

Codenomicon whitepaper:

## Parallel Execution of Defensics 3 Test Suites

Study of maximum throughput, resource consumption  
and bottlenecks for fast-speed fuzzing

- Riku Hietamäki, Ari Takanen and Anna-Maija Juuso - October 6th, 2010 -



- 1 Introduction
- 2 Background
- 3 The Defensics Testing Platform
- 4 Optimizing Test Execution Speed
- 5 Case Study
- 6 Results
- 7 Conclusions

CODENOMICON Ltd. | [info@codenomicon.com](mailto:info@codenomicon.com) | [www.codenomicon.com](http://www.codenomicon.com)

Tutkijantie 4E | FIN-90590 OULU | FINLAND | +358 424 7431  
10670 North Tantau Avenue | Cupertino, CA 95014 | UNITED STATES | +1 408 252 4000

## 1 Introduction

The topic of this whitepaper is high-speed robustness testing and fuzzing, and the use of fuzzing tools for load and performance testing. As a case study, we examine the performance of the Codenomicon Defensics 3 test tools through different performance metrics. As a software-based solution, Defensics is not restricted by the hardware constraints of any specific testing appliance. To see how scalable the fuzzing process is, we test how many parallel test suites the selected hardware platforms can execute simultaneously and how many test cases the suites combined can run per second, against single system under test, and single network interface. We looked at different test scenarios and changed the test setups to see how resource constraints and bottlenecks, such as test target behavior and the amount of CPU, memory and network connections available, affect the performance of the test configurations. The purpose of this whitepaper is to find the configuration for maximum throughput and to help testers to achieve better test performance, when using fuzzing tools in load, stress and denial of service testing.

## 2 Background

The systems and protocols used today are becoming increasingly complex. At the same time, product release cycles are getting faster creating a need for faster and more effective testing. Testers need to run more tests in much shorter time frames. To achieve this, testing platforms need to be able to run faster sequential tests and to run more test cases in parallel. Most load testing solutions are cumbersome and expensive appliance-based solutions. Hardware resources often restrain their performance, and they are difficult to use, with very little options for test cases editing.

In this whitepaper, we evaluate the suitability of the Defensics testing platform for high-speed robustness testing, or fuzzing. There are two factors that make Defensics a most promising solution for high-speed robustness testing. Firstly, the Defensics platform is software-based, and secondly, it runs each test tool and user interface as a separate process allowing parallel test execution setups. Thus, it can run multiple test suites simultaneously and its performance can be improved by simply increasing hardware resources.

Even though its features make Defensics a very suitable solution for high speed robustness testing, there have been no earlier studies on the performance of the Defensics platform when executing multiple parallel test suites. The purpose of this study is, therefore, to test the limits of the fuzzing platform with different hardware setups. High-speed robustness testing can also reveal problems that other testing techniques cannot find. Growing memory consumption, increases in CPU usage and longer processing times often indicate that there are bugs in the software. However, the signs can be so subtle that the bugs cannot be exposed with just a few test cases. For example, a bug might just cause a small increase in the memory or CPU consumption. But, when the target software is attacked with multiple identical test cases simultaneously, it can no longer handle all the requests at the same time. As a result, the free memory runs out or the CPU usage exceeds the critical limits, which leads to a DoS situation.

# 3

## The Defensics Testing Platform

### What is Fuzzing?

Defensics utilizes a particular type robustness testing, called fuzzing. Fuzzing tests protocol implementations in systems and applications with open interfaces. Protocols are the underlying language with which devices communicate. If a system has an open interface, anyone can send protocol inputs to it. For example, hackers can send protocol messages containing anomalies. If the system reacts to these messages in an abnormal manner, then there is a vulnerability in the software and the hackers just need to edit their feeds to get the system to behave in the manner they want. This is also how fuzzing works. Fuzzing is a form attack simulation. Its basic idea is that using hacking tools can proactively find and fix vulnerabilities, thereby making systems more robust, and resistant to attacks. The biggest advantage of fuzzing is that it can find previously unknown vulnerabilities.

### What is Defensics?

Defensics automates robustness testing and fuzzing. It utilizes protocol specifications and real traffic samples to create protocol models, which it then uses to generate test cases. The platform can be used to generate test cases containing both valid and invalid traffic, thus it can be used for all types of functional protocol testing, namely:

1. Robustness and fuzz testing
2. Load and performance
3. Features

Because Defensics utilizes protocol models to generate attacks, it can be used to test both client-side and server-side implementations. Fully model-based test suites, which are based on protocol specifications, are extremely flexible testing tools: message sequences and messages can easily be edited by the end-user. Defensics provides model-based test suites over 200 protocols. All other protocols can be tested with the Codenomicon Traffic Capture Fuzzer, which can be used to replay and test any communication protocols captured by commercial and open source network analyzers, or traffic captures loaded from third party vulnerability feeds or databases. The Codenomicon Defensics testing platform is used by hundreds of companies and organizations around the world for various purposes ranging from fast prototyping projects to more complex conformance testing solutions, which test complex specifications with hundreds of carefully built use cases.

### Defensics for High-Speed Robustness Testing

There are two factors that make Defensics a most promising solution for high-speed robustness testing. Firstly, the Defensics platform is software-based, and secondly, it runs each test tool and user interface as a separate process. Thus, it can run multiple test suites simultaneously and its performance can be improved by simply increasing hardware resources. A third powerful feature is that the Defensics platform can create multiple execution threads from one test generation process, enabling more efficient use of hardware resources.

#### *No Hardware Constraints*

Codemicon Defensics testing solutions are software-based and platform-independent. Thus they are not restricted by any hardware constraints making them fully scalable. On the one hand, the test suites only require nominal hardware. You can even run a test suite on a laptop or from low resource environment, such as a host in virtualized test laboratory. This is optimal for penetration testers and product security specialists, who need to move their test setups from one environment to another, and who cannot tie their tests to a specific hardware. On the other hand, the performance of the tools can be improved simply by employing more hardware resources. This way you can conduct more test cases in a shorter time frame.

#### *Multiple Test Tools and Targets*

The Defensics platform can also handle multiple test tools simultaneously, because it runs the GUI and all the test suites as separate processes. When using the command line, the GUI component is reduced to minimal operation. This slightly speeds up the test execution speed and reduces the memory fingerprint of the test suites. Finally, the test suites can be configured to run each test case multiple times before proceeding to the next test case, to improve their capability to find e.g. memory leaks and busy loops. Although, re-running test cases increases the test execution time, it actually improves test execution efficiency, because the test suites do not need to generate every test case before sending them out.

#### *Multiple Threads*

The Defensics testing platform can also execute test cases in multiple threads. When a test suite uses multiple threads, it uses one main thread to generate the test cases, and which are then executed in multiple parallel threads. Because the platform now only generates one protocol model and one set of test cases for all the threads executed, the test run times

are shorter. Multi-threaded execution uses less memory than running the same amount of tests as multiple parallel processes. Examining how multiple thread execution effects test run times is one the goals of our research.

## Conducting High-Speed Robustness Testing

The purpose of this study was to find out how many test suites can be executed simultaneously on an industry standard hardware platform, and to determine the maximum protocol transaction throughput speed. In practice, this means measuring how many test suites can be executed simultaneously, before the hardware resources or the resources of the operating system run out. By executing the same test setups using various hardware configurations, we can analyze how the amount of CPUs and memory available affect the performance of the testing platform.

## Fuzzing Performance Metrics

To test how suitable the Defensics fuzzing platform is for high-speed robustness testing, we test how many test cases it can generate and execute per second. As you can see in Figure 1, the load generated in performance tests is the result of two factors: the amount of sessions the test suite is running sequentially and in parallel.

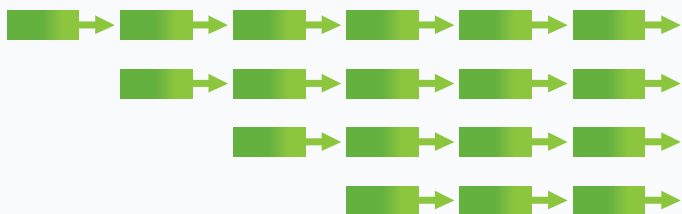


Figure 1: Performance testing with sequential and parallel sessions

The key metrics for fuzzing performance are:

- Amount of sequential test cases executed per second by a test suite
- Number of test suites running in parallel
- Amount of test cases executed per second by all parallel test suites

We will use “test cases per second” as our metric for fuzzing performance, as this is applicable to any protocol. Typical performance metrics such as “packets per second” or “bytes per second” are not as useful for fuzzing performance, because

they can be faked by just increasing the packet size. Also, the number of parallel sessions is also not a good metric for performance, because by increasing the timeouts generated by the fuzzer in each session can easily improve the results.

As Codenomicon Defensics is a software-based solution, its performance can be scaled both up and down by changing the hardware resources. In theory, the amount of test cases the Defensics platform can run grows in a linear correlation to the hardware resources available, until a bottleneck is found in the test system.

First measuring the execution time of one full test suite, or its subset, and then dividing the number of test cases executed with the time it took to run them provides the average speed of fuzzing performance. Especially with complex protocols, it is important to pick test groups that process the entire sequence and not choose test groups where the protocol sequence is aborted after first message.

## 4

## Optimizing Test Execution Speed

### Executing Test Plans Using Command-Line

In this case study, all the tests are executed from the command-line interface. Using the command-line makes it much easier to automate tests with scripts and to reproduce in different setups and configurations. To use the command line, all you need to do is save your test suite settings in a settings file or to save the entire test setup as a test plan. You can then run the tests using exactly the same test configuration in various hardware configurations. To open a test plan, just type in a command similar to the one below:

```
$ ./boot.sh --testplan ~/codenomicon/testplans/http-speed.testplan
```

You can modify or override the pre-configured settings in the test plan with additional parameters directly from command-line.

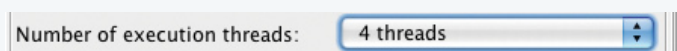
### Running Multiple Test Suites

In Defensics, using the command-line interface does not have much of an impact on the testing efficiency, because the GUI component of Defensics is already a separate process from the actual test suites. The best benefit from command-line use is the capability to script several suites to run in parallel. Scripting is needed to start new test tools automatically after

a certain execution time, to monitor system resources for possible over consumption and to automatically log the test case throughputs and the statistics for the tested system and the test suites for later examination. In our test setup, a simple script starts five parallel test suites, one after the other. Each test suite runs the test cases for five minutes, before five more test suites are started. This script runs until some of the critical system limits are triggered.

## Running Multi-Thread Processes

Multi-Threaded operation can be configured from the GUI with the option named Number of Execution Threads, as shown below:



The same option can be used from command-line with the "threads" option:

```
$ ./boot.sh --testplan ~/codenomicon/testplans/
http-speed.testplan --threads 4
```

Note that multi-threaded versions of test suites might not be immediately available for all protocols.

## Repeat Each Test Case

Repeating each test case for a number of times before proceeding to the next test case can slightly increase the test case throughput and it is often also a specific requirement for passing a robustness test. In Defensics, this is configured with the Repeat Test Case option from the GUI. From command-line, this is achieved as follows:

```
$ ./boot.sh --testplan ~/codenomicon/testplans/
http-speed.testplan --repeat 100
```

Test case repetition enables you to catch some subtle bugs that are not triggered by running test case only once. Test case repetition strengthens the effect of parallel test case execution, as parallel test generation threads execute similar test cases.

## Theoretical Speed by Running Tests with No Target

To achieve maximum theoretical test run speed, the test cases are run without a test target. The Defensics test suites come with a hidden parameter for running the test suites in

this mode. To use this operation, simply add the parameter "--no-inject" at the end of the executed command:

```
$ ./boot.sh --testplan ~/codenomicon/testplans/
http-speed.testplan --no-inject
```

This operation is mainly used by Codenomicon internally to debug bugs within the test suite itself, and to measure test generation efficiency for optimization purposes.

# 5 Case Study

## Choosing the protocols

The used protocol models impact the speed and performance of the tests. The HTTP and SSL/TLS server-side test suites were chosen for closer study, mainly because it was easy to setup efficient real-life test targets for them. HTTP represents a simple stateless protocol, whereas TLS is an extremely complex dynamic multi-stage protocol. Both protocols use TCP, which is probably the most resource-constrained protocol of the entire operating systems.

Additional tests were performed on DNS, because, as for HTTP and TLS, there is a wide selection of fast speed open source test targets for DNS, but, unlike the other two tested protocols, DNS uses UDP instead of TCP as the transport protocol.

All the above-mentioned protocols are tested over IPv4, but some tests are also conducted with IPv6 itself, mainly out of interest for this emerging protocol. For years, probably since 2004, Codenomicon has had full support for IPv6 fuzzing, and all Defensics application fuzzers also run on top of IPv6, but IPv6 still has not managed to overtake IPv4. The main reason for this is its complexity.

## Test Targets

A TLS server based on the GnuTLS open source project ([www.gnu.org/software/gnutls/](http://www.gnu.org/software/gnutls/)) was used as the target for the SSL/TLS server test tool, and Apache ([www.apache.org](http://www.apache.org)) was used as the target for the HTTP server test tool. Both test suites were configured to handle hundreds of parallel child processes and threads. The BIND9 DNS server ([www.bind9.net](http://www.bind9.net)) was used as the test target for DNS and the Standard Linux Kernel was used for IPv6. For IPv6 tests, the application layer was turned into listening mode using a simple server, which was created in the Python scripting language. The code for creating the server is shown below in Figure 2. Neither BIND

for DNS nor the Python server for IPv6 was optimized for load or performance, so in these setups the speed of the SUT was probably the bottleneck.

```

1 import socket, traceback
2
3 host = '' # Bind to all interfaces
4 port = 161
5
6 s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
7 s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
8 s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
9 s.bind((host, port))
10
11 while 1:
12     try:
13         message, address = s.recvfrom(8192)
14         print "Got data from", address
15         # Acknowledge it.
16         s.sendto("I am here", address)
17     except (KeyboardInterrupt, SystemExit):
18         raise
19     except:
20         traceback.print_exc()

```

Figure 2: Simple Python Server

### Test Setup

We built two test setups using a range of high-end servers, see Figure 3, below. In the first setup, we have one single server (Test Setup 1), on which both the test tools and target software (SUT) are running. In the second setup, there are two servers (Test Setup 2). The test tools are running on one server and the SUT on the other one.

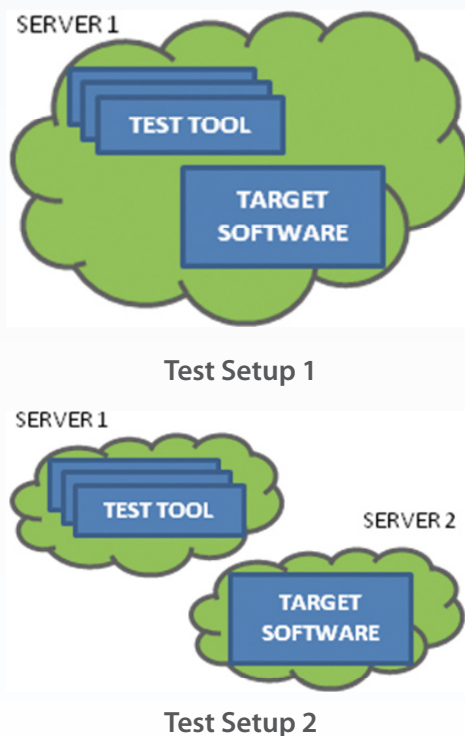


Figure 3: Test Setup 1 and 2

Most of the tests are executed using the first test setup, because it is easier to configure and does not require any external routing of the IP packets between the two systems. The benefit of this test setup is that there is no need to route the packets through the network to the other, thus the network speed will not cause any bottlenecks. The disadvantage of this setup is that both the target software and the test tools consume the same operating system resources, memory and CPU time. Thus, the amount of network connections the operation system can create is likely to become a major bottleneck when running parallel sessions.

To investigate how many test suites Defensics can run simultaneously, we also need the second test setup with two servers, in which the amount of network connections the operating system can open simultaneously is not an equally restrictive factor as in the first setup. Also in this setup, the SUT resources do not impact the speed of test generation.

### Hardware Configurations

To reveal the bottlenecks of the various test setups, we conduct the tests on a selection of high-end hardware configurations. The servers consist of different amounts of processors, which, in turn, contain various amounts of cores, but, all the servers are equipped with extensive processing power and large amounts of memory.

Our first hardware configuration consists of two identical servers (Servers A and B) with the following technical specifications:

#### Hardware Configuration 1:

##### Server A

- 96 GB of RAM memory
- 16 CPU cores, formed by 4 Quad-Core
- AMD Opterons
- Centos Linux with Java 1.6 platform

##### Server B

- 96 GB of RAM memory
- 16 CPU cores, formed by 4 Quad-Core AMD Opterons
- Centos Linux with Java 1.6 platform

Our second hardware configuration consists of two Dell PowerEdge server platforms, Servers C and D, which have the following technical specifications:

## Hardware Configuration 2:

### Server C

- 64 GB of RAM memory
- 24 cores, 4 CPUs, Intel(R) Xeon(R) CPU X5670 @ 2.93GHz, 12288 KB cache
- Centos Linux with Java 1.6 platform

### Server D

- 128 GB of RAM memory
- 32 cores, 4 CPUs, Intel(R) Xeon(R) CPU X7550 @ 2.00GHz, 18432 KB cache
- Fedora Linux with Java 1.6 platform

Some reference tests are also conducted on a low-end laptop with a dual-core Intel CPU and 2 GB of memory, running MacOS (Snow Leopard).

## Test Scenarios

We start the testing by looking at the following test scenarios in Test Setup 1, where both the test suite and SUT are located on the same server:

### Test Scenario 1:

Valid test case is looped infinitely, all logging disabled and instrumentation enabled.

### Test Scenario 2:

Anomalized test cases are looped infinitely, all logging and instrumentation disabled.

### Test Scenario 3:

Anomalized test cases are looped infinitely, valid and failed test cases were logged and instrumentation was enabled. (Note: This is most typical setup)

### Test Scenario 4:

Anomalized test cases are looped infinitely, full logging and instrumentation was enabled.

## Test Scenario 1

In the first test scenario (Test Scenario 1), only valid test cases are run. The primary purpose is to see how fast valid test cases can be run with minimum logging compared to fuzzing performance with similar optimized setup. The secondary purpose of these tests is to see how fast the test cases could potentially be run using valid message sequences, as this type of

approach is typically used for performance and load testing.

The used valid test sequence can be a conformance test or an interoperability test that is built into the test generator. Most Defensics fuzzers come with a handful of built-in valid cases from which you can choose. Some test suites contain more than a hundred valid test scenarios. The instrumentation technique used in Test Scenario 1 consists of sending a valid protocol message sequence to the SUT. Using instrumentation should not impact the test case execution a lot, because the message sequences are typically the same for the test case itself and for the instrumentation sequence.

## Test Scenario 2

In the second test scenario (Test Scenario 2), the test cases contain anomalies, i.e fuzzing, and every test case is unique. The purpose is to see how fast test cases containing anomalies can be generated and run with minimum logging. The anomalized test cases can affect the test target and the test run times in many ways. The anomalies can obstruct the operation of the test target and slow down the test run. Sometimes, the anomalized test cases are faster than valid test cases: unlike valid test cases they do not necessarily need to wait for replies, because there might not be any. Complex test sequences often abort before reaching the end of a sequence. This can already happen immediately after the first message, e.g. when fuzzing the first message of a complex protocol like TLS.

## Test Scenarios 3 and 4

In the third and fourth test scenarios (Test Scenario 3 and 4), varying levels of logging is enabled. In test scenario three, only the valid and failed test cases are logged, whereas, in test scenario four, all test cases are logged. The purpose of these test scenarios is to see how logging levels effect test run times. Full logging can quickly generate gigabytes of logs, and when increasing amount of test suites are writing into their log files simultaneously this is bound to affect the test execution speed.

## Test Scenarios 5 and 6

In addition to the above scenarios, two additional test scenarios are also tested with both Test Setups 1 (one server) and 2 (two servers). The aim of these tests is to see whether running the SUT in a separate host could further increase the testing speed. In Test Scenarios 5 and 6, the tool settings are optimized to achieve the fastest possible test runs. This mostly consists of reducing most timeouts to zero, and disabling all

instrumentation and logging. The purpose of these scenarios is not to look at hard drive consumption, but to achieve maximum throughput. In the following two test scenarios, the test tool settings are optimized in order to achieve the best possible throughput:

*Test Scenario 5:*

Valid test case was looped infinitely, all logging and instrumentation is disabled, test sequence timeouts optimized.

*Test Scenario 6:*

Anomalized test cases were looped infinitely, all logging and instrumentation is disabled, test sequence timeouts optimized.

## 6 Results

This study had two parallel goals. The first goal was to find out how many suites can be executed simultaneously with different hardware resources. The other goal was to see how many test cases, simple requests or more complex message sequences, the Defensics test tools can feed within a second. This section highlights the most significant results collected from most interesting test setups and test scenarios.

The results also show the peak performance point for selected scenarios. After this point, starting more test suites no longer increased the average test run speed, and in most cases the performance actually starts to decrease.

### Theoretical Maximum Speeds

We measured the theoretical speeds for the HTTP and TLS Server test tools using a dual-core MacBook Air with 2 GB of memory. The theoretical speeds were as follows:

- HTTP Server: 1000+ test cases per second per test suite
- TLS Server: 200+ test cases per second per test suite

On the faster servers, the theoretical speed per test suite did not vary significantly with the number of test suites running, but the speed increased linearly as more and more test suites were launched in parallel. This indicates that the speed of the system under test (SUT) and the operating system's network resources are probably the first bottlenecks.

The speed of the real test runs does not reach the theoretical speed. The speed of real test runs, when running single HTTP

test suite against live test targets, ranged from 200 to 650 test cases per second depending on the test configuration. But as with the theoretical speed, the increase in test speed was linear when more test suites were executed in parallel.

### Fuzzing Performance Results for HTTP

A total of 170 HTTP server test tools were started in Test Scenario 1 (valid sequence) with Test Setup 1 (one server) and Hardware Configuration 1 (16 cores), before the available 96 GB of RAM was consumed. In total, 340 test generation processes and 235 multi-threaded Apache processes were running at the same time. Each test tool process consumed an average of 600 MB of memory.

The best average test case speed for servers A and B was 4700 requests per second and this was achieved using ten parallel multi-threaded HTTP server test suites. Servers C and D were able to reach 16,000 test cases per second with 40 parallel suites, each running in 5-threaded mode. This results in approximately 200 parallel HTTP sessions.

### Fuzzing Performance Results for TLS

The SSL/TLS protocol is much more complicated than the HTTP protocol. Firstly, the test cases in the SSL/TLS test suite consist of longer sequences. These sequences contain complex multi-state handshakes, in which the parties authenticate each other and exchange the required parameters and identifiers. Thus, the SSL/TLS test cases consume slightly more memory than the HTTP test cases. In addition, due to complexity of the sequence, each SSL/TLS test suite cannot execute as many test cases per second as the HTTP test suite.

In total of 125 SSL/TLS server test tools were started in Test Scenario 1 (valid sequence) with Test Setup 1 (one server) and Hardware Configuration 1 (16 cores), before all the 96 GB of RAM available was consumed. Testing with servers A and B, the best average for the SSL/TLS tool was 145 test cases per second with 35 running SSL/TLS tools. This low figure is probably due to a badly configured SUT, as servers C and D were able to reach 2400 test cases per second with 40 parallel test suites each running in 5-threaded mode. This results in 200 parallel TLS sessions.

### Effect of CPUs and Logging in Single-Threaded Mode

In single threaded mode, the number of CPUs had the greatest effect on the performance of the testing platform.

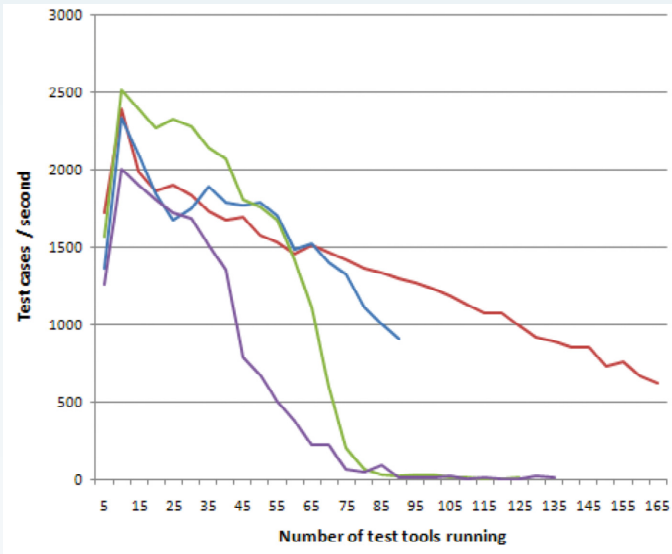


Figure 4: Test Scenarios 1-4 with 16-core Server A and B.

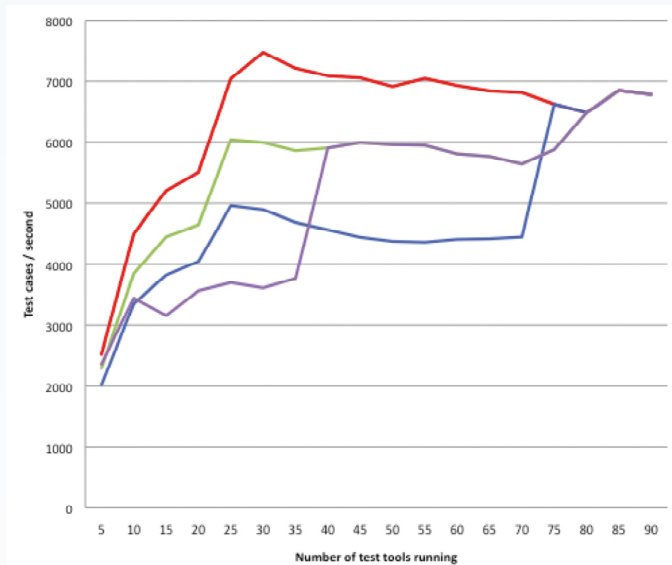


Figure 5: Test Scenarios 1-4 with 24-core Server C.

The Figures 4 to 6 illustrate the results from test scenarios 1-4 in Test Setup 1, when running the HTTP fuzzers in single thread mode on a 16-, 24- and a 32- core server. As the number of CPUs grew, the best performance of the test platform improves. The 16-core Server A and B reach best performance with 15 parallel test suites, the 24-core Server C with 30 test suites and the 32-core Server D with 40 parallel test suites. Each test case generator seemed to consume approximately 60-70% of one single CPU core. These results seem to be in line with the measurements made with single test suites.

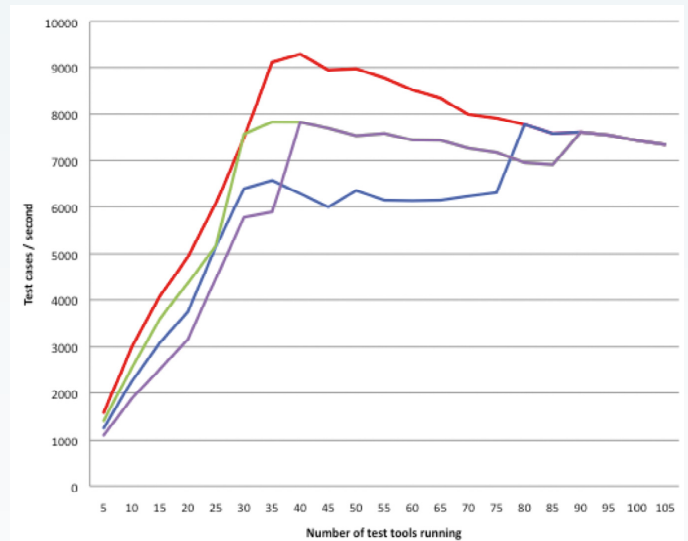


Figure 6: Test Scenarios 1-4 with 32-core Server D.

In all these tests, the test performance was from 10-20% better with no logging, indicating that the file system resources and speed are also a minor bottleneck.

### Single-threaded vs. Multi-threaded

Figure 7 compares the test execution speed when running test cases in 1-threaded mode to running them in 5-threaded mode. In 5-threaded mode, the same test performance can be reached with less parallel processes, thus requiring one fifth of the memory compared to running same amount of parallel sessions in single threaded test suites. With 10 test tools running in 5-threaded mode, the same peak test performance of 10.000 test cases per second is reached with only around 16 cores and 6 GB of memory in use, compared to running 35 test tools in 1-threaded mode requiring approximately 20 cores and 20 GB of memory.

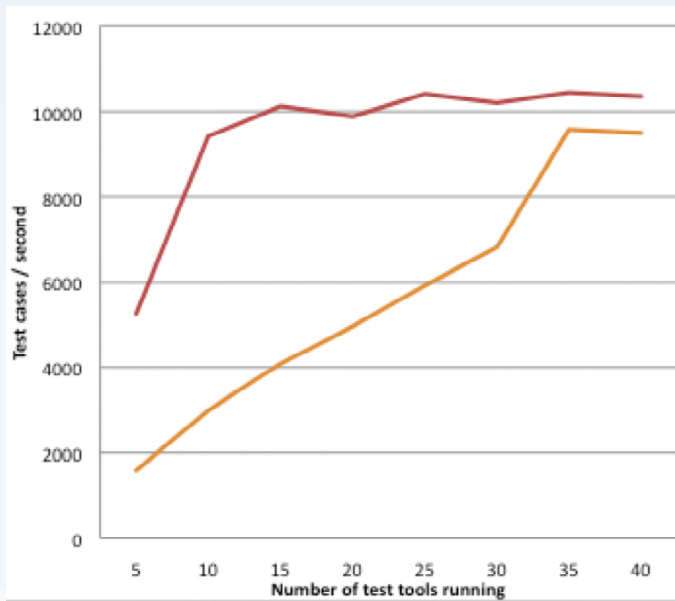
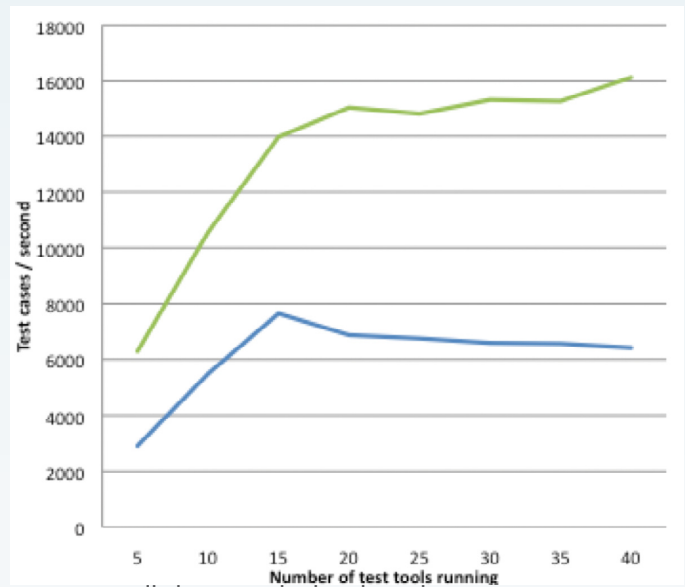


Figure 7: Test Scenario 5 in single-threaded and five-threaded modes.

In theory, the throughput of a single-threaded single-process HTTP server test tool is about 1000 test cases/ second on a modern PC. In one setup, running 15 parallel HTTP server test suites in 5-threaded mode, we were able to run 14.000 test cases per second (tc/s), peaking at 16,000 tc/s with 40 suites in 5-threaded mode. Other speed records included 2400 tc/s with TLS message sequences, 41.000 tc/s with DNS and 4500 tc/s with IPv6. In all the speed records test scenarios, the first bottleneck appeared to be the amount of CPU available. From our tests, we found that each Defensics test case generator seems to consume approximately 60-70% of one CPU core. The most significant benefit of multi-threaded fuzzing is that the execution threads can use the same protocol model, and thus consume less memory. On the other hand, multi-threaded operations can create some CPU overhead, because they share the same process resources.

### Differences between Test Setups 1 and 2

The maximum throughput with optimized default tool configurations was achieved when testing the HTTP server test tool with two servers. The average test case throughput was about two times higher in Test Setup 2 compared to Test Setup 1. This was most likely caused by the ability of the underlying operating system to handle more network connections in Test Setup 2, since the test tools and the target software were no longer consuming common system resources. Network speed did not have any significant impact on the fuzzing performance, although in some configurations the NIC through-



put eventually became the bottleneck.

Figure 8: Test Scenario 6 in Test Setups 1 and 2.

Figure 8 shows that resource constraints of the operating system can be avoided by running tests against another machine instead of running the software on the same machine. (IMAGE)

## 7 Conclusions

As a software based solution, Codenomicon Defensics is a flexible testing platform. It can run multiple test tools in parallel. However, to be able to do so the system requires a lot of memory and a lot of processing power. If there is enough processing power available, the increase in the test performance is linear until some other performance bottleneck is reached. After this, increasing the number of test suites no longer increases the test efficiency. To find the optimal test suite settings and hardware configurations, we looked at potential bottlenecks in a testing environment by testing six different test scenarios in two different test setups. The potential bottlenecks we identified are: the amount CPU cores (or processing power), memory and network connections available and SUT performance.

The biggest challenge in most test scenarios was either the performance limits of the SUT or the operating system's network resources. The latter was a problem with high performance test suites like the HTTP tool and the Traffic Capture Fuzzer. One method to avoid this problem is to either use

several network interface cards or to setup a virtual environment, where each tool is running on a separate virtual host. The problem can also be overcome by using blade-server setups containing, e.g., a multitude of 8-core CPU blades, each with its own operating system generating the load. In this type of setup, the average test generation efficiency can be substantially raised from the values we achieved in our tests. In this configuration the blade-server (or other type of cluster-like setup) should always be able to send more network packets compared to running the same amount of test suites from one operating system.

In comparison to the network resources, the hard drive space or its speed are not as critical for modern systems. Codenomicon Defensics creates optimized logs, which can be even further optimized for higher output performance. In most of the tests, the difference between test execution performance with no logging and with full logging was only 20 percent up until all CPU processing power was used. However, eventually with hundreds of simultaneous test suites, logging can eventually also create a bottleneck.

Performance testing is a big step away from fuzzing, but our tests showed that excellent results can be achieved by combining software based fuzzing tools with modern off-the-shelf hardware. Running multiple multi-threaded test suites on multiple servers and using minimal logging can achieve the best test execution performance. Finding the optimal configuration depends on the protocol that is being tested. It is also important to configure the SUT accurately. However, in addition to attempting speed records, it is important to keep in mind what you really want to test, as for example, logging might be a minor bottleneck, but on the other hand it significantly facilitates fault detection and regression testing. It should also be noted that the cumulative effect of all the parallel test cases can also create false positives, even though normally there are no false positives in fuzzing. This just means that the findings need to be verified more thoroughly.