Creating a Software Testing Environment Using FreeBSD

Chris McMahon <christopher.mcmahon@gmail.com >
Revision: 43184

FreeBSD is a registered trademark of the FreeBSD Foundation.

Intel, Celeron, EtherExpress, i386, i486, Itanium, Pentium, and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Microsoft, IntelliMouse, MS-DOS, Outlook, Windows, Windows Media and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Symantec and Ghost are registered trademarks of Symantec Corporation in the United States and other countries.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document, and the FreeBSD Project was aware of the trademark claim, the designations have been followed by the "™" or the "®" symbol.

2013-11-13 by hrs.

Abstract

A slightly altered version of this paper was first published in the Pacific Northwest Software Quality Conference Proceedings, 2005.

Table of Contents

1. Overview 2
2. The Challenge 2
3. The FreeBSD Solution 2
4. Conclusion 7

1. Overview

From late 2003 until early 2005, I was a tester in an all-Windows® environment. Although unlikely on the face of it, FreeBSD became a valuable test tool platform in that context. FreeBSD contains useful and powerful applications for any tester in any environment.

Unlike Linux, FreeBSD is a single monolithic project, rather than a collection of disparate parts assembled into a distribution. And the most attractive part of FreeBSD for a software tester is the {{{os}}} {{{ports}}}—a very large, managed set of software applications with a single, simple, and uniform installation procedure.

This paper describes several software test tools from the {{{os}}} {{{ports}}} that I used to test software and systems in an all-Windows environment.

2. The Challenge

Software testing environments are radically more complex than software development environments. Interconnected systems to test, network entities, databases, and filesystems present challenges to testers that developers can, for the most part, mock out and essentially ignore. Software testers need more tools, and more complex tools, than do software developers.

On the other hand, software development tools are much more highly evolved than software testing tools. There is no Eclipse, or IntelliJ, or even Visual Studio aimed at testing. Testers struggle and scratch to find tools appropriate to their test environments and appropriate to their Systems Under Test (SUTs).

Every test environment is different. Finding appropriate test tools is challenging. And when a tester has identified a set of useful tools, introducing them to the test environment and integrating them with the test environment is a challenge.

3. The FreeBSD Solution

3.1. Introduction

The set of tools available with the FreeBSD Operating System is amazing. The FreeBSD Ports Collection contains more than thirteen thousand separate applications, all of which have a standard installation procedure and conform to a set of guidelines that make them reliable without the need to manage dependencies, appropriate versions, and all of the other problems that affect even the most well-managed Linux distribution or the various versions of Microsoft Windows. The monolithic nature of FreeBSD and the {{{os}}} {{{ports}}} removes much of the trouble of integrating tools with the test environment, regardless of the OS under which the SUT runs. FreeBSD is a highly evolved server envi-

ronment, and contains so many reliable applications, that every tester should consider adding a FreeBSD machine (or several) to their test environment.

Of course, all of the applications available in the {{{os}}} {{{ports}}} will not be appropriate for any single test environment. Some of the obvious choices for software and systems testing are the six hundred or so system utilities, the more than one thousand network tools, and the fifty-odd benchmarking tools. Whether your test environment is Windows, UNIX, Linux, Mac OS, FreeBSD itself, or some combination of any of them, FreeBSD and the {{{os}}} {{{ports}}} is a great place to look first.

3.2. How To Use The Ports System

Installing an application from the {{{os}}} {{{ports}}} is a simple matter of:

```
# cd -/usr/ports/foo
# make install
```

and the system does the rest. It reports build status and test status, and installs all the relevant documentation as well. This aspect of FreeBSD is very attractive to a tester, who typically is pressed for time!

3.3. FreeBSD For Testing

The test environment should be more stable than the SUT. Once the tester decides to use the tools available on FreeBSD, FreeBSD's long record of reliability makes it an easy choice for a test tools platform.

My own introduction to FreeBSD occurred when I was hired by a major vendor of large-scale network security video services to be their network-testing person in an all-Windows environment. My first assignment was to replace the obsolete, buggy, disk imaging system. I chose to do that with an Open Source disk imaging system called Frisbee which was implemented originally on FreeBSD. I built the system—a feature-for-feature replacement for an expensive proprietary system—but we never actually used it in our production system.

In the meantime, I had discovered the {{{os}}} {{{ports}}} and started to use some of those tools for testing; and I had discovered the power of disk imaging with Frisbee, especially for smoke testing and installation testing; and FreeBSD became a permanent part of my test lab. The test lab I built, and the FreeBSD systems I created still exist, and still provide value to the testers there.

3.4. FreeBSD For Collaboration: Twiki

A wiki is a simple set of web pages to allow many users to share information and collaborate on any sort of documents. Twiki is a wiki fine-tuned for document management. It has built-in version control and security implemented at the user/password level. I used it for requirements management and traceability.

The company I worked for managed their requirements in a proprietary tool with a truly byzantine hierarchical structure. Although the tool was fine for generating requirements, it was horrible for testing. Also unfortunately, the tool supported very few export mechanisms. From the proprietary tool, I had to:

- Export to Access
- Export from Access to Excel
- Export from Excel to a delimited file
- Import the delimited file to the wiki by means of a Perl script

And in Twiki I was able to implement version control, traceability, and test coverage, all features missing from the proprietary tool. This gave me a remarkable ability to trace requirements coverage with my tests. Having imported a complex hierarchy of sets of requirements, I could use Twiki's simple markup features to show each requirement as tested, passed, or failed. I could attach test documentation to wiki pages, and since I had already scripted the process, it was easy to update the requirements as they changed. And Twiki itself handled version control for such updates.

As with all of the examples in this paper, installing Twiki on FreeBSD is fairly simple. It takes just a few minutes on a FreeBSD system. However, if you want to use Twiki on a Microsoft Windows platform, I strongly suggest you read the Twiki documentation extremely carefully. I know someone who installed Twiki on Windows, and it took him several days. Twiki on Windows requires not only knowledge of Windows, but also deep knowledge of Cygwin and Perl.

Furthermore, at one point in the project I had to migrate my wiki from a machine running FreeBSD 4.8 to one running FreeBSD 5.3. The migration consisted merely of installing Twiki on FreeBSD 5.3; using `tar` on the FreeBSD 4.8 machine to gather all of the Twiki data files specific to my testing; FTPing the gathered files to the new FreeBSD 5.3 machine; and untarring the file. The complete set of Twiki documents migrated with no issues or problems at all. That is the power of a unified system like FreeBSD.

3.5. FreeBSD For Disk Imaging: Frisbee

A disk imaging system is a mechanism for saving and restoring all of the data on a physical disk. The most popular commercial system for doing this is probably the product Ghost™ from Symantec.

The Frisbee enterprise disk imaging system mentioned above had a lot of features I never implemented in the test lab. Using Frisbee and an Open Source tool called PXELINUX, I was able to:

- Boot the Windows client machines from the network

- Make an image of the client
- Update the BIOS on the client
- Lay down an existing image on the client machine
- Make a set of restore CDs for the client

In the test lab, I only needed to boot from the Frisbee CD, make an image, or lay down an image on the client machine. Both Frisbee and proprietary imaging systems allow the user to image individual drives on the client, but I never had a need to do this.

Installation testing was a large part of my duties at the company where I used FreeBSD. To do this testing, I would typically use Frisbee to make an image of a machine containing only a Windows OS, install the SUT, and run a smoke test. The smoke test typically left the test machine in a very bad state. But instead of having to painstakingly clean up the mess left by the failed installation, I simply re-imaged the machine in question with the bare OS image and started over. A typical re-image containing only the Windows OS and a few test tools took less than three minutes. Using Frisbee, we could run smoke tests on about six builds per day; before Frisbee, we could run smoke tests on about three builds per week.

Of course, Ghost or other proprietary tools also image machines quickly under these circumstances: once you buy the tool, license the software, install it on an appropriate server, and configure it properly. I prefer Frisbee to Ghost because: Frisbee is marginally faster; Frisbee is very easy to install on FreeBSD; and Frisbee is very efficient. Adding a couple of small Perl scripts to the normal Frisbee distribution gave me an imaging environment tailored for the test lab.

I also used Frisbee to preserve the state of a machine after I had uncovered particularly complex defects. That is, if it took a large effort (many steps and/or a long duration of time) to demonstrate a defect, I could make an image of the machine at the point at which the defect was visible, and restore the image at will to demonstrate the defect to developers or managers.

3.6. FreeBSD Security Testing: Nessus

Whenever you have more than one entity on a network, and whenever you expose a server to the wider Internet, security of the machine itself is always a concern. Nessus is an Open Source remote vulnerability scanner for security and penetration testing that consistently is rated among the top products of its type throughout the security industry.

Nessus probes a remote machine over the network for security vulnerabilities. It does a port scan, finds which ports are open, and investigates the software that has those ports open for a huge number of security risks, for all major OSs. It generates detailed reports in a number of formats that anyone can understand. The number of security probes available in the default installation of Nessus is very large, but sophisticated security and pen-

etration testers take advantage of NASL, the Nessus Attack Scripting Language, to craft their own attacks using Nessus' available features.

Of interest is that, while Nessus is a free download for UNIX-like systems (and is available in the Ports Collection of FreeBSD), it is available on Windows only as a commercial product from a company called *Tenable*. The Tenable product is NeWT, "Nessus on Windows Technology".

3.7. FreeBSD Network Tools

FreeBSD is most widely used as a robust server platform. It follows, then, that tools related to network analysis and performance will be highly evolved on FreeBSD. Here is a brief description of network diagnostic tools that I found invaluable in testing in a networked environment.

From the name, one would assume that ntop emulates the functions of the UNIX top(1) command, but for the network rather than for the local machine. Perhaps the first version did; currently, ntop is capable of providing detailed information about a huge number of hosts and their status and activities on the network.

For testing, two features I found very powerful: at a high level, ntop shows the amount of network traffic on the entire network segment minute-by-minute, hour-by-hour, and day-by-day in a graphical format. Also, ntop shows information about recent connections between individual hosts on the network.

It is easy to see traffic trends on the network as they are occurring; also, if something anomalous appears, ntop records detailed information about network connections between hosts, including the ports over which the connection happened. This was critically important when analyzing software issues. If ntop showed a period of time for which traffic was particularly high, I would find out which host was generating the traffic. I would examine the software running on that host, over that port. Often it was a new build with a bug.

Ettercap is a tool for ARP poisoning which can also decipher passwords on the fly and corrupt IP traffic by means of a Man In The Middle (MITM) attack. However, I used Ettercap as a performance tool. In my test labs, all of my FreeBSD machines ran on discarded hardware, Pentium II processors. I found that when I used Ettercap to sniff traffic between two hosts, the lack of processing power caused Ettercap on the slow MITM machine to start dropping packets, making it look to the client machine in the SUT as if there was interference or other trouble on the network. And by varying the load on the FreeBSD machine, I could in fact control the number of packets being dropped: running Ettercap and the UNIX `yes` utility caused 100% packet loss.

This was my most creative use of a FreeBSD tool for testing. In a more straightforward application, any time a tester needs to eavesdrop on traffic between two hosts on a network, Ettercap is an excellent choice because of its power and ease of use.

Perl gets a special mention here because Perl's network utilities are outstandingly good compared to other languages. Perl `Net::*` modules and `IO::Socket::*` modules are robust and powerful—but they often fail to compile on Windows. It is the ease of use of Perl's network utilities on FreeBSD that gets Perl the mention in this section.

I use Perl's network utilities to impersonate network clients and servers for test purposes. On one occasion, I was required to test software that was a client to an interface on the *New York Stock Exchange*. Unfortunately, the NYSE test server was down about nine days out of ten. I wrote a little network server in Perl to emulate simple functions of the NYSE server in order to test the client software.

On another occasion, I had to test some functions on a web server. It was not difficult to write a simple Perl HTTP client to validate that the server was functioning properly.

I have also used Perl to validate the output from a server sending to a multicast address. I wrote a simple Perl multicast client on FreeBSD to monitor the traffic on a multicast address. Lincoln Stein's excellent `IO::Socket::Multicast` module made it easy. (Note: I never got `IO::Socket::Multicast` to compile on Windows. I tried it on Windows 2000 and on Windows XP.)

4. Conclusion

I used tools from the {{{os}}} {{{ports}}} in four areas: in the network, where the operating system has very little impact on how software behaves; for remote security testing and performance testing in order to manipulate remote machines over the network, regardless of the operating system; for disk imaging of Windows, Linux, and FreeBSD machines; and on the webserver, where Twiki was my collaboration tool of choice.

Because the installation procedure for all of these tools is standard, I spent relatively little time installing and configuring my tools. Because all the tools were hosted on a single platform, I had all of my configuration and diagnostic information located in just a few places. I kept all of my potentially dangerous security tools on a single machine, which made my presence on the network tolerable to the company's network management staff. And the compatibility between FreeBSD versions made it fairly simple to upgrade and to manage multiple FreeBSD machines. And of course, I could rely on the correctness of my test results, because the system itself is so reliable.

I have tried using Linux in a similar way, but my experience is that package management quickly becomes tedious if not overwhelming. The {{{os}}} {{{ports}}} handled that for me. And many of these tools are simply not available on Microsoft Windows. And when they (or their equivalents) are available, their cost, both financial and in terms of overhead, was simply too high.

FreeBSD's simple installation procedures and robust Ports Collection makes it easy to experiment with the huge number of tools available. I often find myself browsing the

Ports Collection looking for interesting applications to install, just to see how they work. (I found Ettercap by browsing the Ports Collection, a tool that became very useful very quickly.) It became clear that the more tools I used on FreeBSD, the more economical became the management of those tools.

The next time you need to reach into your toolbox for some sophisticated, reliable, and powerful testing tools, I hope you find them in FreeBSD.