FLURRY:

A Fast Provenance Framework

Authors: Maya Kapoor, Alex Poloniewicz, Sanya Rattray.

This work is an open source project performed in collaboration with the University of North Carolina at Charlotte and the Cyber Resiliency, Security, and Trust (CReST) Lab. For support, bug reports, or questions, please contact: mkapoor1@uncc.edu.

Table of Contents

What is Data Provenance?

Scientists describe provenance as the lineage of data. Provenance is used to answer questions such as, “where did the data come from?”, and “how was the data produced and modified?”. The W3C Provenance Incubator Group describes data provenance as,

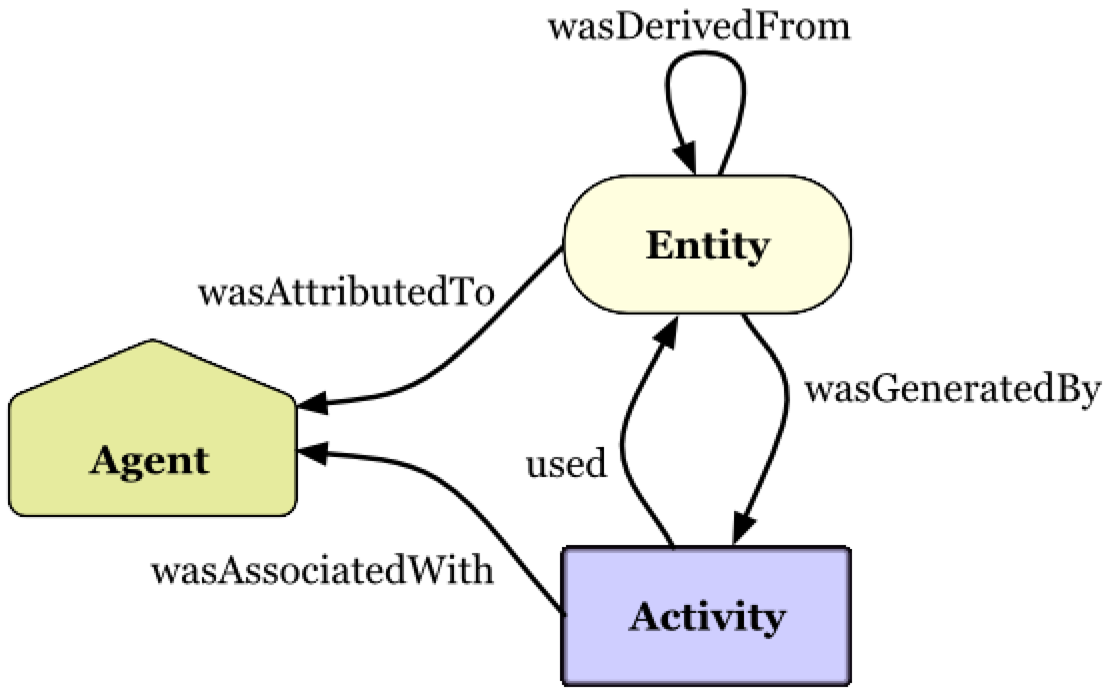
"A record that describes entities and processes involved in producing and delivering or otherwise influencing that resource. Provenance provides a critical foundation for assessing authenticity, enabling trust, and allowing reproducibility. Provenance assertions are a form of contextual metadata and can themselves become important records with their own provenance."[1]

Data provenance records the history of data over time, but also importantly captures relationships among data. These associations can provide credibility to the data and be used to establish trust. Furthermore, this trustworthiness makes data reusable for both scientific research and root cause analysis.

There are many steps to consider in the provenance process, including provenance creation, storage, maintenance, and analysis. In all these stages, a standard model is needed for organization. The W3C PROV Data Model[2] recognizes three data types: Entities, Agents, and Activities, and relations among these which can be studied in depth in their documentation.

PROV Concepts	PROV-DM types or relations	Name	Overview
Entity	PROV-DM Types	Entity	Section 2.1.1
Activity		Activity	Section 2.1.1
Agent		Agent	Section 2.1.3
Generation	PROV-DM Relations	WasGeneratedBy	Section 2.1.1
Usage		Used	Section 2.1.1
Communication		WasInformedBy	Section 2.1.1
Derivation		WasDerivedFrom	Section 2.1.2
Attribution		WasAttributedTo	Section 2.1.3
Association		WasAssociatedWith	Section 2.1.3
Delegation		ActedOnBehalfOf	Section 2.1.3

[1] http://www.w3.org/2005/Incubator/prov/XGR-prov/.
[2] https://www.w3.org/TR/prov-dm/



An example provenance graph, from the W3C PROV Model.

Due to its relational properties, provenance data can be well-represented as a provenance graph. The PROV-DM concepts are modeled as nodes, and relations as the edges. Provenance graphs are *directed*, meaning that an entity for example acts upon an activity in a directional relationship. The graphs are also *acyclic* due to the time series of actions. The graphs which are produced by provenance data are by nature *heterogeneous*, as nodes will be different files, processes, inodes, and more diverse node types which will have their own unique features. Provenance produces *multigraphs* which have multiple relationships between nodes of varying edge types, making the graph problem both rich with information and uniquely challenging for analysis.

What is Graph Representation Learning?

Machine learning methods may be applied to graphs to solve classic network science problems like node classification, link prediction, community detection, and many others. The capabilities of graph machine learning are outside the scope of this work, but for a better understanding we encourage readers to peruse *Graph Representation Learning* by William Hamilton, available freely online: https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf.

For data provenance, graph representation learning may be used to detect anomalies in execution graphs. The assumption which underpins this work is that executions of anomalous behavior (for example, a cross-site scripting attack) will generate provenance subgraphs which differ structurally and in their hidden representations from benign execution graphs.

Graph machine learning can aid cyber security analysts in detecting and adapting to unforeseen attacks in resilient systems. Unsupervised learning may be performed on benign system processes, meaning that we do not need to train on known attacks. Thus, anomaly detection may also work for zero-day attacks. Additionally, recurrent properties of neural networks allow for learned data to be retained over time, making anomaly detection possible over longer periods. This is especially significant for advanced persistent threats that follow a “low and slow” attack pattern. The goal of machine learning in these contexts is to provide alerts for adaptive systems so that they can be resilient even in the worst scenarios against malicious intruders.

Why an Automated Framework?

Graph machine learning and data provenance collection, storage, and summarization have been well-studied in the research community, but their intersection does not have foundational frameworks for reproducibility or abundant public datasets to perform tests. Works like SIGL[3], Unicorn[4], and PROV-GEm[5] have used graph neural networks to successfully detect anomalous execution graphs made from cyber attacks. Some of the work conducted and data sets used could not be shared due to proprietary concern, which inhibits reproducibility for further research and verification. In other works, the datasets generated by provenance tools only provide kernel-level provenance and are captured at the whole-system level without refinement. Furthermore, these datasets are static. They cannot be easily reproduced with certainty that the system specifications and attack scenarios are the same, and users must rely on researcher documentation to be sure of what they are reading in the data.

In order to provide contribution to the research community in provenance graph representation learning, we have developed FLURRY, a fast framework for producing, collecting, storing, and transforming data provenance. Our system allows researchers to recreate several web server-based cyber attacks and capture and store provenance from these scenarios. Additionally, one can perform benign web browsing behavior in order to learn from this data as well. In future versions of FLURRY, we will add other attack scenarios besides web-browser based attacks such as malicious software installation and network-based attacks like denial of service.

[3] https://arxiv.org/abs/2008.11533.
[4] https://mickens.seas.harvard.edu/files/mickens/files/unicorn.pdf.
[5] To be added.

Supported Packages

This section provides both preliminary information and configuration details for the software packages installed on the FLURRY virtual machine. This documentation is provided for transparency to our process as well as reproducibility to build your own version of FLURRY to your liking. The configurations may be changed at researcher discretion, but please be aware that these tools are not foolproof. We provide paths to log files where available for debugging purposes as well as outside documentation from the authors of these packages.

Fedora 33

FLURRY is built on the Fedora 33 operating system. We chose this because it is readily compatible with CamFlow, our chosen tool for kernel-level, whole system provenance capture. You can download your own Fedora 33 VM for free from OSBoxes[6], or just the ISO from the Fedora Project.[7]

Installation:

1. Create a virtual machine on your preferred hypervisor (we used ProxMox).
2. Allow as many processor cores as you deem necessary.
3. Allow at least 16GB of RAM for optimal performance.
4. Start your VM with the Fedora ISO attached.
5. If not pre-built, follow the instructions for installing.
6. If not pre-built, detach the ISO and reboot.

//TODO: add Ubuntu scripts

CamFlow

CamFlow[8] is a Linux security module designed for fine-grained, whole-system provenance capture. The module works by using hooks to monitor security access in the kernel space as well as NetFilter hooks to capture network provenance. These are passed via the CamFlow daemon to user-space, where it may be conveyed to a log file, piped to a storage back-end, or conveyed via message bus to some other interface. There are a number of configuration options both for the application and daemon in order to fine-tune the provenance capture to your

[6] https://www.osboxes.org/fedora-33-virtual-machine-images-available-for-virtualbox-and-vmware/
[7] https://getfedora.org/en/workstation/download/
[8] https://camflow.org/

liking. The system may also be configured for whole-system capture or target capture for specific files and/or processes.

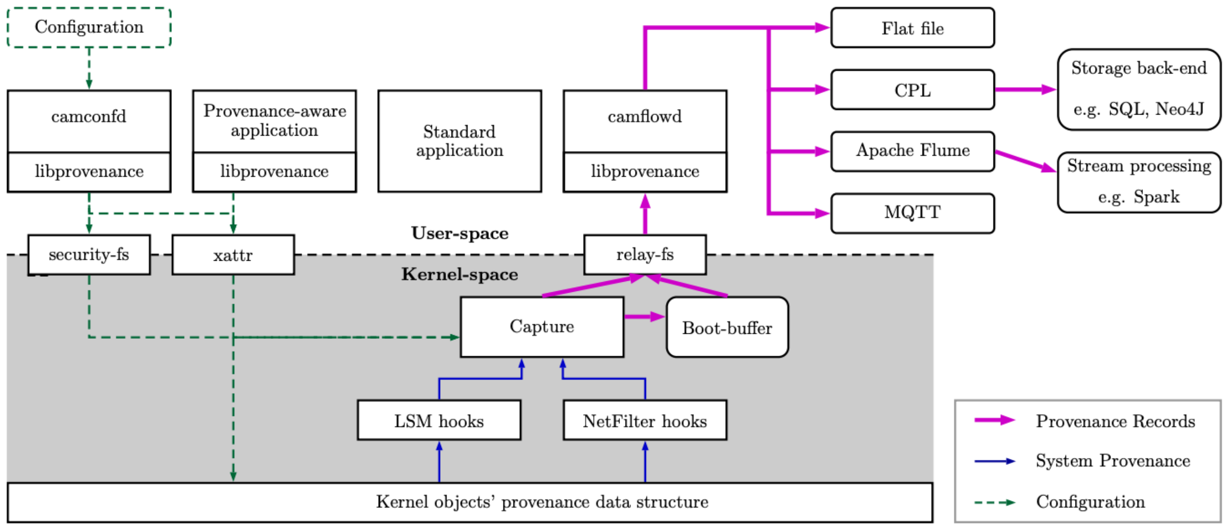


Figure 4: Architecture overview.

System overview of CamFlow.

Please follow the most recent installation instructions provided by CamFlow authors on their website: https://camflow.org/#installation. As of this version of FLURRY, the current recommended installation process is through the dnf package manager.

Installation:

1. Download the setup script from source.

```
curl -1sLf
'https://dl.cloudsmith.io/public/camflow/camflow/cfg/setup/bash.rpm.sh' |
sudo -E bash
```

2. Install via the package manager.

```
sudo dnf -y install camflow
```

3. Activate the modules.

```
sudo systemctl enable camconfd.service
sudo systemctl enable camflowd.service
```

4. Reboot the virtual machine using the CamFlow kernel from the boot menu.

Configuration:

There are two configuration files for users in CamFlow. The file `/etc/camflow.ini` contains details for provenance capture. The `/etc/camflowd.ini` file is used to configure the daemon for conveying information to user space and storage mechanisms. Some of the capture mechanisms may also be dynamically configured.

Run the following command to get more details on dynamic configuration options:

```
sudo camflow -h
```

You can enable/disable whole-system provenance capture using the following commands:

```
sudo camflow -a true
sudo camflow -a false
```

You can track a specific file using this command:

```
sudo camflow --track-file /path/to/file true
```

The static configuration options in the before-mentioned configuration files must be saved and the system rebooted in order to take effect. The following is an example `/etc/camflow.ini` file as we have it set for FLURRY v. 4.0. Please refer to the original CamFlow documentation for the latest updates to their system.

```
[provenance]
;unique identifier for the machine, use hostid if set to 0
machine_id=0
;enable provenance capture
enabled=true
;record provenance of all kernel object
all=false
node_filter=directory
node_filter=inode_unknown
node_filter=char
node_filter=envp
; propagate_node_filter=directory
; relation_filter=sh_read
; relation_filter=sh_write
; propagate_relation_filter=write
```

```
[compression]
; enable node compression
node=true
edge=true
duplicate=false

[file]
;set opaque file
opaque=/usr/bin/bash
;set tracked file
;track=/home/thomas/test.o
;propagate=/home/thomas/test.o

[ipv4-egress]
;propagate=0.0.0.0/0:80
;propagate=0.0.0.0/0:404
;record exchanged with local server
;record=127.0.0.1/32:80

[ipv4-ingress]
;propagate=0.0.0.0/0:80
;propagate=0.0.0.0/0:404
;record exchanged with local server
;record=127.0.0.1/32:80

[user]
;opaque=vagrant
;track=vagrant
;propagate=vagrant

[group]
;opaque=vagrant
;track=vagrant
;propagate=vagrant

[secctx]
;track=system_u:object_r:bin_t:s0
;propagate=system_u:object_r:bin_t:s0
;opaque=system_u:object_r:bin_t:s0
```

```
[general]
output = mqtt

format = w3c
;format = spade_json

[log]
address=m12.cloudqmqtt.com:17065
username=camflow
password=test
qos=0
topic=camflow/provenance/
```

For optimal behavior, in `/etc/camflow.ini` enable data compression as well.

```
[compression]
data = true
```

Audit Log File

For provenance storage, CamFlow can be configured to output to a system log file, messaging service, or socket or pipe. FLURRY is configured to output to the system log audit file as this was the most stable in our build. The downside of the logging file is that it does fill up storage space quickly when whole-system capture is used. Thus, log files must be exported from the VM or discarded so that processes may continue running. In a real system, regular exports to storage, graph summarization, and/or fine-tuned capture should be leveraged to reduce memory overhead.

The audit log file can be found at `/tmp/audit.log`.

Because of the way the CamFlow/SPADE plug-in is written, editing the audit log file causes the reporter to fail until the system is rebooted. For this reason, we provide a reset script which reboots the system after clearing this log file. Otherwise, run the following commands to clear the log and restart.

```
cp /dev/null /tmp/audit.log
sudo reboot now
```

CF2G

CF2G, or "CamFlow to graph," is a tool developed by the authors of PROV-GEm in order to convert the JSON output of CamFlow into graph format which is readable by machine learning libraries. We currently support a method of conversion which takes in either W3C JSON formatted CamFlow data and returns several transformations of the graphs which are compatible with Deep Graph Library[9] constructors and functions. We also export the graphs built into serialized gpickle formats which can be read by NetworkX.[10] Our next steps in CF2G development will be to formalize these transformations into a plug-in compatible with graph storage mechanisms and add to the repertoire of graph output formats.

XAMPP

XAMPP is a free PHP web development environment which can be used to easily create and deploy web applications on multiple platforms. In the underlying framework, XAMPP runs an Apache web server with MySQL or MariaDB as the database backend. More information on XAMPP stack and the link to download for your platform can be found here: https://www.apachefriends.org/index.html

Before working with XAMPP, the following dependencies need to be installed:

```
sudo dnf -y update
sudo dnf -y install libnsl
sudo dnf -y install wget
```

Once the dependencies are in place, download XAMPP for Linux from Apache Friends. Modify the run file to be executable and execute in order to run. Note that the version of the run file must correspond with your system's version of PHP.

```
chmod +x xampp-linux-x64-8.0.6-0-installer.run
./xampp-linux-x64-8.0.6-0-installer.run
```

Follow the installation process as instructed by the pop-up window. Be sure to check the box that installs all core dependencies (MySQL, Apache, etc.) if you do not already have all those components installed.

When installation is complete, the root directory for the stack will be `/opt/lampp`. In order to manage the web services, you will need to re-run the .run file. In order to make this process

[9] https://www.dgl.ai/.
[10] https://networkx.org/.

more straightforward, we provide a desktop application, “XAMPP”, which can be found by searching FLURRY’s applications. XAMPP processes are also started implicitly by starting FLURRY.

Damn Vulnerable Web App

The Damn Vulnerable Web App is a tool for cyber security developers to practice their exploitation skills. Because it is very vulnerable, it should never be installed on a public web server. We have set up the web application on the local web server on the FLURRY virtual machine in order to capture web-based cyber attacks and record their provenance to the graph database. There are many recorded vulnerabilities in the web application along with undisclosed attack vectors which are left to be discovered. You may also configure DVWA to different security levels to provide a bigger challenge. We provide instructions in the “Damn Vulnerable Web App Runner” section of this documentation for six different exploitations.

The DVWA source code can be cloned at https://github.com/digininja/DVWA, or you can download the files from the repository and unzip the archive into the public_HTML folder from your XAMPP installation in the previous step.

```
cd /opt/lampp/htdocs
git clone https://github.com/digininja/DVWA.git
```

In order to set up the database, your config file must be copied from the default. This can be done by copying and just using the defaults:

```
cp /opt/lampp/htdocs/dvwa/config/config/config.inc.php.dist
/opt/lampp/htdocs/dvwa/config/config/config.inc.php
```

You will need to modify the following section in the config as shown below in order for the database to be created in the next steps.

```
$_DVWA[ 'db_user' ] = 'root';
$_DVWA[ 'db_password' ] = '';
$_DVWA[ 'db_database' ] = 'dvwa';
```

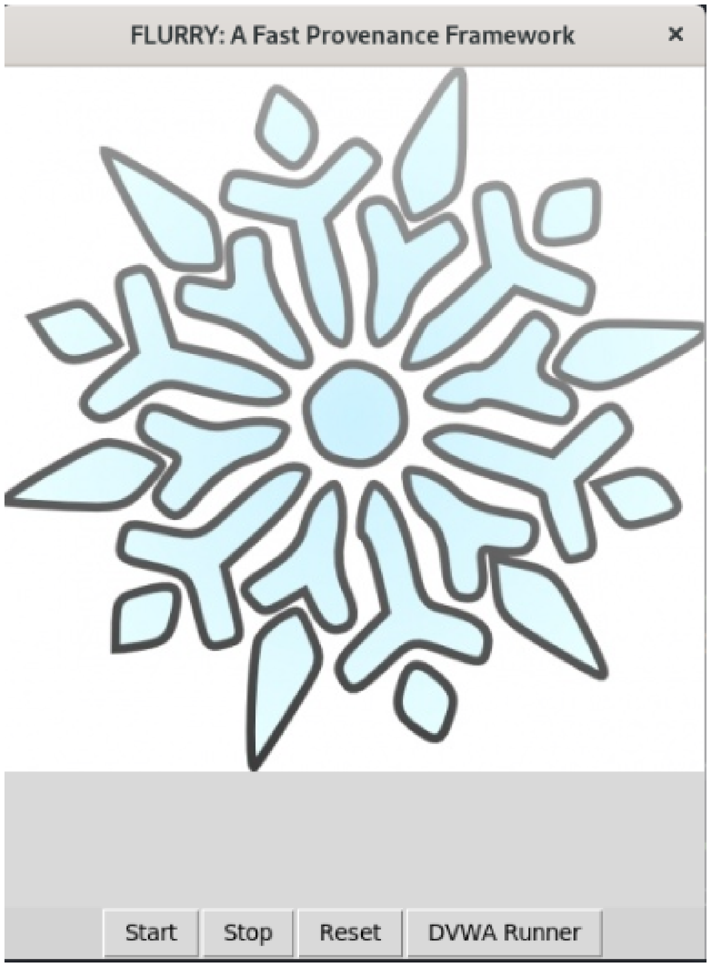
Point your browser to: http://127.0.0.1/dvwa/setup.php. Follow the DVWA instructions for installation. The default username for DVWA is “admin” and the default password is “password”.

Getting Started

The FLURRY virtual machine is set up with a user profile and password. The password can be used for sudo/root permissions.

Username: cyber
Password: ITIS6010

We offer a very simple GUI to start and stop the FLURRY system control services. You can find the FLURRY application by searching for it on the virtual machine. The GUI and a corresponding terminal will be launched. Selecting one of the three control buttons will execute commands on the terminal and may launch the SPADE controller with instructions for the user to complete in the terminal space. If you wish to view the scripts or run them from the command line, the scripts corresponding to the buttons can be found at `/home/cyber`.



Start

Running the start script will first launch the XAMPP server processes. The user will be prompted for the super user password for user “cyber.” The password is “ITIS6010”.

Web Server Configuration

We have developed an automated approach to conducting cyber attacks on the Damn Vulnerable Web Application. The script runs selected attacks dynamically on the virtual machine - the web application is launched and the attack is shown in real time. This allows the user to generate attack provenance in a transparent and recreatable manner. Furthermore, we use a simple level configuration to let the user refine provenance collection to only the browser, web server, database, or the whole system. Because our primary motivation is provenance generation, we set the DVWA security level to “low” and focus instead on complexity on generating recordable activity.

We provide support for six different web-based server attacks, as well as looped execution of benign browsing behavior and malicious behavior. In future versions of FLURRY, we will expand to other attack types. All attacks can be run with the simple choice from terminal, but we provide the steps for each attack and some explanation of what is going on for transparency.

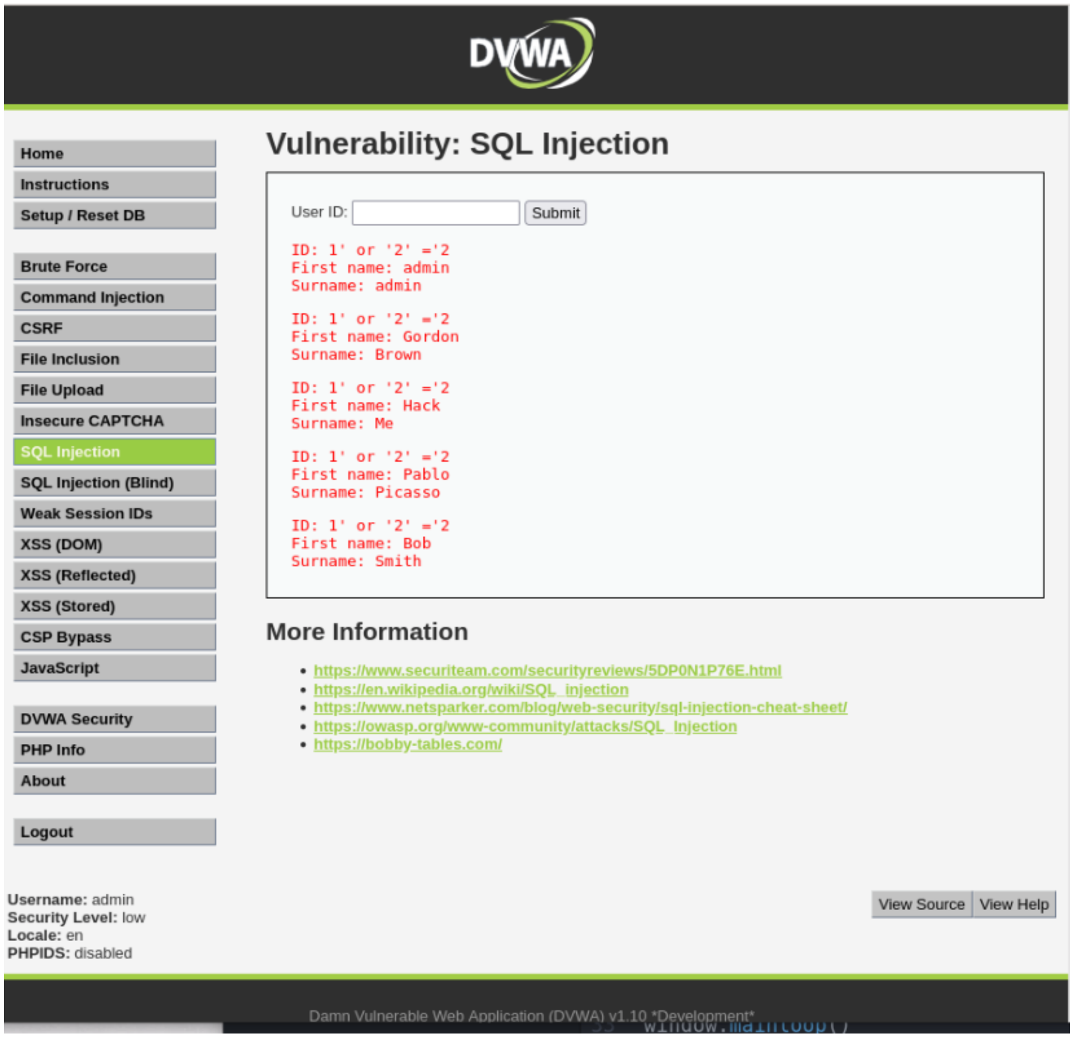
There are a few preliminaries which are required to run the DVWA Runner. First, python and selenium must be installed.

```
sudo dnf install python3.9
sudo pip3 install selenium
```

<u>SQL Injection Attack:</u>

A SQL injection attack involves “injecting” database queries to the server through the client application. These kinds of methods can be used by attackers to impersonate users, exfiltrate data, void or modify transactions, and gain administrative privileges.
In order to conduct the SQL injection attack from the DVWA browser interface, run the following steps:

1) Set the Damn Vulnerable Web App security level to “low.”
2) Set the User ID to 1’ or ‘2’=’2.
    - Since 2=2 will always be true, the First and Surnames will be displayed.
3) Run ‘ORDER BY 1# and continue to increment by 1 to discover the number of columns which will help us know how many fields to attempt to explore.
4) Run ‘UNION SELECT user, password from users# to exfiltrate the list of usernames and hashed passwords within the table.



Vulnerability: SQL Injection

User ID:

```
ID: 1' or '2' ='2
First name: admin
Surname: admin

ID: 1' or '2' ='2
First name: Gordon
Surname: Brown

ID: 1' or '2' ='2
First name: Hack
Surname: Me

ID: 1' or '2' ='2
First name: Pablo
Surname: Picasso

ID: 1' or '2' ='2
First name: Bob
Surname: Smith
```

More Information

- https://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

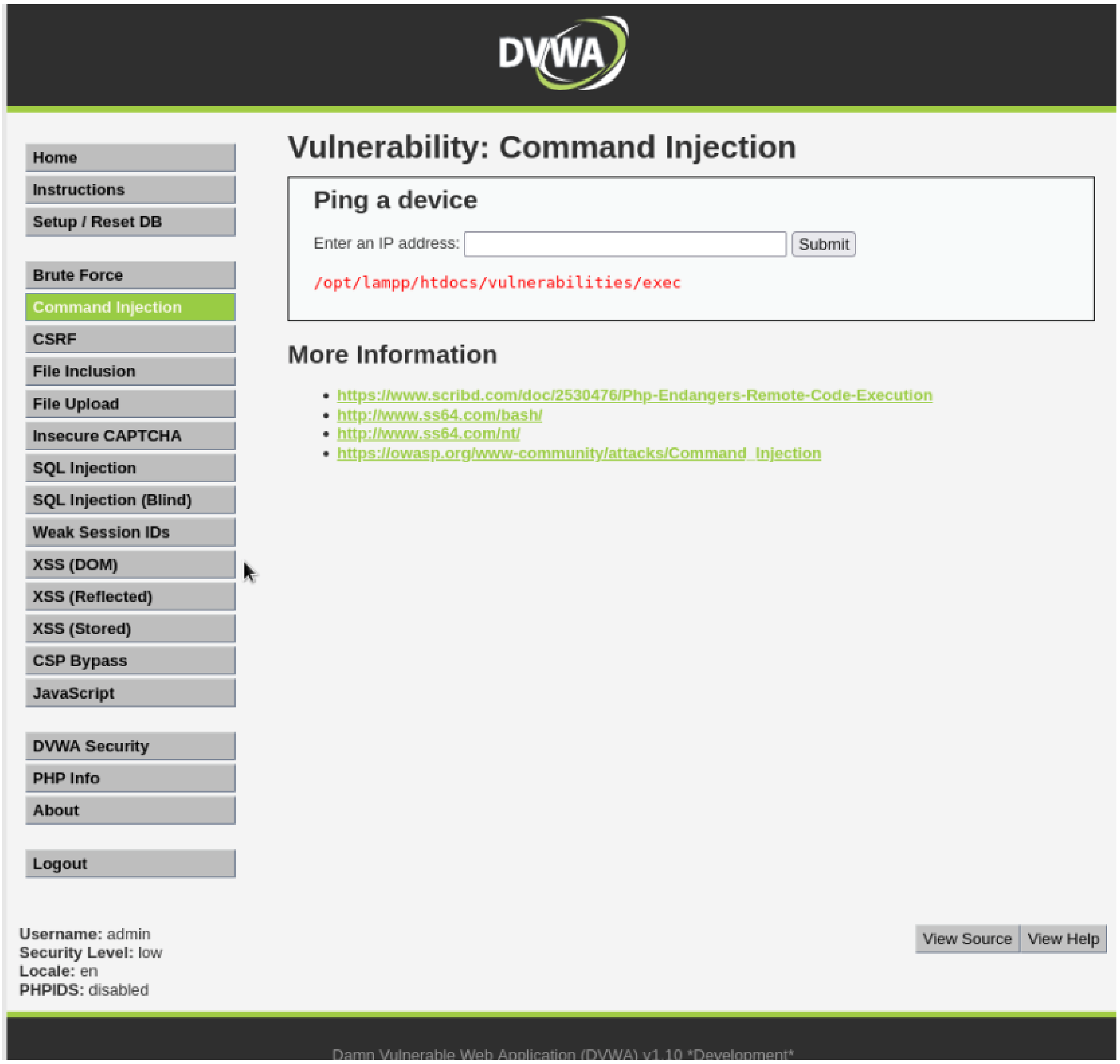
Username: admin
Security Level: low
Locale: en
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Command Injection Attack:

Command injection attacks occur when an application is vulnerable to allowing the execution of arbitrary commands through the application. Much like SQL injection, this is often done through unsanitized user input. In order to conduct a command injection attack, do the following on the DVWA browser:

1) Set the Damn Vulnerable Web App security level to “low.”
2) Enter “ping 127.0.0.1; pwd” into the field and click “Submit.”



Vulnerability: Command Injection

Ping a device

Enter an IP address:

```
/opt/lampp/htdocs/vulnerabilities/exec
```

More Information

- https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://owasp.org/www-community/attacks/Command_Injection

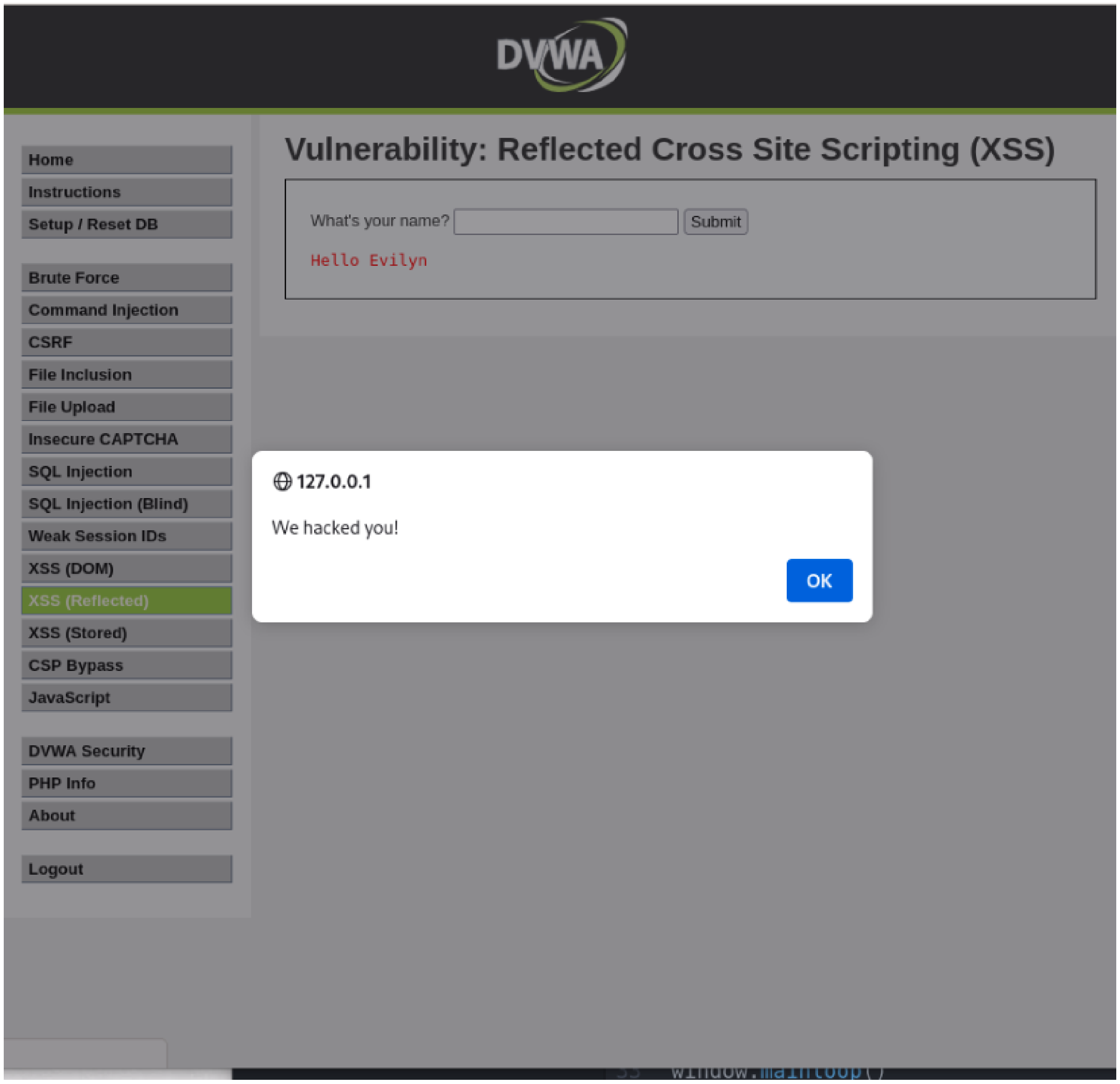
Username: admin
Security Level: low
Locale: en
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Cross Site Scripting (Reflected):

Cross-site scripting allows attackers to inject scripts from the client-side to modify web pages viewed by other users. Reflected cross site scripting occurs when an application receives data in an HTTP request and immediately includes that data in the response unsafely. To execute this kind of attack, perform the following steps:

1) Set the Damn Vulnerable Web App security level to “low.”
2) In the textbox field, enter name followed by a script alert for Cross Site Scripting. For example, *Evilyn <script>alert(‘We hacked you!’)</script>.*



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

```
Hello Evilyn
```

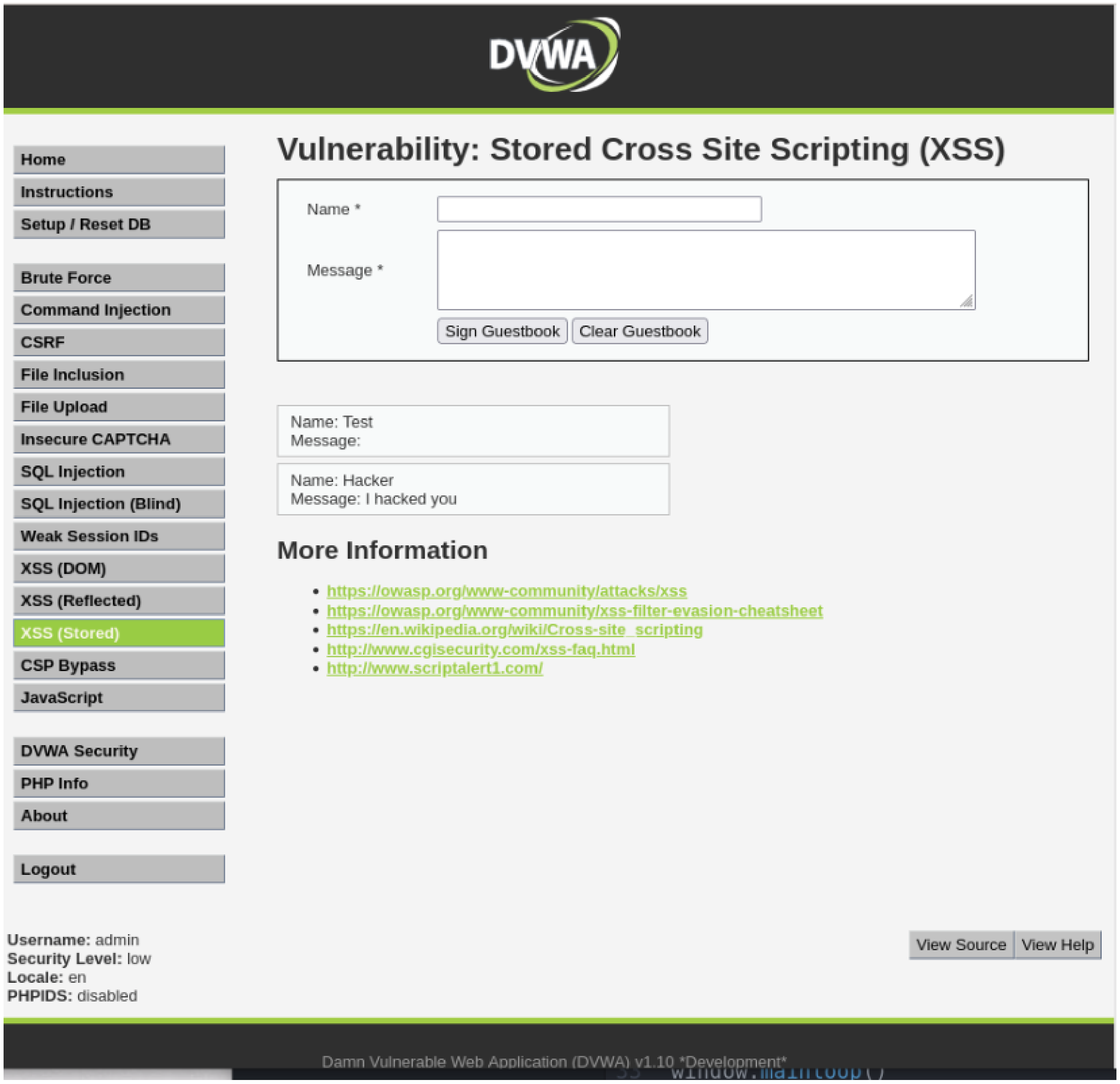
127.0.0.1

We hacked you!

Cross Site Scripting (Stored):

Unlike reflected cross site scripting which infiltrates the victim's browser view, stored cross site scripting is persistent and is injected into the web application. In order to perform this attack, use the following steps in the DVWA browser:

1) Set the Damn Vulnerable Web App security level to "low."
2) In the message portion, write <script>alert("This is stored XSS")</script>.
3) You will now see a popup appear everytime the stored XSS page is visited.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: Test
Message:

Name: Hacker
Message: I hacked you

More Information

- https://owasp.org/www-community/attacks/xss
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

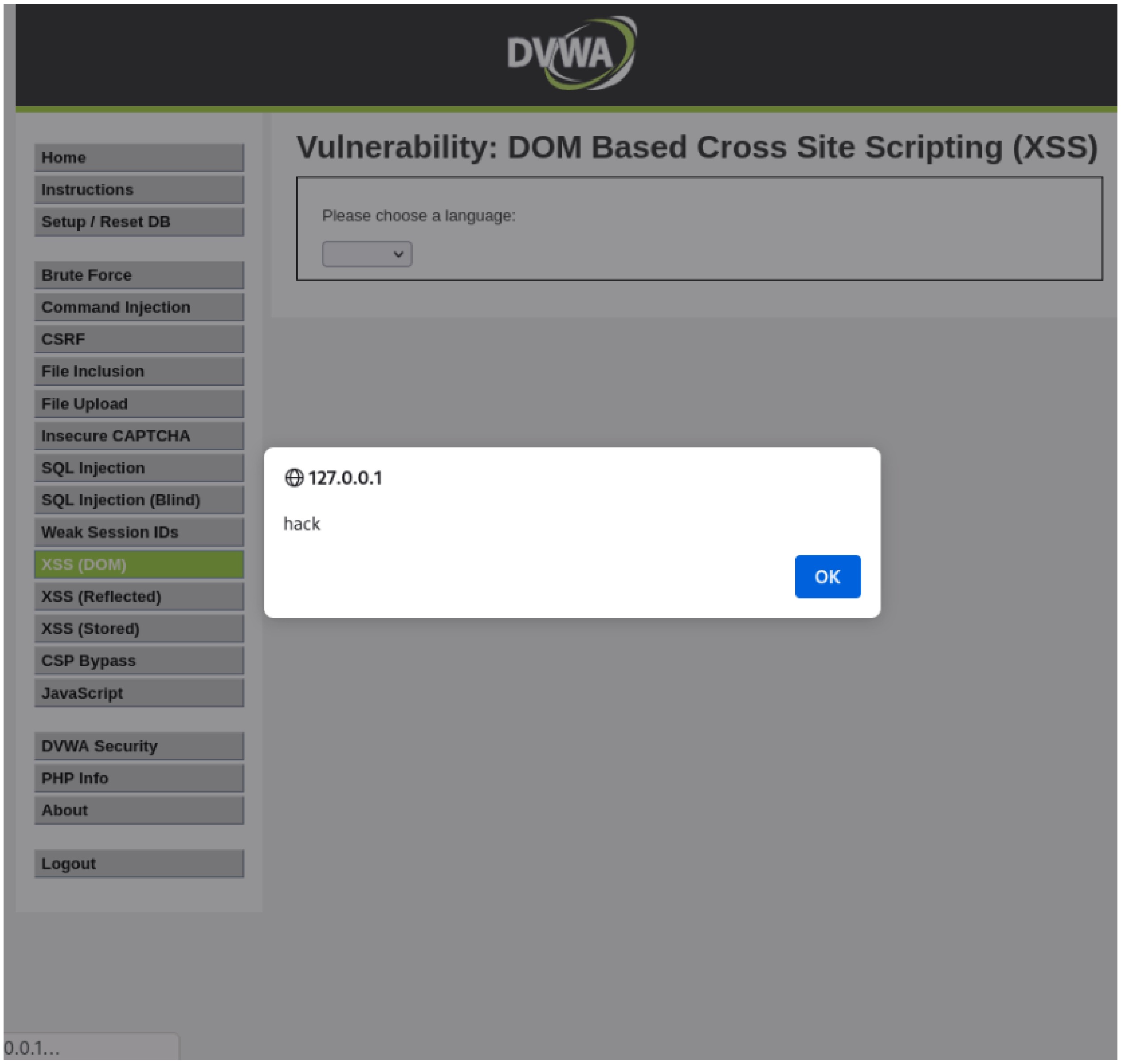
Username: admin
Security Level: low
Locale: en
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Cross Site Scripting (DOM):

Another form of cross site scripting involves modifying the DOM environment in the victim's browser. The vulnerability arises when an attacker can control a source and pass it to a dangerous function (the "sink"). In order to perform this attack, run the following steps:

1) Set the Damn Vulnerable Web App security level to "low."
2) Modify the url to the following:
http://127.0.0.1/vulnerabilities/xss_d/?default=English<script>alert("hack");</script>.



Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

127.0.0.1

hack

Brute Force Password Attack:

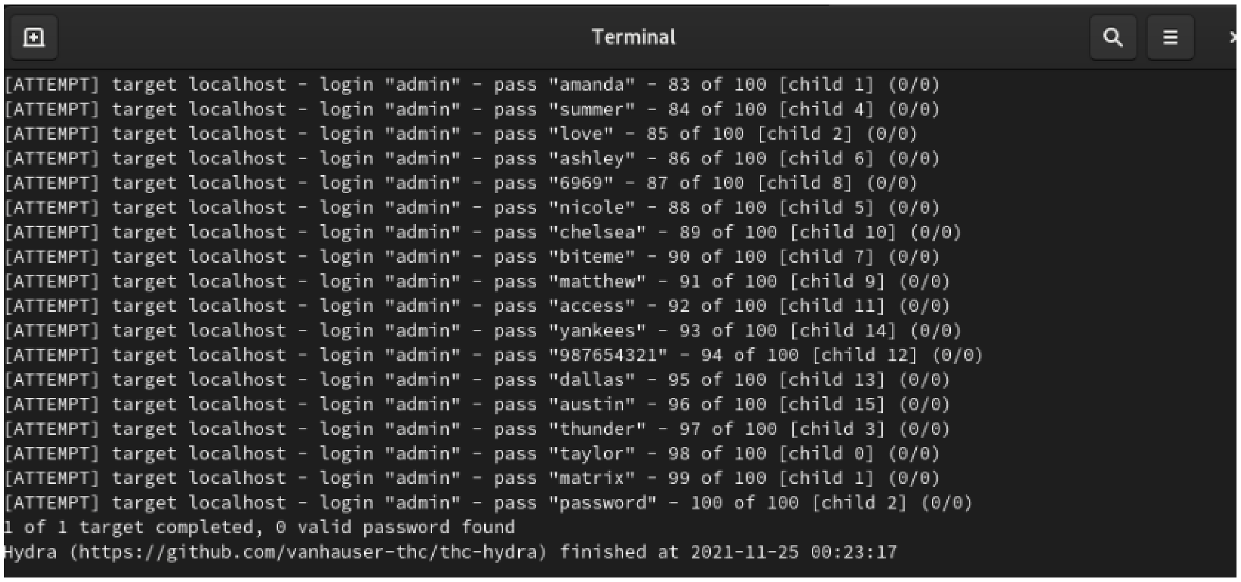
Our last attack method is a simple brute force password attack using a passwords list from the Internet and the Hydra[11] password cracking tool. In order to run the hydra tool, use the following steps:

1) Set the Damn Vulnerable Web App security level to "low."

[11] https://www.kali.org/tools/hydra/.

1. Download a password list (we provide one, 100-password-list.txt, for an example).
2. Log in to DVWA and navigate to the brute force page.
3. Run the following command:

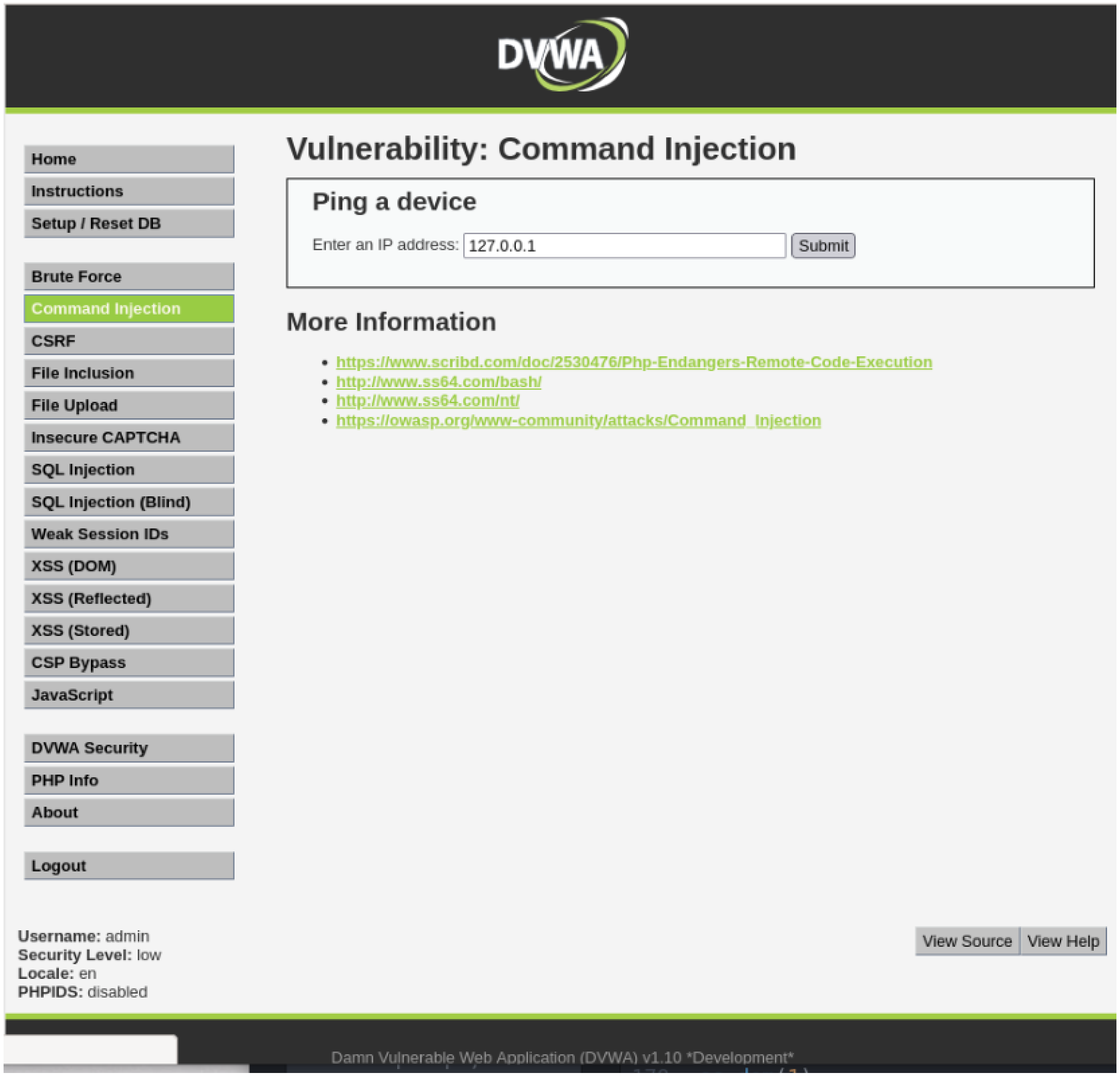
```
COOKIE=$(curl -k -c 'https://127.0.0.1/vulnerabilities/brute/index.php')

Hydra localhost -V -l admin -P 100-password-list.txt http-get-form
"/home/cyber/DWVA/vulnerabilities/brute/index.php:username=^USER^&password=
^PASS^&Login=Login:F=Username and/or password incorrect.:H=Cookie:
security=low; PHPSESSID=${COOKIE: -26}"
```



```
[ATTEMPT] target localhost - login "admin" - pass "amanda" - 83 of 100 [child 1] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "summer" - 84 of 100 [child 4] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "love" - 85 of 100 [child 2] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ashley" - 86 of 100 [child 6] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "6969" - 87 of 100 [child 8] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "nicole" - 88 of 100 [child 5] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "chelsea" - 89 of 100 [child 10] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "biteme" - 90 of 100 [child 7] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "matthew" - 91 of 100 [child 9] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "access" - 92 of 100 [child 11] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "yankees" - 93 of 100 [child 14] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "987654321" - 94 of 100 [child 12] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "dallas" - 95 of 100 [child 13] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "austin" - 96 of 100 [child 15] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "thunder" - 97 of 100 [child 3] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "taylor" - 98 of 100 [child 0] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "matrix" - 99 of 100 [child 1] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "password" - 100 of 100 [child 2] (0/0)
1 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-11-25 00:23:17
```

Benign Execution

We also support benign web browser activity from the DVWA Runner. It would be up to user discretion to make your own benign activity outside of this option. Our script essentially loops through the various tabs of DVWA, inserting into the submission fields various expected values. This makes sense for FLURRY's purposes as it corresponds to the "routine" activity one would expect on the Damn Vulnerable Web App if it were a normal application.



Vulnerability: Command Injection

Ping a device

Enter an IP address: 127.0.0.1

More Information

- https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://owasp.org/www-community/attacks/Command_Injection

Username: admin
Security Level: low
Locale: en
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Provenance Levels

FLURRY supports targeted granularity of provenance capture as well as whole-system provenance. The “levels” of provenance capture which we describe here should be distinguished from provenance “layers.” Levels have to do with a targeted approach to capture that is granular to only specific processes or files. Layers involve gathering provenance at various parts of the system stack, from the kernel/OS layer up to the application and out to the network. Because FLURRY currently only supports provenance capture at the kernel level through CamFlow, we have not yet implemented a multi-layer provenance capture system. In future versions, we hope to add application layer provenance as well.

Web Browser Level:

In order to isolate provenance capture to the web browser level, we track the Firefox process (the default browser for XAMPP stack).

Web Server Level:

For web server level provenance, we track the `httpd` daemon process for the Apache web server which is also part of the XAMPP stack.

Database Level:

For database level provenance, we track the `mysqld` process which is run through the XAMPP stack. Note that if you are configuring this yourself, be sure to track the MySQL installation that is part of your web server environment and not one that might be associated with a SPADE controller or local to your system.

Whole-System Capture Level:

We use CamFlows “-a” flag for capturing *all* kernel-layer provenance.

Network Host

The network host is an implementation of Flurry v4.0’s adaptable attack framework to simulate network-based attacks. There are currently three avenues supported - data exfiltration using ICMP, a TCP-SYN flooding attack, and remote code execution using an hping backdoor. These scenarios are run similarly to the DVWA Runner with corresponding benign behavior.

To-dos:

- **Add Description of Net Bot and network attacks**

Custom Attacks & Custom Benign Behavior

In addition to the provided scripts and scenarios, researchers may use the Flurry interface to run their own scripts and create graphs out of whatever data is run on the virtual machine.