

IAnewsletter

The Newsletter for Information Assurance Technology Professionals

Look out! It's the fuzz

IATAC



also inside

ESSG

IATAC Spotlight on Education

IATAC Spotlight on Research

Ask the Expert

A Snapshot of Some Current CERIAS Research

6th Annual Department of Defense (DoD) Cyber Crime Conference

An IATAC/DACS State-of-the-Art-Report on Software Security Assurance

The Morphing of a Cyber Operations Curriculum at the Air Force Institute of Technology

Look out! It's the fuzz!

by Matt Warnock

When you hear the term “fuzz,” you may think of a delicious peach, or the guitar sound in early Cream or Jimi Hendrix albums, but software fuzzing is a relatively new software auditing technique responsible for finding many of the bugs and security vulnerabilities found in utilities, software applications, and network protocols. To understand what fuzzing is, we need to understand how fuzzing originated.

“It started on a dark and stormy night,” is actually how the authors of the paper “An Empirical Study of the Reliability of Unix Utilities” [1] describe how they stumbled on the technique used in software fuzzing. This article is first in a series of software fuzzing papers from the University of Wisconsin over the past 15 years. The first paper tells the story of a user connecting to a server over a modem connection, and a storm causing noise

Sending random characters to a program is the original and simplest form of software fuzzing—often called simple, or generic fuzzing

on the phone line, and the noise creating random characters on the screen. This phenomenon is very understandable, but the most interesting aspect of it was that the random characters actually caused programs to crash and hang.

These four papers were instrumental in building the groundwork for software fuzzing. While these papers laid the ground work, now many research projects and utilities have been released to aid software auditors with testing. Several papers have been written on the subject, however it is still very new. The first article in the IATAC IA Digest to mention software fuzzing appeared on 6 Feb 06 entitled “The Future of Security Gets Fuzzy” [2] and shows how cutting edge this topic is.

Definition

Sending random characters to a program is the original and simplest form of software fuzzing—often called simple, or generic fuzzing. The random characters are sent via Standard Input (STDIN), in a command line utility. Characters could be standard American Standard Code

for Information Interchange (ASCII), extended ASCII, control characters, null spaces, or any combination of these. When a certain combination of characters are sent, the program may crash, or exit without a proper exit code, hang, or

loop indefinitely, or exit with a proper exit code. If the program crashes or hangs, the output of the fuzzer is reviewed to see exactly what combination caused the crash, and then program is analyzed to see what caused this. Simple fuzzing is very easy to perform as it does not require much prior knowledge of the application, however, it can take longer to find bugs and will not search every aspect of it. Intelligent fuzzing, usually required for advanced programs or network protocols, must take into consideration the structure of data and only fuzz certain portions, and leave the rest untouched. Things like checksums must be considered when creating random data, or the program may reject the data because it is not calculated properly. Programs that input files can also be audited by using file fuzzing, which inputs files with random data to see if they are loaded into the program. Even Application Program Interfaces (API) are vulnerable. API fuzzing sends random data to the common code used between programs to find bugs in reusable code. Also, web browsers, which accept HyperText Markup Language (HTML) data and translates it into visually understandable layouts, can be fuzzed. Malformed HTML data often leads to browser crashing.

Fuzzing is used primarily as a software auditing technique, and is one of several auditing methods that includes reviewing source code (precompiled),



reviewing binary code (post-compiled), and reviewing API calls. Fuzzing allows an engineer to analyze software with minimal application specific knowledge. Although it is a technique that provides a capability to discover numerous bugs, it is very difficult to find every bug using this method alone. [3]

Types of Fuzzing

Unix Utilities

In their first paper, the researchers at the University of Wisconsin created the software auditing technique, and also tested their technique on several Unix command line utilities. Command line utilities usually input data through the STDIN via the Unix pipe. Their fuzzing test was run using the follow format: [1]

```
fuzz 100000 -o outfile | pttyjig vi
```

The tool that created the random data is called fuzz and the program to simulate the terminal is called pttyjig. This creates fuzzed data of maximum length 100,000 bytes, the output is saved as outfile, and the utility tested is vi, a text editor. To audit Unix command line utilities, very little needs to be known about the utility. Random data of varying characters and lengths is passed to the utility to test it. Because of this, the technique is very simple, but still very effective. The researchers implemented their

auditing technique by using a tool to create random data and log it, and a tool to simulate the terminal. Also a script to automate the process was created. They tested 88 utilities in seven versions of Unix. Their techniques caused 24–33% of the utilities to crash or hang. They also tested the technique against network services, but were unable to crash any. The details of the results of their tests can be found in their original paper.

Five years after their original paper was published, researchers at the University of Wisconsin used their software auditing techniques to review the same Unix command lines with a few new operating systems, to include Linux. In this paper, “Fuzz Revisited: A re-examination of the Reliability of Unix Utilities and Services” [4] 80 utilities on nine versions of Unix were tested. Even after revealing many software bugs, the same bugs were found in the software tested five years later. In fact, on commercial versions of Unix, the 15–43% of the utilities crashed or hung. On the free versions of Unix, 9% had flaws, and GNU’s Not Unix (GNU) style utilities had the lowest crash rate, at 6% of the GNU utilities tested.

Graphical User Interface (GUI)

Unix

The second University of Wisconsin paper also addressed programs for X Windows, the Graphical User Interface (GUI), for the Unix operating system. Now, in order

to test X Windows applications, random data must be sent to the program via x-events, not STDIN. To do these tests, the researchers sent random, unformatted data to the X Windows programs, as well as random data sent as legal x-event streams. The random data caused 50% of X Windows applications to crash and the random data in the form of legal x-event streams caused another 25% to crash. They were not able to crash the X Server nor any network services. The details of the test results can be found in their original paper. This paper shows that fuzzing techniques are not limited to command line utilities, and now GUI applications, as well as other types of services, can be successfully audited.

Windows NT

Next in the series of papers from the University of Wisconsin is a paper [5] on using software fuzzing to test Windows NT (and Windows 2000) applications. This paper, published in 2000, details the results of testing their fuzzing techniques on Windows NT applications. Thirty GUI applications were tested including Microsoft Office 97 and 2000, Adobe Acrobat Reader, Eudora, Netscape 4.7, Visual C++ 6.0, Internet Explorer (IE) 4.0 and 5.0, as well as others. This time, valid keystroke and mouse events were simulated, as well as the Windows messages, or Win32 messages. Random Win32 messages were sent to the applica-

tion to see if they would crash or hang. The random keystroke and mouse events caused 21% of applications to crash, and 24% more hung. Also, when random Win32 events were sent to the applications, all applications hung or crashed. The researchers described this as a problem with the Win32 messaging system. The full results of their tests can be found in their original paper.

Apple Mac OS

To date, the last paper in the University of Wisconsin's series was on testing MacOS applications. [6] MacOS X is based on FreeBSD, and therefore contains many Unix-like command line tools, as well as GUI applications. In 2006, the researchers tested 135 command line tools, and 30 GUI applications, such as iChat and iTunes. They revealed that 7% of the command line tools crashed, however all but eight GUI applications crashed.

In this simple example, the first line is the original data packet. The second line is the fuzzing mask. In the mask, 0's are ignored and 1's are fuzzed. In this example, the IP source and destination are not fuzzed, the data is fuzzed, and the checksum must be recalculated. The packet is not entirely random, only portions of it.

Network fuzzing is any kind of fuzzing technique which tests software that is not on the local machine, or requires a network protocol. Fuzzing can test the network layer, such as Transmission Control Protocol/Internet Protocol (TCP/IP), or applications, such as File Transfer Protocol (FTP) servers. Leon Juranic's paper [7] details several bugs found in common FTP servers, and the techniques used to find the bugs. To test FTP, a valid network session must be created, and valid FTP commands must be sent. Other than this, random

File Fuzzing

File fuzzing, or file format fuzzing, is a method of auditing software where files are opened and data is extracted. Many programs input data through files instead of STDIN, but this requires file formats to be standard or the program to be robust enough to detect and detail with anomalies. File fuzzers create files with some portions containing random data, or formats that are not exactly standard. These could be in binary format, or ASCII format. After creating the random file, they will execute the target program and open the new file in it. File fuzzing was used to find the Buffer Overrun in Joint Photographic Experts Group (JPEG) Processing (GDI+) in Microsoft Security Bulletin MS04-028. [8] It is a vulnerability found in other places, however was never exploited by a file.

Application Program Interfaces

Application Program Interfaces (API), are pieces of reusable code that is executed by many applications. Windows uses APIs, such as the Component Object Model (COM). Since APIs are reused by so many programs, they are both available to everyone, and also makes everyone susceptible if a vulnerability is found in an API. Since they are potentially vulnerable, APIs too can be fuzzed to find these vulnerabilities. API fuzzers will scan COM or ActiveX object interfaces.

Browser Crashing

Another method of fuzzing involves a form of file fuzzing called browser crashing. Web browsers, such as Firefox, Netscape, and IE, convert HTML code into viewable content. While HTML has been made standard by the World Wide Web Consortium (W3C), a web browser must be robust enough to detect and deal with non-standard and erroneous HTML input. Browser crashing creates random HTML content, which is correct enough for the browser to detect and parse it, but may contain anomalies that could crash the browser. All browsers have been susceptible to this kind of audit.

While the methods of finding software vulnerabilities varies greatly between the different fuzzing techniques, in the end, many of the same types of programming mistakes are found

Network Fuzzing

While "dumb fuzzing" randomly sends data to an application, intelligent fuzzing requires only certain portions of the data to be fuzzed. An example of this is when legal X-events were sent to an X application, but the data within the events is random. This type of technique is required when fuzzing network applications as well, otherwise, the packet may be rejected, and the data cannot be tested. For instance:

```
IP Source | Destination | Checksum | Data123456789
00000000 | 00000000 | recalculated | 111111111111
```

data is sent. For his tests, he uses Infigo FTPStress Fuzzer which allows you to select a valid username and password, and which FTP commands to use. While sending a valid FTP command, and a random data string, the utility was able to crash several FTP servers, such as GoldenFTPD, WarFTPD, and Argosoft FTP server. The details of the tests are available in the original paper.

Other network fuzzing attacks occur at the network layer [Internet Protocol (IP), Internet Control Message Protocol (ICMP)], and transport layer [Transmission Control Protocol (TCP), User Datagram Protocol (UDP)] and various applications (HTTP, etc). Other protocols, like Bluetooth can be fuzzed as well.

At the CanSec West Conference, H.D. Moore was able to write a simple program to mangle Cascading Style Sheets (CSS), and was able to test it and find bugs in IE at the conference. He was able to find over a dozen ways to crash the browser. In total, he's found hundreds of ways to crash IE and other browsers. [9]

Types of Bugs Found

While the methods of finding software vulnerabilities varies greatly between the different fuzzing techniques, in the end, many of the same types of programming mistakes are found. The University of Wisconsin researchers found the same bugs over and over as described below: [1]

Pointer/Array Errors

Pointer/array errors are caused when an array is created with finite length, however, the program tries to access data it thinks is in the array but actually exists outside of the array and causes the program to read unknown, and possibly malicious data. If the data after the array can be manipulated, the program could read an execute this data, causing unauthorized access. Another problem is with null pointers, where the pointer of the array is null. To make it more interesting, different systems interpret the null pointer in different ways. Some will crash, while others will incorrectly process the data.

Not Checking Return Codes

When a function is called, the function will return a value. Proper coding techniques require a check of the return code, however it is easy to assume the return value is correct and process it accordingly. If the function does return bad data, and the program assumes it to be correct, the program could crash.

Input Functions

When a program inputs data, it should check the bounds of the input string. If the function inputs data beyond the boundaries of an array, undesirable data can be read.



Sub-Processes

A program will sometimes call another program and allow access to itself. If the input from this other program contains errors or anomalies, this could cause it to crash or hang.

Signed Characters

ASCII characters are 7-bit, and are read into an array of signed 8-bit integers. If the array is not declared as signed, the value may be read as a negative number. Then a hash value will be computed differently and the index to the hash table will be out of range.

Race Conditions

A program often looks for a control command, such as the keystroke "Control-C" to break. If a program is designed to perform some other operation, such as return certain values to their original state, the program these tasks once it receives the control key. If between the "Control-C" and the cleanup process, another control key, like "Control-\\" is received, the program will crash.

Undocumented Features

A network protocol follows a Request For Comment (RFC), which is a standard that the network protocol uses. If the protocol

supports new or undocumented features, certain data may trigger these features creating undesired results. Also, these features may contain bugs that may not have been thoroughly tested.

Fuzzing Utilities

Open Source

- ▶ **AxMan**—A web-based ActiveX fuzzing engine
- ▶ **Blackops SMTP Fuzzing Tool**—Supports a variety of different SMTP commands and Transport Layer Security (TLS)
- ▶ **Bluetooth Stack Smasher (BSS)**—L2CAP layer fuzzer, distributed under GPL license
- ▶ **COMRaider**—COMRaider is a tool designed to fuzz COM Object Interfaces.
- ▶ **Dfuz**—A generic fuzzer
- ▶ **File Fuzz**—A graphical, Windows based file format fuzzing tool. FileFuzz was designed to automate the creation of abnormal file formats and the execution of applications handling these files. FileFuzz also has built in debugging capabilities to detect exceptions resulting from the fuzzed file formats.

- ▶ **Fuzz**—The original fuzzer developed by Dr. Barton Miller at my Alma Matter, the University of Wisconsin-Madison in 1990. Go badgers!
- ▶ **fuzzball2**—TCP/IP fuzzer
- ▶ **radius fuzzer**—C-based RADIUS fuzzer written by Thomas Biege
- ▶ **ip6sic**—Protocol stressor for IPv6
- ▶ **Mangle**—A fuzzer for generating odd HTML tags, it will also auto launch a browser.
- ▶ **PROTOS Project**—Software to fuzz Wireless Application Protocol (WAP), HTTP, Lightweight Directory Access Protocol (LDAP), Simple Network Management Protocol (SNMP), Session Initiation Protocol (SIP), and Internet Security Association and Key Management Protocol (ISAKMP)
- ▶ **Scratch**—A protocol fuzzer
- ▶ **SMUDGE**—A fault-injector for many different types of protocols and is written in the python language.
- ▶ **SPIKE**—Network protocol fuzzer
- ▶ **SPIKEFile**—Another file format fuzzer for attacking ELF (Linux) binaries from iDefense. Based off of SPIKE listed above.
- ▶ **SPIKE Proxy**—Web application fuzzer
- ▶ **Tag Brute Forcer**—Awesome fuzzer from Drew Copley at eEye for attacking all of those custom ActiveX applications. Used to find a bunch of nasty IE bugs, including some really hard to reach heap overflows.

Commercial

- ▶ **beSTORM**—Performs a comprehensive analysis, exposing security holes in your products during development and after release.
- ▶ **Hydra**—Hydra takes network fuzzing and protocol testing to the next level by corrupting traffic intercepted “on the wire,” transparent to both the client and server under test.

Vulnerabilities Found by Fuzzing

(thanks to Ilja van Sprundel)

Protos

- ▶ OmniPCX Enterprise 5.0 Lx
- ▶ Cirpack Switches software version < 4.3c
- ▶ Cisco IP Phone Model 7940/7960 running SIP images prior to 4.2
- ▶ Cisco Routers running Cisco IOS 12.2T and 12.2 ‘X’ trains
- ▶ Cisco PIX Firewall running software versions with SIP support, beginning with version 5.2(1) and up to, but not including versions 6.2(2), 6.1(4), 6.0(4) and 5.2(9)
- ▶ Sipc (version 1.74)
- ▶ Ingate Firewall < 3.1.3
- ▶ Ingate SIParator < 3.1.3
- ▶ All versions of SIP Express Router up to 0.8.9
- ▶ Mediatrix VoIP Access Devices and Gateways firmware < SIPv2.4
- ▶ Succession Communication Server 2000 (- Compact)
- ▶ adtran ATLAS 550, ATLAS 800 (Plus), ATLAS 810Plus, ATLAS 890, DSU IV ESP, ESU 120e, Express 5110, Express 5200, Express 5210, Express 6100
- ▶ DSU IQ, IQ 710, 1st GEN, IQ Probe, TSU IQ, TSU IQ RM, TSU IQ Plus, NetVanta 3200, ADVISION, N-Form, T-Watch, OSU 300, Express 6503,
- ▶ Smart 16 Controller, TSU ESP
- ▶ AdventNet Web NMS 2.3
- ▶ ADVA AG Optical Networking: FSP 3000, FSP 2000, FSP II, FSP I, FSP 1000, FSP 500, CELL-ACE, CELLACE-PLUS, FSP Element Manager,
- ▶ FSP Network Manager, CELL-SCOPE
- ▶ iPlanet Directory Server, version 5.0 Beta and versions up to and including 4.13
- ▶ IBM SecureWay V3.2.1 running under Solaris and Windows 2000
- ▶ Lotus Domino R5 Servers (Enterprise, Application, and Mail), prior to 5.0.7a
- ▶ Critical Path LiveContent Directory, version 8A.3
- ▶ Critical Path InJoin Directory Server, versions 3.0, 3.1, and 4.0

- ▶ Teamware Office for Windows NT and Solaris, prior to version 5.3ed1
- ▶ Qualcomm Eudora WorldMail for Windows NT, version 2
- ▶ Microsoft Exchange 5.5 prior to Q303448 and Exchange 2000 prior to Q303450
- ▶ Network Associates PGP Keyserver 7.0, prior to Hotfix 2
- ▶ Oracle Internet Directory, versions 2.1.1.x and 3.0.1
- ▶ OpenLDAP, 1.x prior to 1.2.12 and 2.x prior to 2.0.8

Smudge

- ▶ subversion
- ▶ shoutcast
- ▶ Sambar webserver 0.6 overflow in POST handling
- ▶ Ratbox IRCD < 1.2.3 overflow in newline handling
- ▶ Unexploitable overflows in IE browser
- ▶ DoS in Helix Server < 9.0.2
- ▶ Remote Crashes in Bad Blue server
- ▶ Mailman bugs
- ▶ Cute overflow in mod_security

SPIKE

- ▶ smb stuff
- ▶ dtlogin arbitrary free()
- ▶ windows remote rdp DoS
- ▶ RealServer ../ stack overflow
- ▶ Verde
- ▶ Mdaemon
- ▶ Xeneo Web Server
- ▶ ipSwitch

Mangleme

- ▶ IE
- ▶ maxilla / Netscape / Firefox
- ▶ opera
- ▶ lynx
- ▶ links
- ▶ safari

Mangle

- ▶ libmagic (used file)
- ▶ preview (osX pdf viewer)
- ▶ xpdf (hang, not a crash ...)
- ▶ mach-o loading

- ▶ qnx elf loader
- ▶ FreeBSD elf loading
- ▶ openoffice
- ▶ amp
- ▶ osX image loading (.dmg)
- ▶ libbfd (used objdump)
- ▶ libtiff (used tiff2pdf)
- ▶ xine
- ▶ OpenBSD elf loading (3.7 on a sparc)
- ▶ unixware 713 elf loading
- ▶ DragonFlyBSD elf loading
- ▶ solaris 10 elf loading
- ▶ cistron-radiusd
- ▶ linux ext2fs (2.4.29) image loading
- ▶ linux reiserfs (2.4.29) image loading
- ▶ linux jfs (2.4.29) image loading
- ▶ linux xfs (2.4.29) image loading
- ▶ macromedia flash parsing
- ▶ Totem 0.99.15.1
- ▶ Gnumeric
- ▶ Quicktime
- ▶ Mplayer
- ▶ Python byte interpreter
- ▶ Realplayer (10.0.6.776)
- ▶ Dvips
- ▶ Php 5.1.1
- ▶ IE 6
- ▶ OS X WebKit (used safari)

ircfuzz

- ▶ BitchX (1.1-final)
- ▶ mIRC (6.16)
- ▶ xchat (2.4.1)
- ▶ kvirc (3.2.0)
- ▶ ircii (ircii-20040820)
- ▶ eggdrop (1.6.17)
- ▶ epic-4 (2.2)
- ▶ ninja (1.5.9pre12)
- ▶ emech (2.8.5.1)
- ▶ Virc (2.0 rc5)
- ▶ TurboIRC (6)
- ▶ leafchat (1.761)
- ▶ iRC (0.16)
- ▶ conversation (2.14)
- ▶ colloquy (2.0 (2D16))
- ▶ snak (5.0.2)
- ▶ Ircle (3.1.2)
- ▶ ircat (2.0.3)
- ▶ darkbot (7f3)
- ▶ bersirc (2.2.13)

- ▶ Scrollz (1.9.5)
- ▶ IM2
- ▶ pirc98
- ▶ trillian (3.1)
- ▶ microsoft comic chat (2.5)
- ▶ icechat (5.50)
- ▶ centericq (4.20.0)
- ▶ uirc (1.3)
- ▶ weechat (0.1.3)
- ▶ rhapsody (0.25b)
- ▶ kmyirc (0.2.9)
- ▶ bnirc (0.2.9)
- ▶ bobot++ (2.1.8)
- ▶ kwirc (0.1.0)
- ▶ nwirc (0.7.8)
- ▶ kopete (0.9.2)

isic

- ▶ Logging vulnerability in Checkpoint Firewall-1 4.0
- ▶ IP Stack vulnerability in Checkpoint Firewall-1 4.0
- ▶ Panic of Gauntlet 5.5 Beta
- ▶ Lock up Gauntlet 5.5 Beta
- ▶ Frag DOS of Gauntlet 5.5 Beta
- ▶ Lock up of Gauntlet 5.0
- ▶ Remote exploit of Raptor 6.x

Conclusion

Software auditing will always be an important processing in software assurance. While fuzzing is an easy and effective way to audit software, it is only one of many tools. Fuzzing should be implemented along with code reviews. The University of Wisconsin researchers opened up the door to this type of software auditing by creating a technique still used by many fuzzers. Also, their continued research into GUIs on specific OS showed that fuzzing can be used in many different areas. Now, network, file, and API fuzzing are important auditing techniques. Many of the vulnerabilities discovered every week are done so through fuzzing or use fuzzing to aid in the discovery. Fuzzing still has the ability to expand, improve, and analyze more complicated software. While fuzzing is not the only software auditing technique available to developers and security professionals, it is a technique that is here to stay. ■

References

1. Miller, Barton P., "An Empirical Study of the Reliability of Unix Utilities", December 1990.
2. Seltzer, Larry, "The Future of Security Gets Fuzzy", <http://www.eweek.com/article2/0,1759,1914332,00.asp>, Feb 6, 2006.
3. Van Sprundel, Ilja, "Fuzzing: Breaking software in an automated fashion", December 8, 2005.
4. Miller, Barton, David Koski, Ravi Murthy, et al. "Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services",
5. Forrester, Justin E., Barton P. Miller, "An Empirical Study of Robustness of Windows NT Applications Using Random Testing", July 27, 2000.
6. Miller, Barton, P., Gregory Cooksey, et al, "An Empirical Study of the Robustness of MacOS Applications Using Random Testing", July 20, 2006.
7. Juranic, Leon, "Using fuzzing to detect security vulnerabilities", April 25, 2006.
8. Sutton, Michael, Adam Green, "The Art of File Fuzzing", Blackhat.
9. Lemos, Robert, "Browser crashers warm to data fuzzing", April 13, 2006.

About the Author

Matt Warnock, CISSP, | is an Information Assurance Specialist with the Information Assurance Technology Analysis Center (IATAC). He graduated from Pennsylvania State University with a BS in Electrical Engineering, holds an Information Security Management Certificate from the University of Virginia, and is currently enrolled in a program for the MS degree in Telecommunications at George Mason University. His background includes assignments with the Defense Logistics Agency (DLA) in firewall and border-protection support. He may be reached at iatac@dtic.mil.