AutoPS: A Suite of Tools for Privilege Separation in Software Systems

Amruth Venkatraman, Jonathan Chien, Thomas Lu, Jeffrey Sun

December 12, 2014

1 Introduction

Writing secure applications is not only extremely difficult but also incredibly time consuming. Our contribution to this problem is a suite of tools bundled under the name AutoPS. It is a collection of tools to help developers write and evaluate privilege separated applications quickly, without stepping into the confusing and messy realm of UNIX permissions and UIDs and GIDs. To our knowledge, an application of this type was not previously available.

2 Motivation

Over the course of 6.858, we have examined many different systems and techniques for compromising or securing various applications. However, it seems that almost all systems have some security vulnerability through which an attacker can compromise one or more processes of the system, regardless of the preventative techniques used by the developer. It may thus be better to assume from the beginning that systems *will* be compromised at some point.

By expecting that the system will be compromised at some point, we are put in the position of trying to mitigate the damage that can be done once this occurs. An effective defense for this is called privilege separation. While working on emulating OKWS [1] in Zoobar Lab 2, it became apparent that the traditional method of privilege separation was insufficient. Manually assigning UIDs and GIDs to processes and files is not only frustrating but also unscalable, as the number of dependencies between processes and files grows superlinearly with respect to the number of each. The developer cannot be expected to trace through all dependencies in terms of users and groups as the size of the application grows. Furthermore, in an attempt to make the application functional, the developer may end up assigning permissions which are more lax than necessary, as finding the combination of UIDs, GIDs, groups, owners, and permissions that gives the minimal required set of permissions is very difficult.

It is clear, then, that a solution to allow privilege separation that abstracts the UNIX specific details of permissions is necessary.

3 Components of AutoPS

AutoPS is comprised of three components: the privilege separator, a file monitoring tool, and a visualization platform to view process-file interactions. Each component is designed to be able to offer some service independent of the others.

3.1 Permissions Helper

The first component of AutoPS is the actual privilege separation mechanism. We make the reasonable assumption that the developer understands the application and system he is making. We do not purport to

```
Chroot: /jail/
Whitelist: AutoPS/, usr/, dev/, etc/, tmp/, lib/

Group
Process: login-server.py
Args: /loginsvc/sock
Start
Reads: /, loginsvc, banksvc, stringsvc, login-server.py, cred.db, rpclib.py, pbkdf2.py, bank_client.py, string_client.py, zoodb.py
Writes: /, loginsvc, banksvc, stringsvc, cred.db
Executes: /, loginsvc, banksvc, stringsvc, login-server.py, cred.db

Group
Process: bank-server.py
Args: /banksvc/sock
Start
Reads: /, banksvc, bank-server.py, zoodb.py, rpclib.py, bank.db
Writes: /, banksvc, bank.db
Executes: /, banksvc, bank-server.py, bank.db

Group
Process: string-server.py
Args: /stringsvc/sock
Start
Reads: /, stringsvc, string-server.py, zoodb.py, rpclib.py, userstring.db
Writes: /, stringsvc, userstring.db
Executes: /, stringsvc, string-server.py, userstring.db

Group
Run as: 70000
Process: repl.py
Reads: /, loginsvc, banksvc, stringsvc, repl.py, login_repl_client.py, bank_repl_client.py, string_repl_client.py, rpclib.py
Writes: /, loginsvc, banksvc, stringsvc
Executes: /, loginsvc, banksvc, stringsvc, repl.py
```

Figure 1: An eample of a `config.txt` file for AutoPS.

offer an intelligent system that figures out how the system operates: rather, we expect the developer to write a configuration file that specifies how the system operates. For an example of how to specify a configuration file, consult Figure 3.1.

Using this configuration file, we can then extract the information and programmatically assign permissions to the files specified for particular processes. This effectively removes the mental overhead of reasoning about systems at scale in terms of users and groups. The configuration file provides a rich interface for specifying details of the application. Process groups can be specified such that every process within will be given the same UID and GID. The developer can specify particular UIDs for components as well. The configuration file also allows for specifying files to whitelist as well as the chroot location.

The actual mechanism by which permissions are assigned is through the use of Access Control Lists (ACLs) rather than through the traditional user/group/other model. Permissions Helper determining how to assign UIDs and GIDs. It begins by invoking `chmod 000 [file]` to deny access to all files by any user, group, or other. It then changes the owner of all files to a reserved user and group. Lastly, it grants permissions by calling `setfacl` on files one by one with the (UID, [r/w/x]) pairs it has determined need access. Finally, it then fires off all the processes marked for starting in the configuration file.

3.2 AutoPSMonitor

The second component of AutoPS is a file monitoring service. The ability to track accesses to files and their type is a crucial aspect of security when it comes to evaluating privilege separation. As such AutoPS tries to offer a solution to the problem. By employing the use of `fanotify` [2], we offer a tool that can listen in the background to all attempts to read, write, execute files specified by the developer. The results of the monitoring service are written to an output text file, which can then be searched and handled as desired by the developer.

While the tool is simple, the results are powerful. Being able to identify points of potential significant failure (points where many processes are interacting with a file) suggests to the developer that a possible redesign is necessary. Being able to see that a process unexpectedly wrote to a file is a sign that some permissions in the system are wrong. Developing a privilege separated application is only half the task,

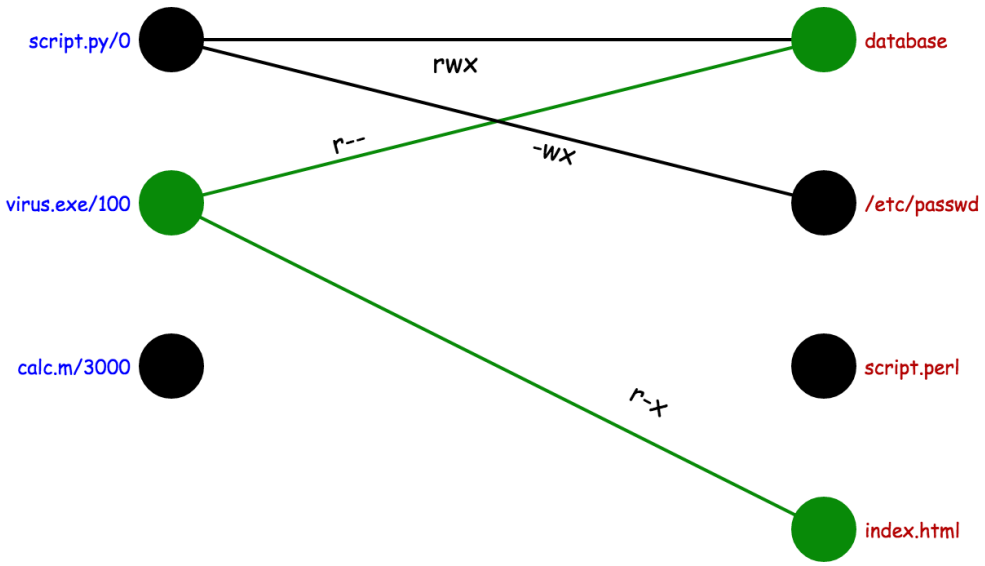

Figure 2: An example of the process-file interaction visualization produced by AutoPS.

while the other arguably more important half is evaluating it.

We envision the usage of this tool as follows: the user runs our file monitoring service and sets it to watch the target directory. Then, the user starts the service(s) and exercises its functions manually. Any access events will be recorded by the logger. Finally, the user can leverage the generated log file to determine the least permissions necessary to enable application functionality.

3.3 Visualization Platform

The final component of AutoPS is the visualization platform for the fanotify log. We generate a bipartite graph from processes to files, indicating whether files were read, written to, or executed by processes in the developer's application. Mouseovers highlight what processes touch a specific file or what files a process touched. With this tool, a developer who is concerned about having overly permissive accesses in their configuration can easily see what files a service actually touches during a trial run. They can then restrict the program's access to only the files it actually touched, providing for better security in the event that service is compromised. The developer can furthermore quickly see if some service(s) accessed a large number of files, and possibly decide to separate those services into smaller services to minimize the damage resulting from a single service's compromise. Likewise, they can identify critical files that many services rely on and possibly split those files into smaller files. The visualization tool is implemented as a Python script that parses the fanotify logs into JSON. We then render the JSON in a local .html file using the D3 [3] visualization library and automatically open it in the user's browser.

3.4 Further Details

Detailed documentation can be found in the README on Github, at `https://github.com/amruthv/AutoPS`.

4 Testing the System

To evaluate the Permissions Helper component, we made an application similar to the services offered by the Zoobar web application in Lab 2. Ignoring the web frontend of the application does not diminish the ability to demonstrate the ability to privilege separate the services that power the Zoobar application.

The application we built is called DemoApp. A copy of the source code can be found on Github: `https://github.com/amruthv/AutoPS`. The application itself offers a CLI for a client to register and login for the system. Once logged in, the client is allowed to make transactions between users as well as view their balance. Users are also allowed to store a secret they can view or replace at any time. Following the model of Lab 2, we want to separate the conceptual differences between authentication, secret storing, and banking. All interactions from the client are sent through RPC servers offering one of authentication, banking, and secret management. The authentication service is an RPC client to the banking and secret storing services to initialize entries for a user.

We then wrote a configuration file located at demoApp/config.txt that specifies the kinds of accesses that are needed by each of the clients, servers, database interface, and the CLI. Finally we run PermissionsHelper on the application and evaluate its ability to 1) function and 2) successfully privilege separate the application. By inspection, the application functioned as intended. Furthermore, to directly verify that our system set ACLs on the specified files as intended, we wrote a Python test script that simply spawns processes with various UIDs: the ones given access to each database (one each for banking, authentication, and secret management) as well as an unrelated one. We then attempted to open each of the databases for reading and writing under each UID, and verified that the intended UID, and only that UID, had read and write access to each database.

5 Future Work

While AutoPS offers an intuitive way to specify an application as well as evaluate privilege separation, there is one significant area for improvement. An issue that we found is specifying the configuration file can be challenging because of the need to trace back dependencies. For example, consider a python script that should be executed that imports other non-packaged modules. The developer would need to trace back through each of the imports and add them to the Read section for that python script for the application to successfully run.

Instead, an automated system that detects file dependencies for certain types of files (like python scripts) would help ease the burden on the developer to manually trace through his code.

6 Conclusion

The contributions of AutoPS are twofold: a simple way of marking the layout of a system for automatic privilege separation as well as two methods for evaluating system privilege separation. With the suite of tools offered, we believe that developers should find it easier to write more secure applications without relying on their applications never being compromised.

7 References

[1] OKWS: http://css.csail.mit.edu/6.858/2014/readings/okws.pdf
[2] fanotify: http://man7.org/linux/man-pages/man7/fanotify.7.html
[3] D3: http://d3js.org/