GOOGLE SUMMER OF CODE - 2018

The FreeBSD Project
Regression TestSuite for Audit Framework

Project Overview and Description

FreeBSD is a rapidly developing operating system with an extreme focus on advanced security & networking features. For an OS with a widespread usage and development, testing and monitoring of security regressions becomes a critical measure. FreeBSD has an audit subsystem which is responsible for monitoring a variety of security-relevant system events, such as user-logins, configuration changes, file system & network access etc. Even though the audit subsystem is indispensable for many security concerned organizations running FreeBSD servers, currently there is no tool to test its reliability and the intended behavior.

The project aims to develop a self-contained regression test-suite which will evaluate the audit framework for the proper logging of most auditable system calls along with the file format of BSM/XML/plain-text output, testing a wide range of functionalities of a particular event and then finally reporting the results and shortcomings if any. Kyua will be used as the automation tool, which will facilitate the regression testing of entire operating system at once, `audit`(4) included. An attempt will be made to stick to the supported architecture of the FreeBSD test-suite while writing test-cases, which will maximize the transparency of integration in the source tree.

About "FreeBSD TestSuite"

FreeBSD has a set of tests under `src/tests` powered by the kyua framework, which supports both plain tests as well as tests written against the `atf`(7) libraries.

About "Kyua framework"

`Kyua`(1) is a testing framework for infrastructure software, originally designed to equip BSD based operating systems with a test suite.

Potential Mentors	Email Address
Alan Somers	asomers@freebsd.org
Robert N.M. Watson	robert.watson@cl.cam.ac.uk
George V. Neville-Neil	gnn@freebsd.org

The Problem

Despite its importance in Operating System Security, the audit framework does not have a test-suite. While Linux maintains a test-suite for its own audit framework, it fails to test for events other than basic system calls and has many cross-platform dependencies. The situation is even worse in FreeBSD which has no testing whatsoever.

Imagine a FreeBSD server responsible for maintaining audit logs of the system activity. Suppose the system administrator configures the server with a separate filesystem (generally `/var/audit`) for maintaining audit trails. Let's brainstorm the events where things may go wrong.

- A change elsewhere in the source code causes a regression, and the events that were properly audited before stop being logged.
- Admin does not filter relevant events resulting in huge amount of audit trail, eventually exceeding the file-system capacity.
- Discovery of new vulnerabilities with no way to check the possible exploitation of the server.
- A dangerous system call which can compromise the whole infrastructure, e.g `chmod`, `setuid`, `execve` etc, not being audited.
- Accidental misconfiguration of audit daemon configuration file resulting in the collapse of the audit system.

With proper testing of the utilities responsible for auditing these events, it would be possible to monitor the integrity of the entire audit system as and when required, without the system administrator having to check on its individual components frequently.

Overview of Event Auditing

- **How is event auditing implemented?**

FreeBSD uses OpenBSM, an implementation of Sun's BSM event auditing file format and API. OpenBSM is made up of a plethora of tools including audit-viewer applications, e.g `praudit`(1) & `auditreduce`(1) as well as the `libbsm`(3) library to provide an interface to BSM audit record stream. The `auditd`(8) daemon is responsible for managing the kernel's `audit`(4) mechanism and also rotates the log files whenever required.

- **How to enable system event auditing?**

FreeBSD already has userspace support for audit system. The audit daemon, `auditd`(8) can be enabled by adding the following line to `/etc/rc.conf`

```
>>> echo 'auditd_enable="YES"' >> /etc/rc.conf
```

A safer way is to use `sysrc`(8)

```
>>> sysrc auditd_enable='YES'
```

Then start (and stop) the audit daemon, with a new audit trail:

```
>>> service auditd start && audit -n;
>>> service auditd stop
```

- **Various configuration options in audit framework**

```
>>> dir='/etc/security'
```

Audit configuration is defined in `${dir}/audit_control`. Various event selection expressions are defined here, also found in `${dir}/audit_class`, which lets a user configure the events to be audited. For example,

- **lo** for 'login-logout'
- **pc** for 'process'
- **nt** for 'network'
- **ad** for 'administration' and many more...

`${dir}/audit_user` specifies events to be audited for each system user, like login attempts. `${dir}/audit_event` contains a list of all auditable system events.

Implementation Details

The tests will span from simple permission check of the configuration files to exhaustive system call testing. The system calls in question will be categorized in TCP/UDP network sockets, process control, file & IO management. I will maintain a priority order of the implementation of the tests after discussion with my mentors.

In addition to the application program which triggers all processes, there will be a set of ATF test cases which will be created in accordance with FreeBSD Test Suite. A self-contained automation infrastructure will be developed which will enable the independent and ad-hoc testing of the audit system.

Test Plan and Current Work

On comprehensive discussion and approval from the mentors, I implemented an initial version of the audit test-suite, (https://github.com/aniketp/AuditTestSuite).

Report of the approach and implementation:

- Report1: Network System Call testing
- Report2: File-read System Call testing
- Report3: Kyua test-program for mkdir (current approach)

Initial Approach : (deprecated)

The Audit Test-Suite creates a TCP Server, connects & sends a message through `telnet` and closes the connection. In the whole process, it is able to trigger 9 network system calls defined by audit_event with "`nt`" audit_class. Next, it launches a UDP server and a client which communicate with each other through `sendmsg`(2) and `recvmsg`(2).

The script then parses the audit trail so produced for the log of the corresponding system calls and checks whether both instances of system calls were present or not (one being correct and other being incorrect with wrong socket descriptor). The final result is published as a nicely formatted report displaying the statistics.

Instructions for Testing

To run the tests on a FreeBSD machine, first set the correct configuration environment.

`>>> ./setup` **Note:** This script will be same for most of the tests

The automation script (`socket/run_tests`) can be launched by the following command :-

```
>>> make && make run
```

The script produces the output along with the statistics as shown in this example.

List of System calls tested

These system calls were tested for the proper audit in both success and failure modes.

S.No	System Call	Success	Failure		S. No	System Call	Success	Failure
1	socket(2)	✔	✔		13	open(2)	✶	✶
2	bind(2)	✔	✔		14	openat(2)	✔	✔
3	setsockopt(2)	✔	✔		15	readlink(2)	✔	✔
4	listen(2)	✔	✔		16	readlinkat(2)	✔	✔
5	accept(2)	✔	✔		17	symlink(2)	✔	✔
6	sendto(2)	✶	✶		18	symlinkat(2)	✔	✔
7	recvfrom(2)	✶	✶		19	mkdir(2)	✔	✔
8	connect(2)	✔	✔		20	mkdirat(2)	✔	✔
9	sendmsg(2)	✔	✔		21	mkfifo(2)	✔	✔
10	recvmsg(2)	✔	✔		22	mkfifoat(2)	✔	✔
11	link(2)	✔	✔		23	mknod(2)	✔	✔
12	linkat(2)	✔	✔		24	mknodat(2)	✔	✔

✶ : BSD libc converts these system calls to their complements (e.g open(2) to openat(2)) to optimize resource usage in specific cases. Their testing was done using `syscall`(2), i.e :

```
>>> syscall(SYS_open, “path/to/file”, O_RDONLY);
```

Developing stand-alone Kyua tests using `atf-c-api`(3) : (Revised Test Plan)

While the above approach is useful for independent testing of the audit system, it becomes redundant if we consider that the final goal is to be able to integrate the tests into FreeBSD source tree. To accomplish it, developing the tests using **atf**(7) libraries is recommended.

Another advantage is that it eliminates the need for a separate automation tool. Stand-alone tests can be automated using Kyua, which can then enable the regression testing of the entire operating system at once, **audit**(4) system included.

Testing mkdir(2) using the new approach

The test program here is an initial attempt at creating independent tests for similar system calls which can then be automated by Kyua. The independence is achieved by following these steps:-

1. In a situation where the system does not have the audit system enabled, the tests would usually fail or skip. However, the audit daemon can be started (and stopped) by "`service auditd one{start, stop}`" without modifying `/etc/rc.conf`. This eliminates the need for a setup script as described in the deprecated test plan.

2. Create separate functions for the startup and termination of the audit daemon. This step also takes into account the possibility of **auditd**(8) running beforehand. If so, the process termination is avoided to maintain the current state of the machine. The running status can be checked by "`service auditd onestatus`".

3. Audit system also allows live auditing via **auditpipe**(4) for IDS and testing purpose. This enables us to open `/dev/auditpipe` and verify the proper audit of the system call in question. Usually in cases like these, **poll**(2) helps in confirming if the opened file descriptor is ready to perform I/O operations.

4. The system may not have the proper settings in `/etc/security`. This would fail the events not covered by "audit_control" configuration file. However, **auditpipe**(4) describes quite a few **ioctl**(2) requests in audit_ioctl.h for such scenarios. **getauclassnam**(3) can be used to obtain required information from the **audit_class**(5) database.

List of Testing Scenarios (In order of priority)

1) Explicit System Call Testing

This application would consist of a set of test programs written in `atf-c`(3), containing independent test cases for triggering similar system calls, e.g open(2) & openat(2). Each program would test a wide variety of functionalities of corresponding system calls. Testing scenarios would ensure that the audit record contains all expected parameters, e.g the arguments, valid argument types, return values etc. The testing will be done for various success and failure modes, with cross-checking for appropriate errno codes in case of failure mode.

2) Format of BSM/text/XML File Output

This test will check if the audit-reduction of log file has all the necessary information regarding the event audited like `header`, `event`, `attribute`, `subject`, `return`, `trailer` etc. The tests might vary depending on the attribution of the event, audit records and audit classes. A necessary semantic check will also be done.

Optional Section: Time-Permitting

1) Event Selection Expression (Audit Class)

Selection expressions are used in a number of places in the audit configuration to determine which events should be audited. Expressions contain a list of event classes to match. A test automation can be done to ensure each of the `20` class expressions behave as expected and log the events they are supposed to audit.

Note: For the current scope of the project, only the first two tests will be implemented. Later on (if time permits), an attempt will be made to create tests for rest of the scenarios on a priority basis from the optional section and the following document.

- ➢ Brainstormed List of Testing Scenarios

Expected Results

Final goals which are expected to be completed by the end of the summer are:-

- Set of `atf-c`(3) test programs integrated into Kyua framework which will evaluate the robustness & exactitude of the FreeBSD audit system.
- An `atf-sh`(3) test program for the file format check which will be automated by Kyua and will evaluate the integrity of BSM/XML/plain-text output audit trails.

Deliverables

Deliverables for the first evaluation

- Set of tests for a small subset of the system calls supported by OpenBSM.
- Initial work done on writing `Kyua` compatible test programs, presumably using `atf-c-api`(3).
- An inceptive automation tool for running all the tests developed till now.

Deliverables for the second evaluation

- A comprehensive and automated test-suite covering “most” system calls.
- Stand-alone tests for all system calls tested till now, to facilitate regression testing of the entire operating system at once.
- Set of test cases for checking file-format of audit-trails (text/XML/BSM).

Deliverables for the third evaluation

- Finalization of independent and stand-alone tests for audit system, arranged in the appropriate directory structure.
- `atf-sh`(3) & `atf-c`(3) tests for all audit tracepoints integrated into Kyua framework.
- An attempt will be made to test the integrity of `audit_control`(5) configuration file. (If time permits).

Project Schedule

April 23 - May 13 (Community Bonding Period)	Get familiar with the intricacies of Security Auditing. Familiarize with the accepted conventions in FreeBSD as well as the technologies to be used in the project. (Coding conventions, documentation, communication etc)
May 14 - May 20 (Week 1)	Classify all auditable system calls in the corresponding audit_class. Write test scripts to confirm the successful audit of each system call.
May 21 - May 27 (Week 2)	Convert the current method of retrieving audit trails by `praudit`(1) to a native `libbsm`(3) api. Use several `au_fetch_tok`(3) calls to get all the tokens for a record, which can be read using `au_read_rec`(3). This step may vary for each test case of a system call.
May 28 - June 3 (Week 3)	Initiate the merging of similar and complementary system calls in a single test program. Check if the sequential trigger is being audited successfully. Replicate each system call with an invalid argument to generate test cases for failure modes.
June 4 - June 10 (Week 4)	Work on extending the tests for audit argument, return value and errno codes (if any) for both modes. Try to log the test failures in a separate file, giving an explanation of why the tests failed.
June 11 - June 17 (Week 5)	(Buffer Period) : Catch up with any work left from previous weeks. **--------Phase 1 Evaluation------**
June 18 - June 24 (Week 6)	Extend the test-suite to collect the XML & plain-text outputs separately. Write a script to check the correct syntax and semantics of the audit trail in both formats. Report any discrepancies to Bugzilla.

June 25 - July 1 (Week 7)	Initiate work on testing the BSM binary format of audit trails. If successful, integrate the functionality into output file-format test. Henceforth, develop a single `atf-sh`(3) test program for validation of the file-formats by Kyua framework. Continue the work on testing system call audit in parallel.
July 2 - July 8 (Week 8)	Extend all developed test-cases to support `atf-c`(3) and `atf-sh`(3) test scripts. An attempt will be made to keep it coherent with the recommended directory structure of the FreeBSD Test Suite.
July 9 - July 15 (Week 9)	(Buffer Period) : Catch up with any work left from previous weeks. If done, test some miscellaneous system calls. **--------Phase 2 Evaluation------**
July 16 - July 22 (Week 10)	Finish up the testing of as many auditable system calls as possible. Complete the standalone Kyua tests for all remaining cases.
July 23 - July 29 (Week 11)	Testing Phase: Rigorous testing of every developed test-case, automation tool & Kyua scripts in various build scenarios.
July 30 - Aug 6 (Week 12)	Refine the code to make it more presentable. Add final touches to documentation, code-formatting. Preparation for the submission of the project.

The Code

The source code will be kept updated at - https://github.com/aniketp/AuditTestSuite

Biography

I am a sophomore at Indian Institute of Technology Kanpur, majoring in Mathematics and Scientific Computing.

I'm a security enthusiast, with a plethora of projects, internships and experience in writing secure and scalable web applications. I have above par understanding of Linux Systems, Network protocols, Cryptography and Operating Systems sandbox due to my highly concentrated work in the field of Computer Systems Security for the past few years.

My work experience includes interning as a Security Analyst at Lucideus Tech, where I employed various VAPT techniques to understand and mitigate security risks in network systems.

Having started early on this project and after thorough discussion with my mentors, I went ahead and automated the tests for many socket and file I/O system calls. Furthermore, I also wrote a Kyua test program which sets up the environment for audit testing. Hopefully, this will enable me to deliver more than what is feasible during the summers.

Projects/ Open Source contributions

My previous projects, relevant to Systems/Web Security are -

- **Gymkhana Nominations Portal** (Github link)
  Used: (Python/Django, Rest API, PostgreSQL)
  A secure and scalable web application written in Django framework with PostgreSQL for the elections of students' government organization of my institute.

- **Crunchie, Wordlist Generator** (Github link)
  Used: (Lua, Nmap libraries)
  An implementation of Crunch in Lua. Working on integrating it with Nmap Scripting Engine's SSH, HTTP, FTP brute-force library to generate words on the fly.

- **Network Programming Projects** (Github link)
  Used: (Python, C, Sockets)
  Set of small projects on creating a Reverse Shell, Packet Sniffer, Website Scanner and echo time server for learning the elements of Network & Web Security.

My Open Source contributions include -

- ➢ **Nmap Security Scanner** (Contributions) [Languages: `C, Lua`]
  Worked on improving some aspects of the http-fetch script, like separating download directory for distinct host:port combination for a given destination (link). Corrected the XML output in usage doc (link). Fixed the socket descriptor parsing issue in the FTP library (link). Resolved the issue of recursive fetching of identical web pages from separate ports (link).

- ➢ **Astroplan - Observation Planning Tool** (Contributions) [Language: `Python`]
  Minor contributions on improving constraints and observer modules. Profiled various Astroplan classes to identify the reason for latency while scheduling. (code)

- I have also reported a bug on FreeBSD bug tracker about the conflict of argument types of audit `ioctl`(2)s (link), and contributed to OpenBSM. (link)

Availability/ Working Hours

My summer vacation will be from April 29 to July 30, which completely encloses the duration of Community Bonding & Coding period (except the last week, which will be the beginning of the semester with little workload). During the entire duration of the project, I will be persistent on IRC and will be available for work all the time when I'm awake.

Contact

Name : Aniket Pandey
E-mail : aniketp@iitk.ac.in
Github : aniketp
Phone : +91 9599881876
IRC : `aniketp41` on freenode & efnet
Blog : http://blog.aniketpandey.com/
Address : B-210, Hall of Residence: 12
Indian Institute of Technology
Kanpur, Uttar Pradesh
India - 208016