. Once one is answered, the next question will be presented until all are. At the end of the questionnaire, a short-answer quiz will be presented to gather feedback about tool usage and features. For example, users may be asked

if the tool provided enough information to complete the tasks, if they had to use external information sources, or if error reporting was considered adequate, all focusing on validating our initial assumptions.

All data collection will be performed by *second degree techniques* to favor automation. Time taken to perform activities can be compiled offline once a certain number of questionnaires have been done, as it will be measured automatically by the course online platform. We can then analyze them at a later stage. Additionally, extra information about needed GUI adjustments could be obtained by observing random users' behavior while working with the GUI while doing the audit tasks.

- *Analysis of collected data*: we are mainly interested in the amount of time spent by both user groups to perform the same tasks. Ideally, our tools will be able to reduce the time taken to perform a task for users with the same knowledge level, while also reducing the amount of failures due to unsolved errors. Note that precise error reporting should also reduce time to perform each activity, as it should have a positive impact on that matter. Geometrical means of task completion times will be used to calculate the average completion times of the tasks. Additionally, some of the proposed audit tasks will include performing scheduled tasks or checking if the output presents certain words. We expect a substantial time gain here, as students using `nmap` should create its own scripts for it, requiring additional knowledge about operating systems or programming languages.
- *Validity*: the described procedure has been designed to reduce threats to validity, although some remain and must be considered. The *construct validity* tries to measure the increase of productivity when designing on-demand audit tasks or when solving predetermined errors on existing ones, which is directly related with our research questions. Introduced errors may also reflect the most common ones according to the experience of the lecturers, so it also helps knowledge transfer. We have also separated students by its starting knowledge level and isolate those that work with different tools, to try to minimize casual relations and thus trying to improve *internal validity*. However, a casual relation problem may appear if the students do not correctly estimate their initial knowledge of `nmap`. Extra care should be taken to measure it, so we can correctly analyze the performance of the tools.

We also think that our findings with real student groups can be directly applicable to other security-related courses in similar

environments, as the outcome of the results will be largely dependent on the amount and precision of the information the tools provide. This increases the *external validity* of the procedure, although care should be taken to use environments with the same goals (improve on-demand audit scans). Finally, the *reliability* of the collected time data could be negatively affected if the users have sources of distraction while doing the intended work: as part of the collected data are time spent to perform activities, the evaluation environment should favor focusing on the ongoing activities, avoiding teamwork, unnecessary communication between students, or limiting access to sources of information to try to measure only time working with tools.

- *Reporting and scheduling*: The described tasks will be incorporated as part of the evaluation activities of the mentioned courses once we reach the *remote system auditing* topics. Reports and analysis about student actions will be compiled and compared to see if there are improvements, if these are different depending on the previous user experience and incorporate adjustments to the tools by analyzing the feedback.

5 Conclusions

The work described in this paper enables users with different experience levels to make better use of the `nmap` security auditing capabilities with less effort. *NMapGUI* is a web application that improves the official `nmap` GUI solving the shortcomings we identified. It gives users structured and immediate access to all `nmap` features, facilitating performing on-demand scans without requiring creating a custom scan profile. Its design approach encourages users to create scan profiles once their options have been tested, so it is easier to share or reuse tried and tested configurations. Additionally, it uses these profiles as a base to allow scan scheduling, output analysis, and usage of advanced capabilities without requiring a deep study of `nmap` features.

NMapDSL is a domain specific language created to develop robust, advanced and portable scanning tasks easier, following the approach implemented by DSLs of popular infrastructure-as-code products. The language supports common useful `nmap` features via keywords, facilitating its usage even without a GUI. This language also incorporates early error validation of scripts, covering critical areas of the scanning process. This way, the probability of generating scan tasks that fail at runtime decreases. Error location is reported precisely, and detailed descriptions of errors facilitates

task debugging. The detailed information displayed by translated scripts about its purpose is also a valuable learning, debugging and logging tool, as the user knows exactly what it is running. Also, modeling robust scan tasks in a simple DSL eases sharing them between systems or users with different skill levels, which also is a valuable tool to use in security courses. We believe that our work may increase the productivity of `nmap` users, ease the design of auditing tasks, the analysis of their results, and enable some user types, like students, to better understand and effectively use the wide set of capabilities of this highly popular security tool.

In a future iteration of *NMapGUI* we are considering adding more options to the interface, such as enhanced reporting options, one-click scan to intermediate nodes in the interactive node graph with the same scan profile as the target one, and ways to compare scan results performed to the same machine. Additionally, as we said in Section 2.3, more work using UI patterns to improve the usability of our GUI can be done [16]. For example, the scheduling part (see Figure 4) can be enhanced using the *Rule Builder* pattern to create search queries based on several conditions, especially when the output analysis feature acquires more functionality in the future. *Shortcut Dropdown* UI pattern can also be applied to dropdowns in the same screen. Navigation through the whole application can also be improved via the *Keyboard Shortcuts*, *Breadcrumbs*, and *Adaptable View* UI patterns, while validation of audit tasks can be easily integrated within the GUI thanks to the *Input Feedback* design pattern. Finally, we also plan *NMapGUI* to achieve the AA rating according the WAI standards [38], including checking background and foreground color contrast.

The next evolution of *NMapDSL* will expand language possibilities by enabling generated scripts to automatically calculate a security rating of each scanned target. This will be done extracting each service type and version information, using the `cve-search` project features [39] to search for corresponding CVEs (*Common Vulnerability Exposures*) containing vulnerabilities of the products found. This way, the users will have immediate feedback about the vulnerability of each scanned machine, also integrating the features of the `CVE-Scan` [40] project. This can also be used to prioritize the machines or infrastructure parts with more critical problems to implement corrective actions. We are also exploring the possibility of applying DSLs with early validation of type errors to lower the complexity or other similar security tools, to incorporate new keywords that further facilitate the usage of advanced `nmap` scan options, and to use *Perl-Compatible Regular Expressions* (PCRE) [41] in the `WITH` block. Additionally, we will implement

the proposed systematic validation procedure described in Section 4.1 in the mentioned security-related courses.

Future work will also aim apply the same approach followed in this research to enable early validation in DSLs of common infrastructure-as-code products. Beginning with *Docker*, we plan to create a special parser of the DSL used in its `Dockerfiles` that provides early validation of their elements. This way, we intend to identify missing *Docker* images, external files, packages to install or, in general, things that may cause conflicts while executing the program or build invalid containers. The aim will be to run the actual `docker` process only over parsed and error-free `Dockerfiles`, attempting to reduce runtime problems. If the file contains errors, a detailed output of the problems found, similar to the output provided by *NMapDSL*, will be provided without deploying any infrastructure.

The current version of *NMapGUI* can be downloaded from https://github.com/danicuestasuarez/NMapGUI. At the date of writing this paper, the project has earned 249 *GitHub* stars, 75 users have cloned it in their own *GitHub* account, and 26 have subscribed to project updates and notifications. An alpha version of the *NTrans* translation tool, able to output *Python* language scripts (versions 2 and 3, also compatible with *PyPy* 2 and 3), can be downloaded from https://www.dropbox.com/s/du0rf0u75k28ny4/ntrans.alpha.v01.zip. This distribution also includes the full source code, tests, ANTLR [24] grammar, documentation, and sample scripts. Once the DSL is finished, both projects will merge into a single product.

6 Related Work

This section describes other products whose aim is also to facilitate working with `nmap`, trying to solve shortcomings like the ones described in Section 1. The focus of these products varies, trying to improve different functionalities or features needed during an audit task. Some aim for simplicity, while others favor on-demand scans the same way *NMapGUI* does. Regarding features, there are alternatives that focus on facilitate reporting (the final goal of every audit activity) or also implement additional features that *NMapGUI* also has, like scan scheduling. There are solutions that even allow management of users and groups, to give different scanning rights to people in a company. The main advantages of *NMapGUI* and *NMapDSL* over the alternative solutions are:

- They cover all `nmap` options and installed NSE scripts, providing substantial usage information.

- Includes scheduling and output analysis features. Not all the alternatives cover scheduling, and output analysis, if available, is typically limited to compare multiple scan outputs.
- Validates scan tasks and precisely report the found errors (see Table 1). This is the most important difference, as our research ensures that the generated scan tasks contain a substantially lower number of potential errors and, if errors are found, precisely locate and describe the problems to facilitate their debugging. None of the following alternatives provide these capabilities.
- Facilitates sharing audit task configurations (DSL source code) and/or multiplatform ready-to-use audit tasks (translated DSL programs). This, combined with the previous feature, ensure that the shared information presents fewer errors, facilitating proper knowledge transfer in production or learning environments.

Regarding the concrete related work of our research, additionally to the `nmap` official GUI (*Zenmap* [12]), which is an evolution of a previous GUI (*NMapFE*), there are several desktop-based projects that created a `nmap` GUI to facilitate its usage to inexperienced users or increase their productivity. *Nmap GUI* [42] simplifies performing simple scanning tasks by just specifying a target a scan profile, like the *Zenmap* approach. It also mimics the `nmap` output and allows the organization of scans using timestamps.

NMapWin [43] is another GUI with similar capabilities, although aimed only to `Win32` platforms, with an outdated interface and no longer maintained by their authors. A more advanced GUI for `nmap` is *NmapSi4* [44], that targets the *Qt* library. This GUI also favors on-demand scans by offering a series of options to use `nmap` in an organized way, grouping the different capabilities of the tool in a set of sections, like the approach of *NMapGUI*. Another similar GUI is *NMapW* [45], a *Windows* tool that organizes the main `nmap` options in a category tree like the one we used in *NMapGUI*, but do not have explicit support for NSE scripts. It also allows users to store different scan configurations.

There are also web-based GUIs for `nmap`. *WMap* [46] allows the user to execute a limited set of common `nmap` commands easily, listing them along with information about its expected output. Additionally, *Nmap-webgui* [47], currently under development, is a project that aims to give users advanced capabilities to handle `nmap` scans, like scan scheduling, comparison of results using `diff`, and review of scan results.

The *nmap-cgi* [48] web GUI allows the usage of typical `nmap` options (scan IPs, subnets, OS detection...) through a very clear and simple interface. It incorporates user and group management, so it can assign different rights over individual `nmap` options to them. It also incorporates three types of scans (single, scheduled and periodic) and exportable output format in XML. Another example is *WebMap* [49], a *Django* `nmap` web interface currently under development with several features like *NMapGUI*, such as the `traceroute` graph. It imports and parses `nmap` XML generated files instead of working with its raw output, and allows to run and schedule auditing tasks from its dashboard, with a focus on reporting features and interacting with external applications. It does not provide the same degree of information than *NMapGUI* about `nmap` options and scripts to maximize its usage potential in on-demand scans, and requires *Docker* installed to run.

We can also find GUIs for `nmap` targeting *Android* devices. Examples are *nmap-gui* [50], *Cydia Tweak Nmap GUI* [51] and the lightweight *pentesting* tool *Lightpen* [33] that, although it does not allow to directly control `nmap` options, it could execute some of its most common scan types as part of its future plugins.

Finally, the `nmap` tool itself is also an integral part of more powerful and popular *pentesting* tools such as *OpenVAS* [52], *Metasploit* [53] or the *IVRE Network Scanning and Analysis* tool [54]. However, the complexity of these tools abstracts the usage of `nmap` within the *pentesting* processes they perform, so users have no direct control over every specific `nmap` option that it is used.

Acknowledgements

This work has been funded by the Spanish Department of Science, Innovation and Universities: project RTI2018-099235-B-I00.

References

[1] D. Harley, L. Myers, S. Cobb, and C. Gutierrez. Cybersecurity trends 2019: Privacy and intrusion in the global village. Technical report, ESET, 2018. (Dec 10, 2018).

[2] A. Bendovschi. Cyber-attacks trends, patterns and security countermeasures. *Procedia Economics and Finance*, 28:24–31, 2015. 7th

INTERNATIONAL CONFERENCE ON FINANCIAL CRIMINOLOGY 2015, 7th ICFC 2015, 13–14 April 2015, Wadham College, Oxford University, United Kingdom.

[3] Y. Gilad and A. Herzberg. Off-path tcp injection attacks. *ACM Trans. Inf. Syst. Secur.*, 16(4):13:1–13:32, April 2014.

[4] P. M. Vidhya. Cyber security: Threats and challenges. *Int.l J. of Computer Science and Mobile Computing*, 3:586–590, 02 2014.

[5] R. Shay, S. Komanduri, A. L. Durity, P. Huh, M. L. Mazurek, Sean M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor. Designing password policies for strength and usability. *ACM Trans. Inf. Syst. Secur.*, 18(4):13:1–13:34, May 2016.

[6] T. Matthews. What DDoS attacks really cost businesses. Technical report, Imperva Incapsula, 2016. (Dec 10, 2018).

[7] N. A. S. Lima and M. P. Fernandez. Towards an efficient DDoS detection scheme for software-defined networks. *IEEE Latin America Transactions*, 16(8):2296–2301, Aug 2018.

[8] J. Cheng, J. Zhou, Q. Liu, X. Tang, and Y. Guo. A ddos detection method for socially aware networking based on forecasting fusion feature sequence. *The Computer Journal*, 61(7):959–970, 2018.

[9] S. Hsiao and D. Kao. The static analysis of WannaCry ransomware. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 1–1, Feb 2018.

[10] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark. A first look at browser-based cryptojacking. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 58–66, April 2018.

[11] D. Kaur and P. Kaur. Empirical analysis of web attacks. *Procedia Computer Science*, 78:298 – 306, 2016. 1st International Conference on Information Security & Privacy 2015.

[12] G. F. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Nmap Project, 2009.

[13] Linux Journal. Editors' choice awards. https://www.linuxjournal.com/article/5525, 2001. (Jul 29, 2019).

[14] OWASP. *OWASP Pentesting Guide v4*. Open Web Application Security Project, 2014.

[15] AlienVault. Alienvalut: Threat detection, incident response product. https://www.alienvault.com/, 2018. (Jan 30, 2019).

[16] A. Toxboe. *User Interface Design Patterns Card Deck: UI Patterns*. UI Patterns Education. Anders Toxboe, 2016.

[17] J. M. Redondo and F. Ortin. A comprehensive evaluation of common python implementations. *IEEE Software*, 32(4):76–84, July 2015.

[18] Google. Google safe browsing. https://safebrowsing.google.com/, 2018. (Jan 30, 2019).

[19] J. M. Redondo and F. Ortin. Efficient support of dynamic inheritance for class- and prototype-based languages. *Journal of Systems and Software*, 86(2):278 – 301, 2013.

[20] F. Ortin, B. G. Perez-Schofield, and J. M. Redondo. Towards a static type checker for python. In *European Conference on Object-Oriented Programming (ECOOP), Scripts to Programs Workshop, STOP*, volume 15, pages 1–2, Prague, Czech Republic, July 2015. ECOOP.

[21] I. Lagartos, J. M. Redondo, and F. Ortin. Towards a java library to support runtime metaprogramming. In Ernesto Damiani, George Spanoudakis, and Leszek Maciaszek, editors, *Evaluation of Novel Approaches to Software Engineering*, pages 224–242, Cham, July 2018. Springer International Publishing.

[22] I. Lagartos, J. M. Redondo, and F. Ortin. Efficient runtime metaprogramming services for java. *Journal of Systems and Software*, 2019.

[23] IETF7. Internet message format. https://tools.ietf.org/html/rfc2822, 2001. (Apr, 2001).

[24] T. Parr. Antlr (another tool for language recognition). http://www.antlr.org/, 2018. (Jan 30, 2019).

[25] Penetration Testing: Security Training Share. NMapGUI: Advanced Graphical User Interface for Nmap. https://securityonline.info/nmapgui-advanced-graphical-user-interface-nmap/, 2017. (Jan 30, 2019).

[26] Div Security. NMapGUI: Interfaz gráfica de usuario para Nmap. http://security.divdesign.mx/nmapgui-interfaz-grafica-de-usuario-para-nmap/, 2017. (Jan 30, 2019).

[27] Homputer Security. Découvrez NMapGUI la version graphique de Nmap. http://homputersecurity.com/2017/10/26/decouvrez-nmapgui-la-version-graphique-de-nmap/, 2017. (Jan 30, 2019).

[28] 1024Megas. NMapGUI - Graphical User Interface. http://www.1024megas.com/2017/09/nmapgui.html, 2017. (Jan 30, 2019).

[29] StackTrender. Nmap GUI Java/Web Front End for Nmap – YouTube. https://stacktrender.com/post/st/nmap-gui-java-web-front-end-for-nmap-youtube, 2017. (Jan 30, 2019).

[30] S. De Luz. NMapGUI: Conoce esta interfaz grafica de Nmap basada en Java. https://www.redeszone.net/2017/09/03/nmapgui-conoce-esta-interfaz-grafica-de-Nmap-basada-en-java/, 2017. (Jan 30, 2019).

[31] A. Zanni. Rawsec's cybersecurity inventory: An inventory of tools and resources about cybersecurity. http://inventory.rawsec.ml/tools.html, 2018. (Jan 30, 2019).

[32] J. M. Redondo and L. del Valle. Filesync and era literaria: Realistic open sourcewebs to develop web security skills. *Journal of Web Engineering*, 17(5):1–22, 2018.

[33] I. Llaneza, J. M. Redondo, and L. Vinuesa. Towards lightweight mobile pentesting tools to quickly assess machine security levels. *IEEE Latin America Transactions*, pp, 2019.

[34] U. de Oviedo. Escuela de ingeniería informática. https://ingenieriainformatica.uniovi.es/infoacademica/grado/, 2018. (Jan 30, 2019).

[35] J. M. Redondo. Improving student assessment of a server administration course promoting flexibility and competitiveness. *IEEE Trans. on Ed.*, 62(1):19–26, 2018.

[36] J. M. Redondo. *Introducción Práctica a la Administración Segura de Servidores Apache Bajo Linux*. Servicio de Publicaciones, Universidad de Oviedo, 2019.

[37] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[38] Web Accessibility Initiative. Wai: Strategies, standards, resources to make the web accessible to people with disabilities. https://www.w3.org/WAI/, 2019. (Apr 30, 2019).

[39] W. Remes, A. Dulaunoy, and P. Moreels. A tool to perform local searches for known vulnerabilities. https://github.com/cve-search/cve-search/, 2018. (Jan 30, 2019).

[40] P. Moreels. Cve scan. https://github.com/NorthernSec/CVE-Scan/, 2018. (Jan 30, 2019).

[41] T. Stubblebine. *Regular Expression Pocket Reference, 2nd Edition*. O'Reilly Media, Inc., 2007.

[42] O. Morten. Nmap gui. https://sourceforge.net/projects/nmapgui/, 2016. (Jan 30, 2019).

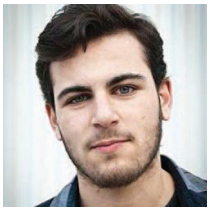
[43] G. F. Lyon and J. Vogt. Nmapwin. https://sourceforge.net/p/nmapwin/wiki/Home/, 2002. (Jan 30, 2019).

[44] F. Cecconi. Nmapsi4. https://github.com/nmapsi4/nmapsi4, 2015. (Jan 30, 2019).
[45] Syhunt. Nmapw: Free port scanner for analyzing network security or internet exploration. http://nmapw.software.informer.com/, 2018. (Jan 30, 2019).
[46] E. Suarez. Wmap. https://github.com/ericsuarez/wmap, 2017. (Jan 30, 2019).
[47] R. Savon. Nmap-webgui. https://github.com/savon-noir/nmap-webgui, 2013. (Jan 30, 2019).
[48] J. Delange. nmap-cgi project. http://nmap-cgi.tuxfamily.org/, 2006. (Jan 30, 2019).
[49] Rev3rse Security. Webmap: Nmap dashboard and reporting. https://github.com/Rev3rseSecurity/WebMap, 2019. (Jan 30, 2019).
[50] F. Dominguez. Nmap-gui. https://github.com/FernandoDoming/nmap-gui, 2017. (Jan 30, 2019).
[51] IDroid.us. Cydia tweak nmap gui. https://web.archive.org/web/20121030090623/https://idroid.us/cydia-tweak-nmap-gui-0-93.html, 2012. (Jan 30, 2019).
[52] OpenVAS. Openvas open source vulnerability scanner and manager. http://www.openvas.org/, 2018. (Jan 30, 2019).
[53] Rapid7. Metasploit: The world's most used penetration testing framework. https://www.metasploit.com/, 2018. (Jan 30, 2019).
[54] P. Lalet. Ivre official web page. https://ivre.rocks/, 2018. (Jan 30, 2019).

Biographies

Jose Manuel Redondo is an Assistant Professor in the University of Oviedo, Spain since November 2003. Received his B.Sc., M.Sc., and Ph.D. degrees in computer engineering from the same university in 2000, 2002, and 2007,

respectively. He participated in various research projects funded by Microsoft Research and the Spanish Department of Science and Innovation. He has authored three books and over 20 articles. His research interests include dynamic languages, computational reflection, and computer security.



Daniel Cuesta is a Computer Network Attack (CNA) consultant in S2Grupo (Valencia, Spain). He has worked as a security consultant in CapGemini Spain and is also a SecurityArtWork Collaborator. We will receive his B.Sc. in computer engineering from the University of Oviedo (Spain) in 2020. His main research interests focus in vulnerability discovery and assessment, along with other projects related with computer security.