2009

Demystifying Fuzzers

Michael Eddington

Leviathan Security Group, Inc.

1/1/2009

Abstract

Fuzzing is an important part of the secure development lifecycle (SDL) and a popular tool for both defensive and offensive security researchers, consultants, and even software developers. With this popularity comes a plethora of fuzzers both open source and commercial. It seems like a new fuzzer pops up every other day. This makes it especially confusing for the corporate adopter who would like to reap benefits of fuzzing but has a high risk of adopting technology that is not supported, or likely to be rarely maintained after the creator has given his talk/written a book/posted to his blog and moved on.

This whitepaper and associated briefing takes a look at these different fuzzers and provides insights in to "if" and "what" they should be used for. Throughout this paper criteria for fuzzer adoption will be discussed and information about a selection of fuzzers provided, both open source and commercial. At the end we will examine several corporate use cases and select one or more fuzzer that best meets the perceived criteria. It is the hope of the author that this document will provide as a guide for your own adoption of fuzzers into your SDL or 3rd party adoption reviews.

Author Background

Perhaps the first question you might have is about me, the author of this paper. My name is Michael Eddington (mike@phed.org) and I'm the author of the widely used open source fuzzer Peach (http://peachfuzzer.com). I'm also a long time security researcher and consultant with over 10 years industry experience performing engagements from network penetration testing, to application security reviews, code reviews, design reviews, etc. Prior to my life as a security researcher I was a developer for many years.

Over the past 4 or 5 years I've developed and maintained Peach, seeing it through two major releases and working on the 3rd. For the past 2 years I've spent the majority of my time performing allot of fuzzing with both Peach and other fuzzers. In addition I've helped several companies adopt fuzzing into their security programs or supported the programs already in place. In fact it's this experience that lead me to consider the many challenges facing the corporate adopter.

So, as you might expect, I'm likely somewhat biased towards Peach. However, I will try and keep my bias in check for the duration of this whitepaper :)

Comments and feedback welcome, email to mik@phed.org.

Why are we fuzzing?

Fuzzing finds bugs. Period. That's a pretty good reason to fuzz, but it gets better, fuzzing is typically light on resources (or so we are told). Cheap bugs? Sounds pretty good!

So let's lay down a few goals for our own fuzzing efforts:

Find bugs This is the most obvious goal, if our fuzzing finds no bugs it's not very useful. Fuzzers are a good way to find bugs, but not so good about making assurances that no bugs exist. For the purposes of

this whitepaper we will assume all fuzzers discussed are equal at locating bugs. This is of course true fantasy, however this is out of scope.

Light on resources Our resource investment (expertise + time + money) should be on the low side when compared to other means of locating bugs and our return on investment (ROI). Note that I've added expertise into the mix. It tends to be that your high end smart resources are highly constrained. The more regular to junior talent can be utilized the better.

Types of Existing Fuzzers

Trying to classify fuzzers is a hard task, especially given how many fuzzers are generated simply as part of ongoing research. At practically every conference someone is releasing a new fuzzer that does things just a bit differently.

At a high level there are several groups we can toss most fuzzers into:

File Fuzzers As the name implies, fuzzers that target file formats only. They do not have the ability to speak any network protocol.

Network Fuzzers And these are fuzzers that target only network protocols. There are allot of these as the discovery of network based vulnerabilities has always attracted allot of attention.

General Fuzzers Following with our captain obvious theme, these fuzzers that can target a wide variety of targets, typically both file and network, and also others via custom I/O interfaces. For example: COM, shared libraries, RPC, etc.

Custom or One-off Fuzzers These are custom written fuzzers that target a specific format or network protocol. Typically these hand written, many times by testers. Custom fuzzers vary widely on how good their data mutation/generation is. For the purposes of this document we will not examine any custom or one-off fuzzers.

Open Source Fuzzer Types

File Fuzzers	Network Fuzzers	General Fuzzers	Custom/One-off Fuzzers
FileH/FileP	Sulley[1]	Peach	AxMan
FileFuzz	GPF	SPIKE[2]	DOM-Hanoi
	EFS	Fuzzled	hamachi
	TAOF	Fuzzware	mangleme
	Querub		

[1] Sulley is unable fuzz things other than network protocols out of the box, but as Pedram points out it can write files to disk and tools included with PAIMEI can be used to run the target application (PAIMEIfilefuzz).
[2] To be fair, SPIKE is more of a framework to create fuzzers than a fuzzer in its own right.

Commercial Fuzzer Types

File Fuzzers	Network Fuzzers	General Fuzzers	Custom/One-off Fuzzers
	Mu Security	Codenomicon	Protos
		beSTORM	

Comparison of Fuzzer Features

Fuzzer	Type	Data Model	State Model	Monitoring	Support	Coverage
Peach	General	X	X	X	X	-
Sulley	Network	X	X	X	-	X
SPIKE	General	X	-	-	-	-
Fuzzled	General	X	-	-	-	-
Fuzzware	General	X	-	X[3]	-	-
GPF	Network	X	-	-	-	-
EFS	Network	-	-	X	-	-
TOAF	Network	X[4]	-	X[5]	-	-
Querub	Network	X	-	-	-	-
FileH/FileP	File	-	N/A	-	-	-
FileFuzz	File	-	N/A		-	-
Mu Security	Network	?	?	X	X	?
Codenomicon	General	?	?	X	X	?
beSTORM	General	X	X	X	X	?

Fuzzing, the Process

Fuzzing can be broken up into the following basic phases. Depending on the fuzzer some will be automated or require pure manual attention.

Investigate This task is when you determine what you want to fuzz and generally how you will go about it. For example, you may identify a network protocol that is of high risk that you need to fuzz. You might select the portions of the protocol, and some of the use cases to test. This is usually a purely manual process and should be fairly short. For example, a quick chat with the developer to determine were interesting code could be, or reviewing threat models and other risk based documentation.

Modeling In this task we model the data and state of our target system. Commercial fuzzers may already include models for common protocols and file formats. Other fuzzers may not allow or need a definition. This task tends to be time and resource intensive when needed.

	Data Modeling	State Modeling	Includes Definitions	Method	GUI Tools	Generate Definitions

[3] Monitoring is limited and unknown fault triggers are mapped back to main engine
[4] TOAF does not provide true data modeling, however "fuzz points" can be created to identify interesting data. There is also no support for advanced concepts like size, count of offset relations.
[5] Monitoring limited to Windows, no symbols, and same machine

Peach	X	X	X[6]	XML	-	X[7]
Sulley	X	X	-	Python	-	-
SPIKE	X	-	-	C	-	-
Fuzzled	X	X		Perl/XML	-	-
Fuzzware	X	X	-	XML	X	X[8]
GPF	X	-	-	Custom	-	-
EFS	-	-	-	-	-	X[9]
TOAF	X[10]	-	-	GUI	X	-
Querub				Ruby	-	-
FileH/FileP	-	-	-	-	-	-
FileFuzz	-	-	-	-	-	-
Mu Security	?	?	X	?	X	-
Codenomicon	?	?	X	?	X	-
beSTORM	X	?	X	XML	X	-

Some fuzzers include tools to help speed this process up by analyzing a set of templates, static/dynamic analysis of the target application, or importing network captures. This can be a real time saver.

Analysis vs. Fuzzer[11]	**Peach**	**EFS**	**FuzzWare**
Feedback Engine	-	X	-
Wireshark PDML	X	-	X
Header Files	X	-	-
XML	X	-	-
ASN.1	X	-	-
String Token Analysis	X	-	-

Validate Now we need to verify our model is correct and the fuzzer can indeed talk to our system in a meaningful way. Many fuzzers provide a way to test their models w/o actually fuzzing the data to verify everything works. This is an essential step to make sure no small mistake causes the fuzzer to be useless. The use of a tool is not always required, but can speed things up. Otherwise you will need to spend some time with a network monitor and/or debugger/code coverage tool.

	Validation Tools	**Comment**
Peach	X	GUI validation tool
Sulley	X	Coverage
SPIKE	-	
Fuzzled	-	

[6] Peach definitions are available for purchase. Current offerings at time of writing are largely common file formats and some protocols.
[7] Peach includes a number of tools for generating fuzzing definitions, or assisting in that generation. Wireshark captures can be converted to partial data models, as can header files, and other structured data like XML or ASN.1.
[8] Can import Wireshark PDML exports
[9] Uses fitness testing and code coverage to feedback into debugger
[10] TAOF does not provide data modeling per say, but does allow marking "fuzz points."
[11] The commercial fuzzers may have some tools as well, but information was not available at time of writing.

Fuzzware	-	
GPF	-	
EFS	-	
TOAF	-	
Querub	-	
FileH/FileP	-	
FileFuzz	-	
Mu Security	?	
Codenomicon	?	
beSTORM	X	Coverage

Monitor Before we perform the actual run we also need to make sure we can monitor our target. Most advanced and full features fuzzers provide this as a feature and will also restart processes and continue fuzzing in the event of a crash. Should your fuzzer not provide this it will be a major slow down.

	Network Capture	Debugger	VM Control	Extensible	Process Restart
Peach	X	X	X	X	X
Sulley	X	X	X	X	X
SPIKE	-	-	-	-	-
Fuzzled	-		-	-	-
Fuzzware	-	X[12]	-	-	-
GPF			-	-	-
EFS			-	-	-
TOAF		X[13]	-	-	-
Querub	-	-	-	-	-
FileH/FileP	N/A	-	-	-	X
FileFuzz	N/A	X	-	-	X
Mu Security	X	X	-	?	?
Codenomicon	?	X	-	?	?
beSTORM	?	X	-	?	?

Run Now we get to run our fuzzer. This should be the fun part, were we sit back and wait for the bugs to trickle in. However, there are some things that will impact our leisure. The biggest impact is our fuzzers ability to restart the target when a crash occurs. One of the largest time sinks during the running cycle are pesky non-exploitable issues like null pointer exceptions. If the fuzzing stops every time you hit a fault things get very frustrating and means you cannot have unmanned runs being performed.

Another time sink that can occur during a run is just how long it will take to complete, or reach our minimum number of iterations (typically 500,000+). Another big benefit is if our fuzzer support a parallel mode were multiple targets can be fuzzed concurrently allowing us to throw additional machine resources at the problem.

[12] Limited to same machine
[13] Limited to same machine

Debugger Support	UNIX/OS X	Windows	Kernel	Symbols	Parallel Operation	Process Restart
Peach	VDB	WinDbg	Windows	Windows	X	X
Sulley	-	System Debugger	-	-	X	X
SPIKE	-	-	-	-	-	-
Fuzzled	-	?	-	-	-	-
Fuzzware	-	WinDbg			-	-
GPF	-	-	-	-	-	-
EFS	-	PAIMEI (System Debugger)	-	-	-	-
TOAF	-	System Debugger	-	-	-	-
Querub	-	-	-	-	-	-
FileH/FileP	-	-	-	-	X[14]	X
FileFuzz	-	System Debugger	-	-	X[15]	X
Mu Security	?	X	?	?	?	X
Codenomicon	?	X	?	?	?	?
beSTORM	?	X	?	?	?	?

Review Results After we run our fuzzer the last step is to review our findings. Some of the advanced fuzzers will even perform some amount of crash analysis for us, grouping similar or duplicate findings and even providing a guess as to whether the issue is exploitable. If your fuzzer does not provide these capabilities, then a large result set can be a huge time sink to review. It also tends to require a higher skilled person to perform the triage.

	Group Duplicates	Crash Analysis
Peach	X	Yes
Sulley	?	-
SPIKE	-	-
Fuzzled	-	-
Fuzzware	X[16]	-
GPF	-	-
EFS	-	-
TOAF	-	-
Querub	-	-
FileH/FileP	-	-
FileFuzz	-	-
Mu Security	?	?
Codenomicon	?	?

[14] Multiple instances can be run on different computers
[15] Multiple instances can be run on different computers
[16] Only by exception address and type

beSTORM	?	?

Adoption Risks

Selecting a fuzzer for use in a commercial situation means taking a good look at the risks involved in any technology adoption. Obviously there will be time invested in training people to use the tools, purchase of the tools (if applicable), and any IP generated for the tool (custom protocol or file specifications, etc).

Sustainability This is a key concern for technology adoptions and probably one of the hardest to quantify given how new fuzzing is to the commercial space. The commercial space is fairly small with only three main vendors who have all had products out for about the same amount of time. The open source arena is chuck full of fuzzers that were written, the talk given, and then abandoned or gone straight into maintenance only mode. Additionally, most open source fuzzers are small projects consisting of one or two developers. Should one of these developers lose interest or die the project may end. On the upside, with open source software the source code is available, though the license it's under may prove to be an issue.

There are some common questions to ask and information to gather as we try to determine how sustainable a project is:

- How many years has the project existed? We will count from first public release.
- When was last release made?
- Does the project have commercial backing?
- How many project leaders are there?
- Active community? Forums, mailing lists, etc.

	Current Version	Last Release Date	Years Available	Commercial
Peach	2.3	2009	5	No
Sulley	?[17]	2009	2	No
SPIKE	2.9	2004	7	No
Fuzzled	1.1	2007	2	No
Fuzzware	1.4	2008	1	No
GPF	4.6	2007	2	No
EFS	?	2007	2	No
TOAF	0.3.2	2007	2	No
Querub	?[18]	2009	<1	No
FileH/FileP	0.2	2006	3	No
FileFuzz	?	2006	3	No
Mu Security	?[19]	2005	4	Yes

[17] Sulley no longer has official releases and can only be retrieved by checking out the source code. It is unknown if another major version of Sulley will be produced.
[18] Querub does not have actual releases just a source control point
[19] No longer list version on web-site

Codenomicon	3.0	2001[20]	8	Yes
beSTORM	3.7	2004?[21]	5	Yes

Usability This might be called "maturity." For commercial products our rating will likely be high since the product is geared toward use by other companies. Additionally commercial products tend to have a dedicated testing team, UI designers, etc. Open source projects on the other hand typically start for use by the author and are opened up to others as a second thought. This can lead to poorly documented and hard to use systems.

As we review fuzzers we will want to take a good look at the following:

- How mature is project?
- How well documented is it?
- Is there an online support forum? Is the forum manned with project members who answer questions?
- Are there publications on the product (e.g. Books)?
- Are external users a priority for the fuzzer? This is a fairly subjective question for sure.

Training Availability of training can be critical for quick adoption and advanced usage of a any technology product. Obviously this will vary depending on the complexity of the product and also the resources staffed to use it. However, even if training is not viewed as needed currently the availability of training is a good indicator of a projects sustainability and focus on end users. Training offerings are not limited to commercial products, many well adopted open source projects provide training services, and in fact is a typical way of helping to fund continued development.

Support Another adoption risk is application support. Should issues arise, or bugs be found that are critical to project success, having access to commercial level support is key. Typically this is included with most commercial products, and many open source projects offer paid support cases or consulting as a way to help fund development.

License Restrictions Finally we need to be aware of any licensing restrictions. While GPL is very popular in the open source community, it can cause real problems for corporate adoption. A long term adoption will likely include custom modifications to the source code of the project (since it's available), and possibly even adopting to maintain and extend the project should it be abandoned. Additionally, given some of the many legal battles surrounding GPL many corporations now frown on using GPL solutions at all. My general feelings is non-restrictive licenses like BSD and MIT are preferable over GPL for corporate adoption. While GPL would require public release of any changes, BSD and MIT would not.

Fuzzer	Sustainability[22]	Usability	Training	Support	License

[20] Frist known as the PROTOS Test Suites.
[21] First product around 2004, was this there fuzzer though?
[22] For sustainability I largely used the last release, and during research how many releases there were. A 2 means I project is in maintenance for most part, 1 is abandoned.

					Restrictions
Peach	4	3/4	Yes	Yes	No (MIT)
Sulley	3	4	-	-	Yes (GPL)
SPIKE	1	1	-	-	Yes - GPL
Fuzzled	2	2	-		
Fuzzware	3	4	-	-	Light - Almost BSD
GPF	1	3	-	-	Yes - GPL
EFS	1	2	-	-	Yes - GPL
TOAF	2	3	-	-	Yes - GPL
Querub	2	2	-	-	Unknown
FileH/FileP	1	3	-	-	Yes - GPL
FileFuzz	1	3	-	-	Yes - GPL
Mu Security	4	5	X	X	Yes
Codenomicon	5	5	X	X	Yes
beSTORM	4	5	X	X	Yes

Upfront Costs

Another area that fits into our evaluation are the upfront costs associated with technology adoption. This is one area that open source jumps out in front with a big zero dollars. However we should take a good look at the cost for some of the commercial solutions as this could have a big impact on our selection.

I'll stop here quick and point out that my cost numbers are from Internet research and may not accurately represent the real cost of current products. During your own evaluation you will want to engage the sales rep from each company to get accurate pricing for each product. Additionally I recommend evaluating each product to make sure it can be used as expected and will properly fit into your process.

	Price	Restrictions/Time limits
Open source (Peach, Sulley,...)	$0	No restrictions
Codenomicon	$5,000 for 5 protocols, 5 days[23]	5 days
Mu Security	$50,000 for 10 protocols $250,000 for all protocols[24]	12 month license
beSTORM	$15,000 per module[25]	

Pulling it All Together

Now we are left with the task of pulling all of this information together and making sense of it. If you have been following along you will no doubt find that so far most of the commercial products have

[23] Codenomicon pricing from Routers news article April 20, 2009
[24] Mu Security pricing from SC Magazine, Aug 1, 2007
[25] beSTORM pricing from Software Mag.com, May 2, 2006

come out about the same except in pricing. On the open source side of things we can narrow things down pretty well to Peach and Sulley, though Sulley has largely gone into maintenance mode.

However, the true choice for which fuzzer to use is very much dependent on the adopter there needs.

If you have found this document interesting please join me for the Blackhat Briefing which will include updated information not available at time of writing along with additional details and comparisons of fuzzers as time permits. Stories about failures and successes available on request.

- Mike