LWASP

Advanced Users Guide

LWASP is a two part service: The setup utility and the installer. The setup utility is a simple GUI (graphical user interface) with several tabs, one for each category of what can score. All this setup utility does is read what is checked and generate two files: elements.csv and commands.bash.

The elements.csv is a CSV (comma-separated values) file that contains the scoring elements, or what will show up in the scoring report as the student secures the image. The commands.bash is a BASH (bourne-again shell) script that lists the commands that, when run, creates the needed vulnerabilities on the image so that when fixed, they can be scored. BASH commands are just commands that can be run in the terminal app. The setup utility places these two files in the lwasp-install directory after they are generated. The installer downloads various libraries, runs the command.bash script, turns the elements.csv file into an encrypted scoring.json file, and runs various other commands to install the scoring engine.

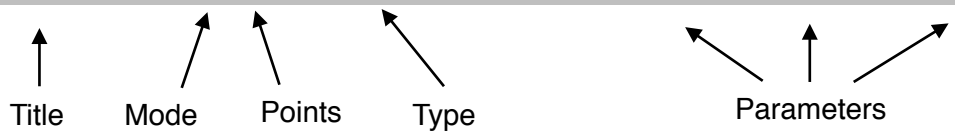
There are two ways that advanced users can leverage these characteristics of LWASP to score more advanced vulnerabilities on a Linux system. First, you can create your own vulnerabilities on the image, such as modifying boot files in the /boot folder to create a low-level backdoor, and then add how to score this in the elements.csv file before you run the installer. Secondly, you can modify the scoring engine code itself (written in python) so that you can score different types of vulnerabilities, such as whether a given file has been modified by a specific daemon. This guide will document both ways. You can also add features to LWASP, but the details of how to do that will not be covered in this guide.

Modifying the elements.csv file

The best way to edit this is in a text editor such as Notepad
on Windows, Textedit on MacOS, or Gedit on Linux

Sample row:

```
Guest account is disabled,V,10,FileContents,/etc/lightdm/lightdm.conf,TRUE,allow-guest=false
```



In plain text, this line says that when the file /etc/lightdm/lightdm.conf contains the words allow-guest=false, 10 points will be awarded to the student and the scoring report will reflect that the “Guest account is disabled”.

Title	What the competitor will see on their scoring report
Mode	Either V for vulnerability or P for penality. The vulnerability points are added to the score, and the total points is the total number of vulnerability points. The penalty points are subtracted from the score, allowing for potentially negative scores.
Points	The points to be added or subtracted from the competitor's score if the item is complete
Type	One of the 8 types explained below
Parameters	The extra information as laid out below

Types

What can be checked	Type Name	Function
Contents of a file	FileContents	Parses through a file, given the file path, to search for any number of distinct strings using python `open().read() command`
Existance of a file or directory	FileExistance	Checks to see if a file or directory exists at a given path using python `os.path.isfile()` & `os.path.isdir`
Service active running status	Service	Checks to see if a service is running using output of bash command `ps -A`
Forensics Question	Forensics	Parses through a CyberPatriot style forensics question looking for any number of distinct answers following the string "ANSWER: "
Version of software/ service	Updates	Checks the current version or a service with bash command `dpkg -l` *`servicename`* `| grep -E "^ii" | tr -s ' ' | cut -d' ' -f3)`
Port currently listening	Port	Checks whether port is listening using bash command `netstat -tunelp`
Permissions on a file	Permissions	Checks permissions on a file using python `oct(os.stat(filepath)[ST_MODE])[-3:]`
Any shell command	Command	Basically a catch-all, checks output of any other bash/shell command using python `subprocess.check_output()`

Parameters

Type Name	Parameter 1	Parameter 2	Parameter 3
FileContents*	Absolute File Path	(T/F) Whether the file should contain the search string(s)	Search String 1
FileExistance	Absolute File Path	(T/F) Whether the file/directory should exist	
Service	Service Name	(T/F) Whether the service should be running	
Forensics*	Absolute File Path	Answer 1	Answer 2 (if needed)
Updates	Service Name	Version Required	
Port	Port number	(T/F) Whether the port should be open	
Permissions	Absolute File Path	Permissions Needed in numerical form (e.g. 745)	
Command	BASH command to run	(T/F) Whether the output should contain the text	Text to look for in output of command

* These types are capable of handling an unlimited number of parameters, so if you want to check if a file contains 5 different items, just continue adding them on to the end of the line, separated by commas

---

Hopefully, using this documentation, you can score your custom vulnerabilities by adding entries to the elements.csv file after you run the setup utility but before you run the install utility. Looking at the other auto-generated entries in the elements.csv file can be a great reference to how it can be used.

Add Scoring Categories

1. Add a function in analyze.py that can take in the extras as parameters and return True or False as to whether that item should be marked complete according to those parameters
2. Near the bottom of the analyze.py file, where the recording file is read, add your category beneath all of the others, with the same syntax, but use function = myNewFunction(extras...)
3. In initialize_gui.py, add your new category to where it checks each type for validity.
4. That's it! Add your category in the elements.csv file with the needed extras, and start using it. Contact me with any questions.

Breakdown of the files

(indicates a file that you should provide)**

	//// These are the python files that do all of the grunt work ////
initialize_gui.py	Sets up the scoring engine to run, and puts everything in motion.
analyze.py	Called every minute from a crontab and every time a file that is scored changes. It parses through what should be scored and determines what score the user of the image has achieved, then writes that back to recording.json.
restart.py	Sets up the file watches and does several other tasks that must be run on boot. It is called from cse.bash
	//// These are the files that are referenced/used by initialize_gui.py ////
cron.bash	Called in initialize.py to setup the cron job that calls analyze.py. It is only ever called once
elements.csv	The generated comma-separated-value file that contains what should be scored and what should be penalized
	//// These elements in the lwasp-report folder make up the UI ////
main.js	The heart of what powers the UI. It is loaded from report.html, which is opened when the user double clicks on it from their desktop
jquery.js	Used in main.js to power the ajax calls that retrieves recording.json and settings.json in javascript to do some calculations on and display to the user
logo.png	The image that is used on the scoring report and on the desktop images
react.js	Powers the react code in main.js
	//// These store data to be used by the main three python scripts above ////
recording	Stores all of the user-imported elements from elements.csv but in a more computer friendly format, also adding a field for whether each item is completed.
settings.json	Contains some user-generated settings that were entered during initialization.

- Feel free to read through all of my code, I have tried to comment it as much as needed. If you know any amount of python it should be fairly easy to understand and modify to fit your needs.
- main.js is written in javascript using a framework called React, which was created by Facebook in 2013. Feel free to research and learn it, but I have to warn you that just messing with it as plain javascript may not work that well.