Unifying Web Security Assessments with MALA: A Modular Approach

Zeus Sze Hao Chan, Hon Ngee Teo, Jing Hao Lim, Leslie Zhen Yong Tan, Huaqun Guo
Infocomm Technology Cluster
Singapore Institute of Technology
Singapore
2003268@sit.singaporetech.edu.sg; 2003280@sit.singaporetech.edu.sg; 2003264@sit.singaporetech.edu.sg;
2003258@sit.singaporetech.edu.sg; huaqun.guo@singaporetech.edu.sg

2023 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI) | 979-8-3503-9600-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/SOLI60636.2023.10425601

***Abstract*—In the realm of cybersecurity, web applications are increasingly vulnerable to cyber-attacks, which necessitate effective web security tools to identify and mitigate vulnerabilities and threats. Open-source web security tools contributed by the community play a crucial role in detecting such vulnerabilities and threats. However, the multitude of open-source web security tools available often present a challenge for cyber security professionals in terms of syntax variations and inconsistencies across different tools, requiring them to familiarise themselves with multiple tool-specific syntaxes. The team has prototyped a novel new web security tool called Modularised Attack Landscape Analyser (MALA) to combat this. MALA aims to solve the problems faced by cyber security professionals by providing an interface by which commonly used open-source web security tools can be run using a standardised syntax, thereby improving overall productivity of cyber security professionals and easing their burdens as they carry out their work.**

Keywords—Cyber, Open-Source Tools, Web Security, MALA, Modularised Attack Landscape Analyser

I. Introduction

The continuous growth of web applications has significantly increased their attractiveness as a potential attack surface. Consequently, web security has become a critical concern for organisations seeking to protect their assets and user data. The rapid evolution of web security threats is evident from the changing landscape of the Open Web Application Security Project's (OWASP) top 10 list which have added three new categories to the original ten. This shows the quick evolution of web security risks, emphasizing the pressing need for efficient security measures. The complexity and functionality of web programs continue to advance, and so do the possible risks. Fig. 1 shows a frequent changing of the top 10 list from 2013, 2017 to 2021 [1, 2], which reflects a solid proof of this dynamic.

OWASP Top 10 -2013	→	OWASP Top 10 -2017	→	OWASP Top 10 -2023
A1 - Injection	→	A1 - Injection		A1 - Broken Access Control
A2 - Broken Authentication and Session Management	→	A2 - Broken Authentication		A2 - Cryptographic Failures
A3 - Cross-Site Scripting (XSS)	↘	A3 - Sensitive Data Exposure		A3 - Injection
A4 - Insecure Direct Object References [Merged+A7]	U	A4 - XML External Entities (XXE) [new]		A4 - Insecure Design [New]
A5 - Security Misconfiguration	↘	A5 - Broken Access Control [Merged]		A5 - Security Misconfiguration
A6 - Sensitive Data Exposure	↗	A6 - Security Misconfiguration		A6 - Vulnerable and Outdated Components
A7 - Missing Function Level Access Contr [Merged+A4]	U	A7 - Cross-Site Scripting (XSS)		A7 - Identification and Authentication Failures
A8 - Control-Site Request Forgery (CSRF)	⊠	A8 - Insecure Deserialization [New]		A8 - Software and Data Integrity Failures [New]
A9 - Using Components with Known Vulnerabilities	→	A9 - Using Components with Known Vulnerabilities		A9 - Securing Logging and Monitoring Failures
A10 - Unvalided Redirects and Forwards	⊠	A10 - Insufficient Logging and Monitoring [New]		A10 - Server-Side Request Forgery (SSRF) [New]

Fig. 1. Top 10 mapping from 2013, 2017 to 2021

To tackle the latest attack vectors, the development and deployment of new web security tools is essential. While the arising of new web security tools are proactive measures to respond to evolving threats, it has oversaturated the market resulting in redundant functionalities which professionals now find very hard to choose an ideal one for their specific need. Furthermore, it can be troublesome and time-consuming to familiarise with multiple tools and it would also divert attention from the main tasks.

Modularised Attack Landscape Analyser (MALA) addresses the issue of oversaturation while providing a streamlined and modular approach for users. The emphasis on modularity strives to enable users to curate a set of security modules which align with their needs. This reduces redundancy and optimizes the web vulnerability assessment process as it allows users to focus on security strategies rather than learning various tools.

MALA presents a promising solution by offering a modular approach that enables professionals to focus on their core responsibilities and effectively defend against web application vulnerabilities. Embracing modularity will empower security practitioners to safeguard their systems proactively, ensuring a robust defence against ever-evolving cyber threats.

In this paper, MALA addresses these threats by emphasising on automation, efficiency, and modularity. By integrating a range of web security tools, MALA aims to provide a unified and streamlined environment for performing web security assessments and penetration testing, aiding penetration testers, security researchers, and ethical hackers in their efforts to identify and address vulnerabilities in target systems.

The paper is structured as follows – an introduction to the web application security and the reason a tool like MALA was made, followed by a description of MALA's toolset which explains how MALA works and solves the current problems faced by cyber security professionals working within the realm of web security. The section after that goes into greater technical detail of MALA's inner workings, its code and how it runs. After which, comparisons will be made between MALA and tools that currently exist to highlight MALA's strengths and weaknesses. Following that section is one that discusses the future development of MALA as a tool and as a framework. This section will also highlight suggestions and reveal the general direction the team intends to go in as of writing this paper. Finally, a conclusion with findings by the team and acknowledgments will be at the end of the paper.

II. MALA's Unique Features

In this section, we delve into the core of MALA, which revolves around three main components: the Query Builder,

the Cognitive Input Retention, and the Integration of Modules for the seamless integration of modules into the tool.

A. Query Builder

The Query Builder, the cornerstone of MALA, serves as the central interface for constructing and executing commands across its modules. It offers a standardized command syntax that simplifies query crafting and aids module integration. This consistency ensures user adaptability to future updates. The Query Builder streamlines complex command creation through an intuitive and user-friendly framework. Its modular design allows seamless transitions between modules, enhancing MALA's overall usability.

B. Cognitive Input Retention

The Cognitive Input Retention feature in the MALA Toolset is a unique addition that addresses a common problem in web security tools—repeatedly entering arguments during penetration testing. The feature remembers previously entered commands and automatically relates them into subsequent commands, which is especially useful for modules like Gobuster where they often require similar information. It also recognises and saves common arguments across modules such as IP addresses, ports, URL strings, and payloads. With this, users no longer need to re-key frequently used information, thus boosting productivity and reducing risks of manual input errors.

C. Integration of Modules

In anticipation of the extensive range of modules that MALA may integrate with in the future, loading all modules at compile time poses a considerable time burden. To address this challenge, dynamic importing is employed, allowing the loading of essential resources on demand. For instance, when a user initiates the program and selects a specific module, only common variables shared among modules are loaded during compile time. Subsequent loading of a module triggers the inclusion of module-specific variables into memory. While the potency of a tool may be undeniable, its efficacy is hindered without a resource-conscious design. This strategic solution guarantees that MALA maintains optimal resource efficiency, ensuring accessibility across diverse computing environments.

D. Streamlined Initialization & Compilation

MALA boasts a distinguished feature in its modular integration framework, seamlessly accommodating a variety of modules for potential contributors. Each module resides within a dedicated Python class responsible for crafting commands tailored to its respective binary, adhering to predefined command syntax. The range of modules spans essential functionalities like reconnaissance, vulnerability scanning, and fingerprinting. Developed in Python, this framework elevates MALA's adaptability, furnishing security professionals with a robust suite for pinpointing and addressing vulnerabilities in web applications. Contributors can effortlessly expand MALA's functionalities by inheriting from a parent class and specifying module-specific variables within a designated dictionary. The inclusion of 'required' variables ensures the obligatory configuration of critical parameters before module execution. These essential variables undergo a meticulous validation process, guaranteeing their validity. In case of an error, users receive a warning, preventing further execution. This thoughtful design enhances the extensibility and reliability of MALA's modular architecture.

E. Efficient Concurrent Process Management

MALA exhibits a sophisticated attribute proficient in managing concurrent processes, thereby endowing users with a broad spectrum of capabilities. This feature facilitates seamless execution, termination, and control of numerous processes, delivering users paralleled flexibility. MALA's concurrent process management serves as a pivotal aspect of its usability, enabling real-time monitoring, task termination, and comprehensive insights into executed processes. This distinctive functionality markedly augments the tool's adaptability to varied contexts in web security assessments and penetration testing, establishing it as an essential tool for security professionals.

In the domain of web penetration testing tools, the adept management of concurrent processes is pivotal, conferring users the capability to concurrently execute myriad tasks. This attribute serves as a catalyst for heightened efficiency, resulting in a pronounced reduction in the overall duration of the testing phase.

III. MALA Technical Details

The development of the tool, MALA, was preceded by comprehensive research into the prevailing challenges encountered by cyber security professionals. In this section, we present a detailed technical exposition of MALA's functionalities, delving into its capabilities and explaining how they synergistically enhance the overall toolset's performance. Through this section, the team intends to demonstrate the extensive scope and proficiency of MALA, highlighting the significant value it imparts to web security assessments and penetration testing endeavors.

A. Mala.py

Mala.py file serves as a central configuration hub, managing static settings for imported modules and performing administrative tasks, such as storing paths, default variables, and recognized modules. It is responsible for initializing controller.py, which is the core component described in the next sub-section.

B. Controller.py

The controller.py file acts as the tool's brain, handling historical command tracking, identifying loaded modules, and orchestrating query string construction before passing it to executer.py for execution. All modules integrated into MALA are discovered using Python's os.walk method within controller.py, with the requirement that integrated modules must be Python files (.py). Modules, along with their paths and names, are loaded on demand to enhance load times.

1) Initialization: MALA's startup loads vital variables and universal settings, optimizing runtime efficiency. Many products have several features that are not used by most consumers. By strategically choosing which functions to load upfront and deferring others, MALA tackles the challenge of potential feature creep [4]. As the tool integrates more modules, it prevents longer initialization times and high memory usage, aligning with our focus on enhancing the efficiency of cybersecurity professionals.

2) Variable Management: MALA efficiently manages variable changes, prevents invalid selections, and ensures proper values. Variables are organized in a dictionary structure, including a nested mapping for common and

module-specific variables. When modules are used, their specific variables are loaded, granting precise control over variable management. Fig. 2 illustrates the module variables loaded when the http_bruteforce module has been selected.

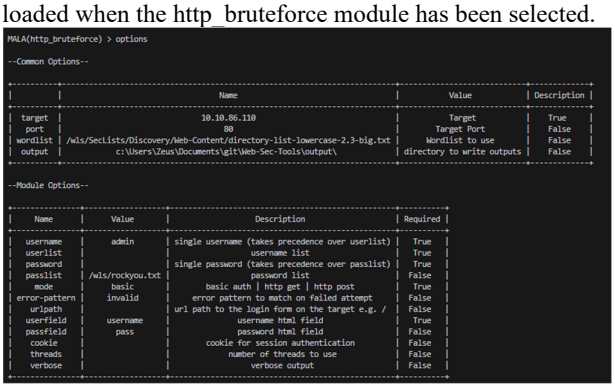

Fig. 2. Http_bruteforce module options displayed

a) Changing a value in the dictionary: If the user wishes to change a value in the dicionary, MALA will search the variable dictionary for the supplied variable and change the value according to the supplied value or clear the value.

Fig. 3 shows an example of changing the target common variable string to “target.com”. If the variable or the value supplied is not valid, the action is not executed, and an error message will be shown to inform the user. Fig. 4 shows what error is thrown and displayed to the user upon detection of an invalid change being made.

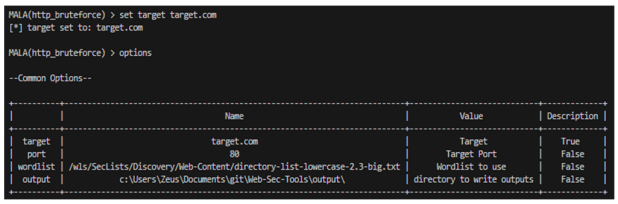

Fig. 3. Changing of the target common variable to “target.com”

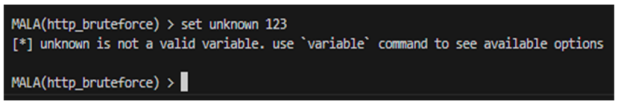

Fig. 4. Invalid variable attempted to be set by user

b) Displaying modules: Users may select a module from the list of available modules. This table is printed based on the modules dictionary maintained by MALA. Fig. 5 illustrates the modules being displayed to the user upon calling the mods command.

Fig. 5. Table of modules being displayed to the user upon calling the mods command.

c) Adding new modules: New modules may also be added. MALA will look through the modules folder for all `py` files and add any new entries into the modules dictionary. Fig. 6 shows how any new modules can be added into the dictionary and subsequently the table to be displayed.

Fig. 6. Modules add command

d) Selecting a module: When the user selects a module to use, MALA performs a simple search through a dictionary mapping to retrieve the path of the module. It is then imported using python’s importlib library. A new module object will be instantiated, replacing any previous instance of any other module selected previously. Fig. 7 shows an example of how to select the url_enum module for usage.

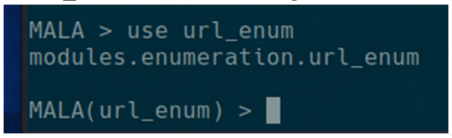

Fig. 7. Use <module> command

e) The module class: In MALA, modules are classes with unique attributes defining their variables. For instance, the "http_brute" module may require a "username list", while "url_enum" may not. This ensures users see only relevant options. Variables are added as earlier described. Some are always necessary, like the "range fuzzing attack" module requiring a range value. A "required" boolean in the module variables dictionary handles this. When conditions are met, it guides users on essential configurations before running the module. MALA keeps separate dictionaries for module attributes, including relevant and required variables, and changing conditions. Fig. 8 shows a print of common variables and module variables

f) Base module: All modules inherit from a parent class “Base Module”, which acts as a foundation for all MALA modules. It handles the initialisation of attributes for child classes and includes abstract methods that child modules can use during their setup. By using the "Base Module", the process of creating new modules is streamlined and follows a standardised approach, making it easier to add modules to MALA efficiently.

3) Execution: During execution, various components work together simultaneously to check for errors, get necessary options and prepare the command for running.

a) Module: All modules in MALA start by invoking a common variable initialization abstract method. This abstraction ensures the modules can access shared variables that are consistently initialized. After selecting a module, its attributes are initialized, allowing it to start constructing the command using user-set options. These options are added to

a list based on the user's specified values, and the order of these options is determined by the user. Some modules, like url_enum, have different modes (e.g., wfuzz for range fuzzing, gobuster for subdomain fuzzing), leading to distinct command structures handled within the module class. The module class returns a list with the command's prefix, arguments, and values for execution by executer.py. If required options are not set, a tailored error message is displayed to the user. Fig. 9 illustrates this scenario.

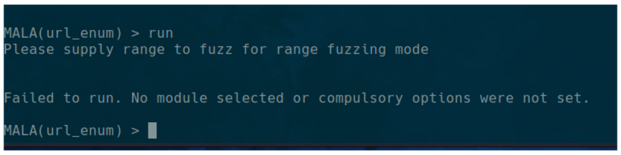

Fig. 9. Failure to run due to no range variable set

Similarly, if the value set is not valid, the module class will not run and will show the user an error message. These validity checks are done through regular expression. Fig. 10 shows an example from the `nikto` module where its tuning option has user validity checks.

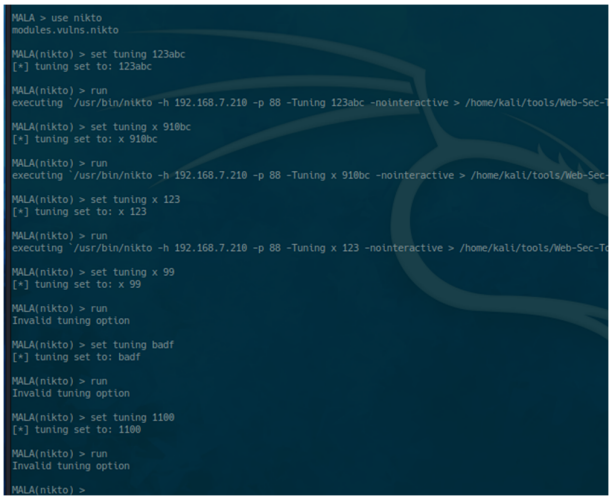

Fig. 10. Nikto tuning value validity check

b) Controller: After obtaining the command list from the module instance, the controller hands it over to executer.py for execution. The process's PID (Process ID) is acquired, and the controller adds a new entry in the executed_processes dictionary to monitor these processes. This dictionary records details like the command, status, execution time, and output file location.

4) Process Status Checking: Controller will loop through the "executed_processes" dictionary, which contains the processes' information to create a nicely formatted table to display to the user all processes that are running and have been run. Fig. 11 shows the process table.

Fig. 11. Process table displaying statuses of processes

5) Output Checking: Users provide an index from the process table. MALA uses this index to locate the associated process and its output file location. If a valid process index is provided, the output file path is forwarded to executer.py. Any output successfully obtained by MALA is then displayed to the user. Fig. 12 represents an example of displaying the command output.

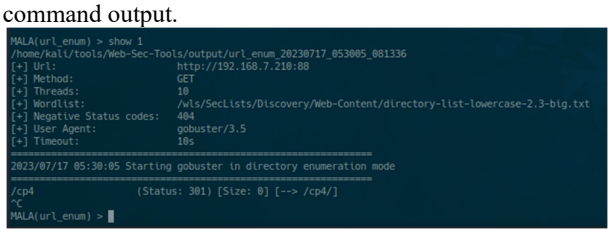

Fig. 12. Output of running the Show <command>

6) Process manipulation: Users can terminate ongoing processes by providing either an individual process index or purging all processes in the "executed_processes" dictionary. Confirmation is sought from the user before proceeding. The request is then sent to executer.py for termination, updating the process status in the table. Fig. 13 and Fig. 14 show the termination of 1 and all processes respectively.

Fig. 13. Kill <index> command

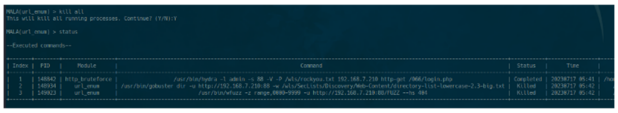

Fig. 14. Kill all command

7) Clearing the process table: Users can maintain the process table by clearing unwanted processes. For a specific index, if the process is running, the user can choose to terminate it; otherwise, the process is removed. Clearing the entire table follows a similar approach for each process. Importantly, clearing the table does not delete output files, enabling users to access the file system and read them. Fig. 15 shows the removal of a single process, and all processes from the process list.

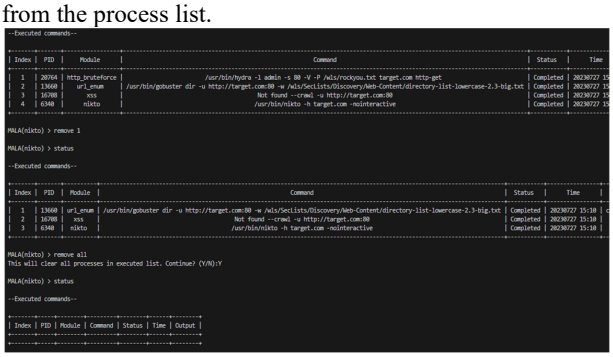

Fig. 15. remove <index> and remove all commands

C. Executer.py

The executer.py file is the engine and driving force of MALA. It contains functions and works in tandem with controller.py to build queries, handle errors, and interface with the user to perform majority of the actions that MALA offers. The functionality of executer.py is as follows:

1) Command execution: A command string to execute along with the output file is supplied to this method. Using subprocess, the command is executed along with shell options to pipe the *stdout* and *stderr* to the supplied output file. If the output file does not exist, the file will be created first. This method returns the PID of the process it created. Fig. 16 shows the running of a module

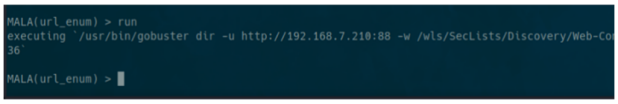

Fig. 16. run command

2) Get process status: Supplied with a PID, executer.py will use *psutil* to check the status of the process. It returns either "Running", "Completed" or the process return code.

3) Read output: Executer.py uses "tail +1f" to dynamically show the output to console. The user will press the "Enter" key to stop the output.

4) Kill process: Given the PID, psutil is used to retrieve the process object. Executer.py then terminates the process and waits for it to fully terminate before continuing with the program. This is to avoid instances where multiple filed `kill` commands could cause the program to bloat the main memory. Executer.py will also handles cases when the supplied PID was not found. In this case it would inform the user.

D. MALA Workflow

The flowchart in Fig. 17 depicts an overview of how all functionalities of MALA work together.

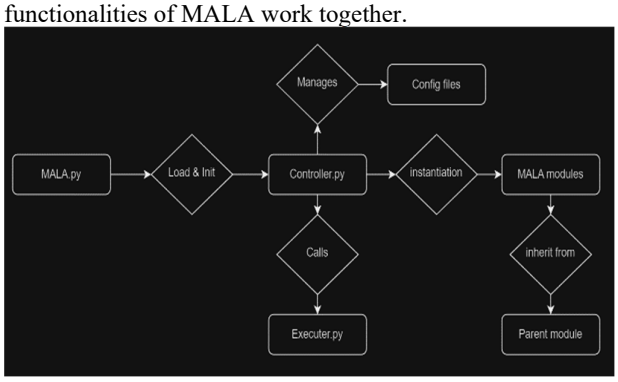

Fig. 17 Workflow of MALA

IV. Comparison To Other Tools

MALA, though in early development, stands out as a unique tool with no direct counterparts. It amalgamates open-source web security tools, streamlining the user experience and eliminating the need for multiple separate tools and varied syntaxes. Unlike other existing tools like Gobuster, Fuff, wFuzz, Metasploit, and Kali, MALA aims to optimize the efficiency of cyber security professionals by providing a novel and standardized approach to web security analysis. [5].

A. Gobuster

Gobuster is an open-source command-line tool used for directory and file brute-forcing during web application penetration testing and reconnaissance [6]. It is designed to discover hidden directories and files within a web server by systematically attempting to access them using a list of potential names or words. The primary purpose of Gobuster is to aid security professionals, ethical hackers, and system administrators in identifying vulnerable or hidden areas of a web application or website. By performing directory and file brute-forcing, Gobuster helps assess the security posture of a web server, uncover potential misconfigurations, and identify weak points that attackers could exploit. Fig. 18 shows an example of the Gobuster command.

Fig. 18: Gobuster url enumeration example syntax [7]

B. Fuff

Fuff, short for "Fuzz Faster U Fool," is a free and open-source web fuzzer developed primarily for penetration testers and security researchers. It was created to automate the process of injecting malformed or unexpected data into web applications and analysing their responses, which helps discover potential security weaknesses [8]. By sending a wide range of input variations, Fuff can effectively expose vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), and others that may not be apparent during regular usage. Fig. 19 shows an example syntax of how to use Fuff to bruteforce directories.

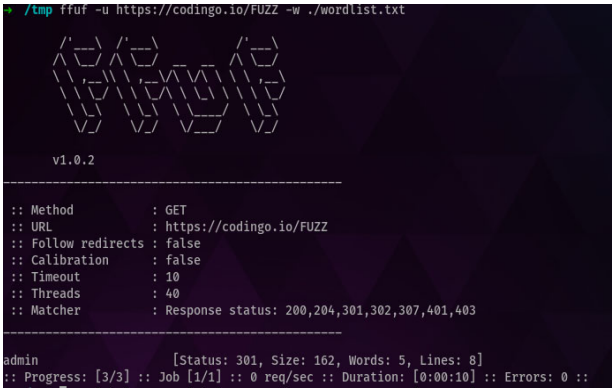

Fig. 19: Fuff url enumeration example syntax [9]

C. WFuzz

WFuzz is a powerful open-source web application security testing tool designed to help penetration testers and security analysts identify vulnerabilities and weaknesses in web applications [10]. The primary method used by WFuzz is "fuzzing," which involves manipulating data input and analysing responses from the web server to identify points of weakness. This tool uses a brute force approach to fuzz web applications, injecting custom data into various parts of the HTTP request to uncover potential vulnerabilities, such as SQL injection, cross-site scripting (XSS), and directory

traversal. Fig. 20 is an example syntax of how to use WFuzz to bruteforce directories.

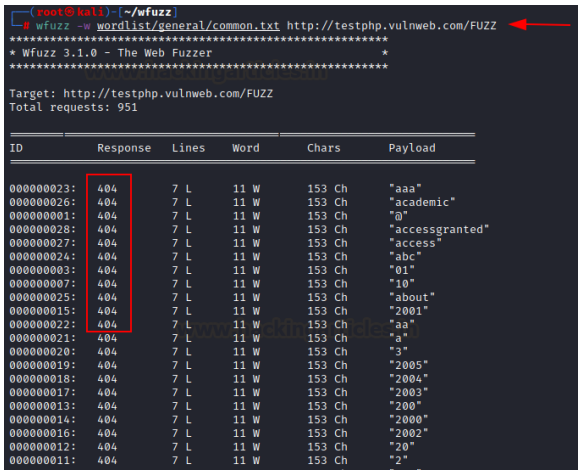

Fig. 20. WFuzz url enumeration example syntax [11]

D. *Metasploit*

Metasploit is a renowned penetration testing framework used to assess and enhance the security of computer systems. Metasploit has evolved as a one stop shop for all the needs of ethical hacking [12]. However, it is more focused on general penetration testing rather than web security and enumeration, which is the domain of MALA.

Our research team found that tools like Gobuster, Fuff, and wFuzz had less user-friendly interfaces for those new to command line tools in web security assessment due to how different all their command syntaxes are. We drew inspiration from Metasploit's standardized command syntax across modules and developed MALA, which modularizes and standardizes commands, allowing users to customize each argument. This approach is illustrated in Fig. 21 and Fig. 22, where MALA transforms and integrates the Gobuster tool's syntax for greater user friendliness.

Fig. 21. Setting argument options on MALA

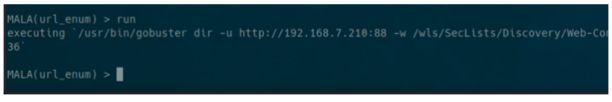

Fig. 22. Executing the command, MALA automatically builds the syntax and runs the command for the user.

E. *Kali Linux*

While having a similar ideology with MALA, Kali Linux has a broader scope, encompassing web security and enumeration. However, it requires substantial manpower for maintenance and updates. The team found Kali Linux inspiring but opted for a more focused approach, leveraging the ideologies behind both Kali Linux and Metasploit. Our goal was to create a tool specifically designed for cybersecurity professionals, providing a convenient framework for web security assessment, while avoiding the resource-intensive demands of a full distribution like Kali Linux.

V. Conclusion

In conclusion, MALA is a user-friendly tool that significantly enhances the workflow of cybersecurity professionals engaged in web security analysis. MALA's modular structure allows for easy integration of current and future modules, streamlining the addition of new features and functionalities. This adaptability ensures that cybersecurity professionals can stay up-to-date with the latest tools and techniques, improving their ability to tackle evolving security challenges effectively. The tool's user-friendly interface empowers users to effortlessly manipulate variables, track processes, and access output information, reducing the learning curve and enabling quick adoption. This simplicity is invaluable, especially in high-pressure situations where rapid analysis and decision-making are crucial. Moreover, MALA's approach of loading modules on demand and prioritising essential functions during initialisation optimises load times and memory usage. This thoughtful design choice aligns with the philosophy of enhancing work efficiency, ensuring that professionals can focus on their tasks without being hindered by long startup times.

Overall, MALA emerges as a valuable and novel asset for cyber security professionals, empowering them to optimise their work and time spent on web security analysis. Its practicality, effectiveness, and user-friendly nature make it a valuable tool for both seasoned experts and newcomers in the field. With MALA, the process of web security analysis becomes more efficient and effective, allowing professionals to stay one step ahead of potential threats and safeguarding digital assets with confidence.

For the future work, our research team has an intention of continuing to maintain the current version of MALA as well as release development of future integrated modules and updates to the community. Our ideas include releasing a version of MALA that can support iOS, improving the Windows version of MALA, and including other open-source tools that the community might be interested in seeing included in the MALA tool. Future implementations may also involve expanding the module library and introducing session control for revisiting prior sessions with retained variables, enhancing MALA's adaptability and usability for evolving security demands.

Acknowledgements

This research is supported by an ignition project with Grant No. R-IE2-A405-0002, Singapore Institute of Technology and Ministry of Education, Singapore.

References

[1] D. Wichers, and J. Williams, “OWASP Top-10 2017”, OWASP Foundation, 2017.

[2] OWASP Foundation, Top 10 Web Application Security Risks, https://owasp.org/www-project-top-ten/, last accessed October 9, 2023.

[3] Bianca, “Exploring the Cybersecurity Trifecta: Windows, macos, and linux in the spotlight,” AITechTrend, May 2023.

[4] S. Jain, "Time inconsistency and product design: A strategic analysis of feature creep," Marketing Science, vol. 38, no. 5, pp. 835-851, 2019.

[5] Y.Nader, “Top 10 open source security testing tools for web applications,” Hackr.io, April 17, 2023.

[6] G. Gandal, “Gobuster - penetration testing tools in Kali Tools,” GeeksforGeeks, 2023.

[7] "Gobuster – an elegant CLI utility for brute forcing URI directories," Latest Hacking News, March 26, 2019.

[8] Intigriti, “Hacker tools: Ffuf (Fuzz Faster U Fool),” Intigriti, May 3, 2021. https://blog.intigriti.com/2021/05/03/hacker-tools-ffuf-fuzz-faster-u-fool-2/, last accessed October 9, 2023.

[9] Codingo, “Everything you need to know about FFUF,” September 17, 2020. https://codingo.io/tools/ffuf/bounty/2020/09/17/everything-you-need-to-know-about-ffuf.html, last accessed October 9, 2023.

[10] X. Mendez, “The web fuzzer,” Wfuzz. https://wfuzz.readthedocs.io/en/latest/, last accessed October 9, 2023.

[11] R. Chandel, “A detailed guide on WFUZZ,” Hacking Articles, March 5, 2022. https://www.hackingarticles.in/a-detailed-guide-on-wfuzz/, last accessed October 9, 2023.

[12] K. Ranganath, "Instant Metasploit Starter," Packt Publishing, 2013.