SAND2016-3765C

Exceptional service in the national interest

Sandia National Laboratories

Application Performance and Security Testing

NLIT May 1-4, 2016

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Agenda

- Overview of Testing Services
- Load/Stress Testing
- Application Performance Troubleshooting
- Application Security
- Discussion

Why offer performance and Security testing services?

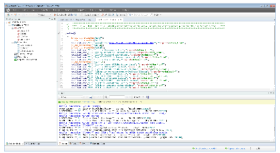
Performance and security testing services use expensive tools and require specialized training. It is more cost efficient to train a few staff and offer testing as a service

- Application Performance – Load/Stress testing
  - Objective: Verify performance for expected production load and growth
- Transaction Analysis Baselining and Troubleshooting
  - Objective: Troubleshooting of transaction issues for multi-tiered applications
- Application Security
  - Objective: Identify potential security weaknesses in applications prior to production deployment

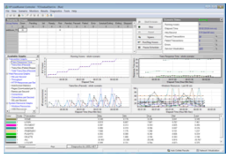
Load/Stress Testing

- Service
  - Load – Can system handle anticipated real-world load?
    - Customer requirements to model
      - How often is high volume business process executed in 1 hour
      - How long does it take to execute the high volume business process
      - What is the maximum allowed transaction response time?
  - Stress –How much more load can the system handle before performance and functionality is compromised?
  - Protocols: Web applications, web services, and mobile web
- Limitations
  - The tool doesn't automatically identify the root cause of issues
  - Risk: Only high volume business processes are identified

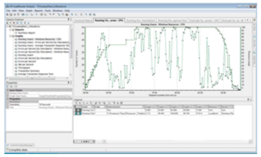
Design & Script



Execute Test



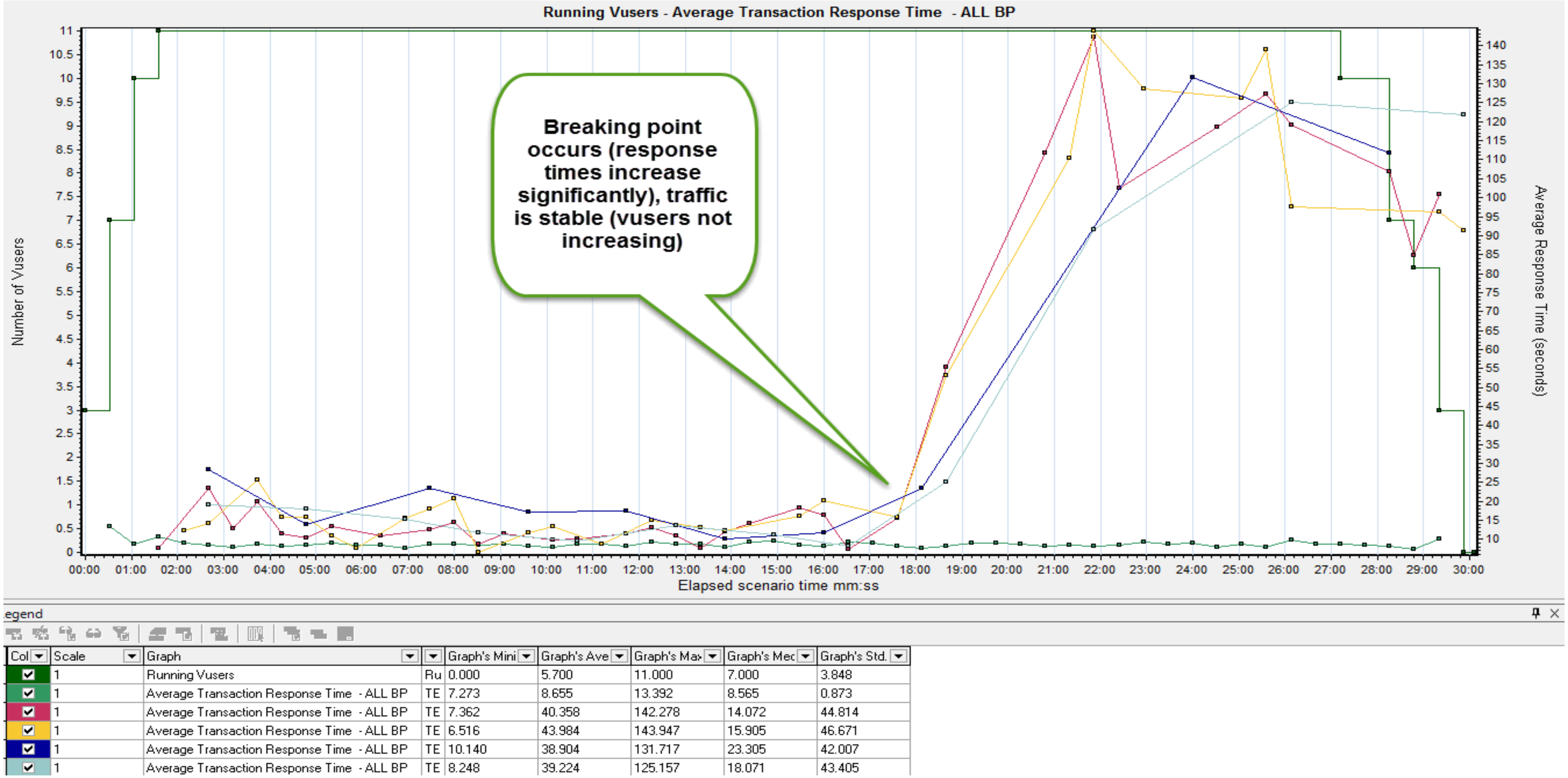
Analyze results & report



Tool: HP LoadRunner

Load/Stress Testing Analysis

- Example of application performance hitting a breaking point
  - Traffic is at full load and response times are steady, then at a certain time the application hits a breaking point, response times spike to large amount

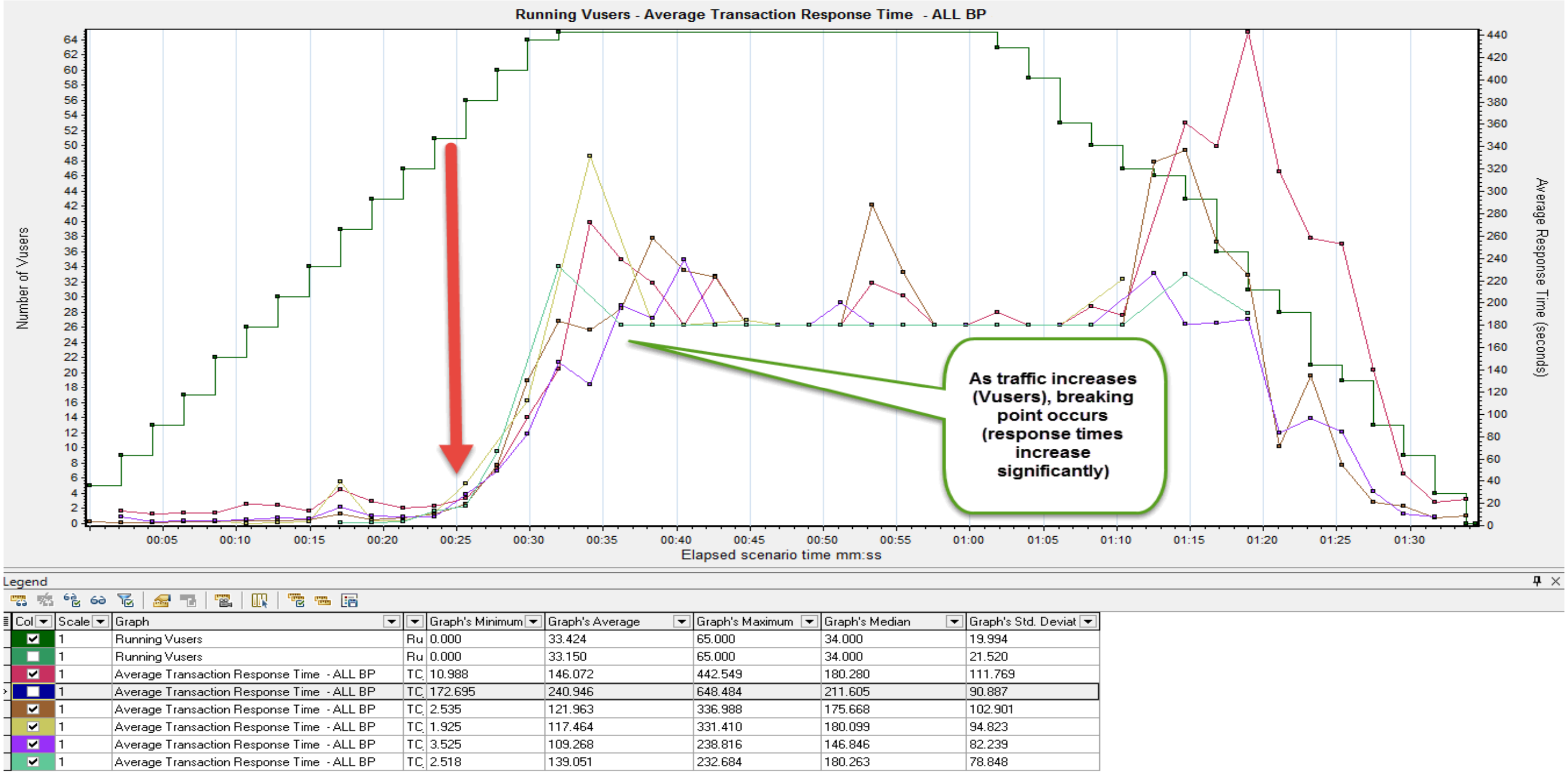


Breaking point occurs (response times increase significantly), traffic is stable (vusers not increasing)

Col	Scale	Graph		Graph's Mini	Graph's Ave	Graph's Ma	Graph's Mec	Graph's Std.
🟩 ☑	1	Running Vusers	Ru	0.000	5.700	11.000	7.000	3.848
🟩 ☑	1	Average Transaction Response Time - ALL BP	TE	7.273	8.655	13.392	8.565	0.873
🟥 ☑	1	Average Transaction Response Time - ALL BP	TE	7.362	40.358	142.278	14.072	44.814
🟨 ☑	1	Average Transaction Response Time - ALL BP	TE	6.516	43.984	143.947	15.905	46.671
🟦 ☑	1	Average Transaction Response Time - ALL BP	TE	10.140	38.904	131.717	23.305	42.007
🟦 ☑	1	Average Transaction Response Time - ALL BP	TE	8.248	39.224	125.157	18.071	43.405

Load/Stress Testing Analysis Cont…

- Another example of application performance hitting a breaking point
  - Traffic is increasing (vusers ramping up: green steps)
  - Application hits breaking point (~50 vusers: red arrow) – Response times increase significantly

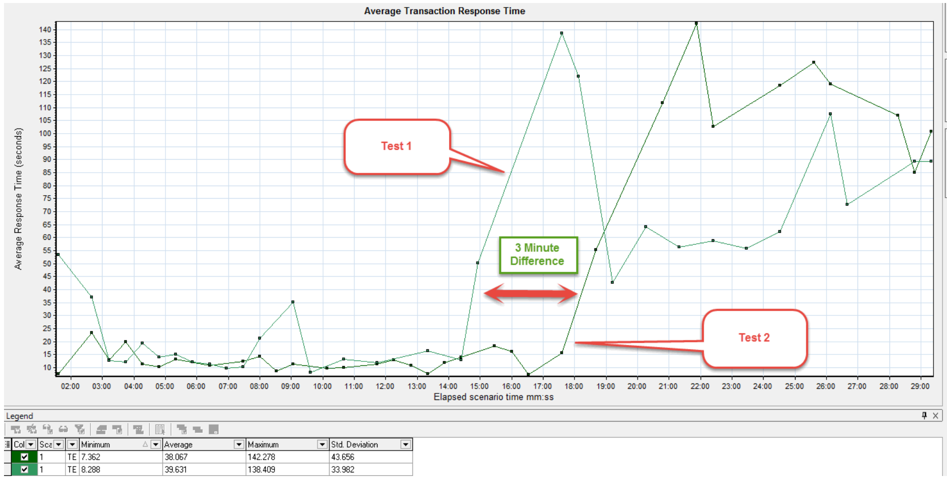


As traffic increases (Vusers), breaking point occurs (response times increase significantly)

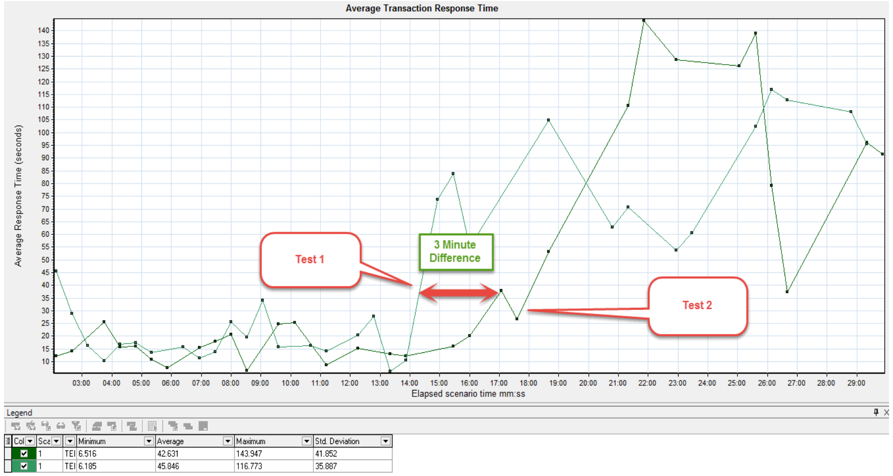
Col	Scale	Graph		Graph's Minimum	Graph's Average	Graph's Maximum	Graph's Median	Graph's Std. Deviat
🟩 ☑	1	Running Vusers	Ru	0.000	33.424	65.000	34.000	19.994
🟩 □	1	Running Vusers	Ru	0.000	33.150	65.000	34.000	21.520
🟥 ☑	1	Average Transaction Response Time - ALL BP	TC	10.988	146.072	442.549	180.280	111.769
🟦 □	1	Average Transaction Response Time - ALL BP	TC	172.695	240.946	648.484	211.605	90.887
🟫 ☑	1	Average Transaction Response Time - ALL BP	TC	2.535	121.963	336.988	175.668	102.901
🟨 ☑	1	Average Transaction Response Time - ALL BP	TC	1.925	117.464	331.410	180.099	94.823
🟪 ☑	1	Average Transaction Response Time - ALL BP	TC	3.525	109.268	238.816	146.846	82.239
🟩 ☑	1	Average Transaction Response Time - ALL BP	TC	2.518	139.051	232.684	180.263	78.848

Load/Stress Testing Analysis Cont…

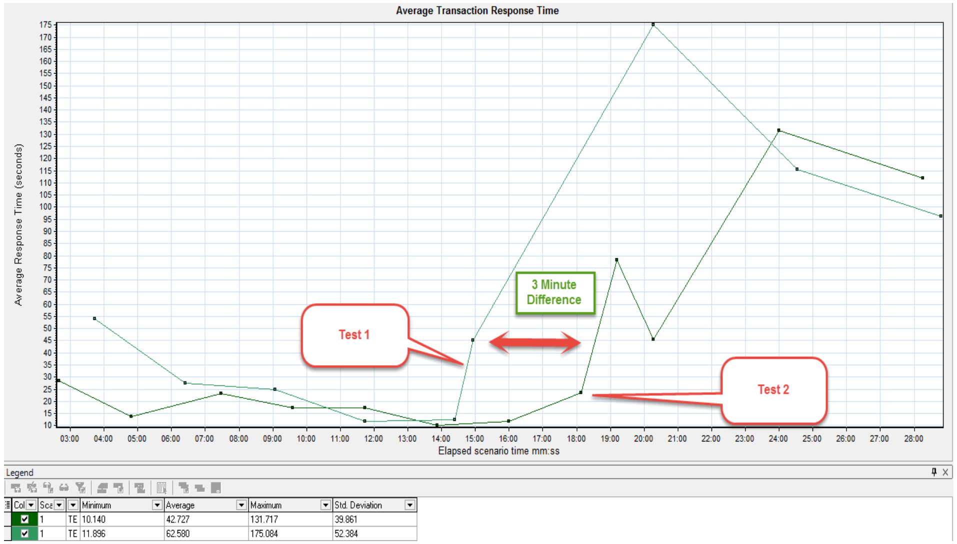
- Example: Adding more memory only delays the problem



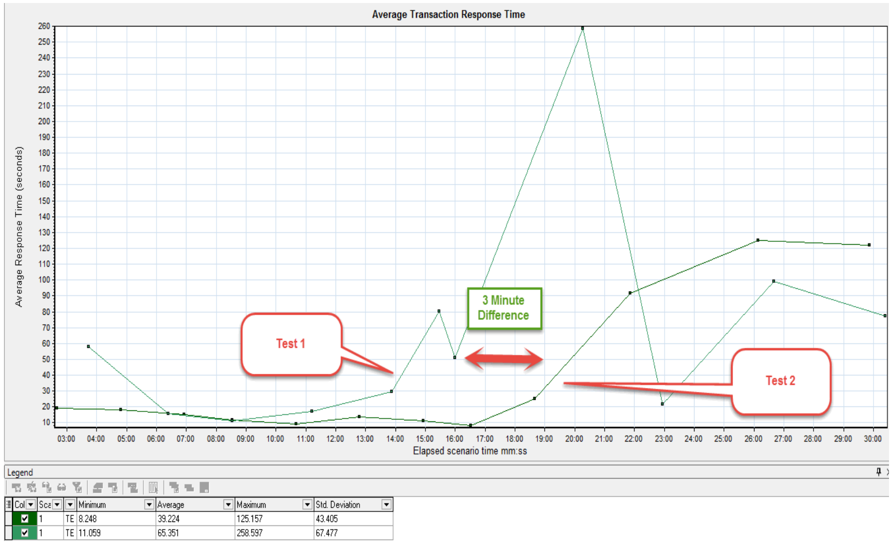
Col	Sca		Minimum	Average	Maximum	Std. Deviation
☑	1	TE	7.362	38.067	142.278	43.656
☑	1	TE	8.288	39.631	138.409	33.982



Col	Sca		Minimum	Average	Maximum	Std. Deviation
☑	1	TEI	6.516	42.631	143.947	41.852
☑	1	TEI	6.185	45.846	116.773	35.887



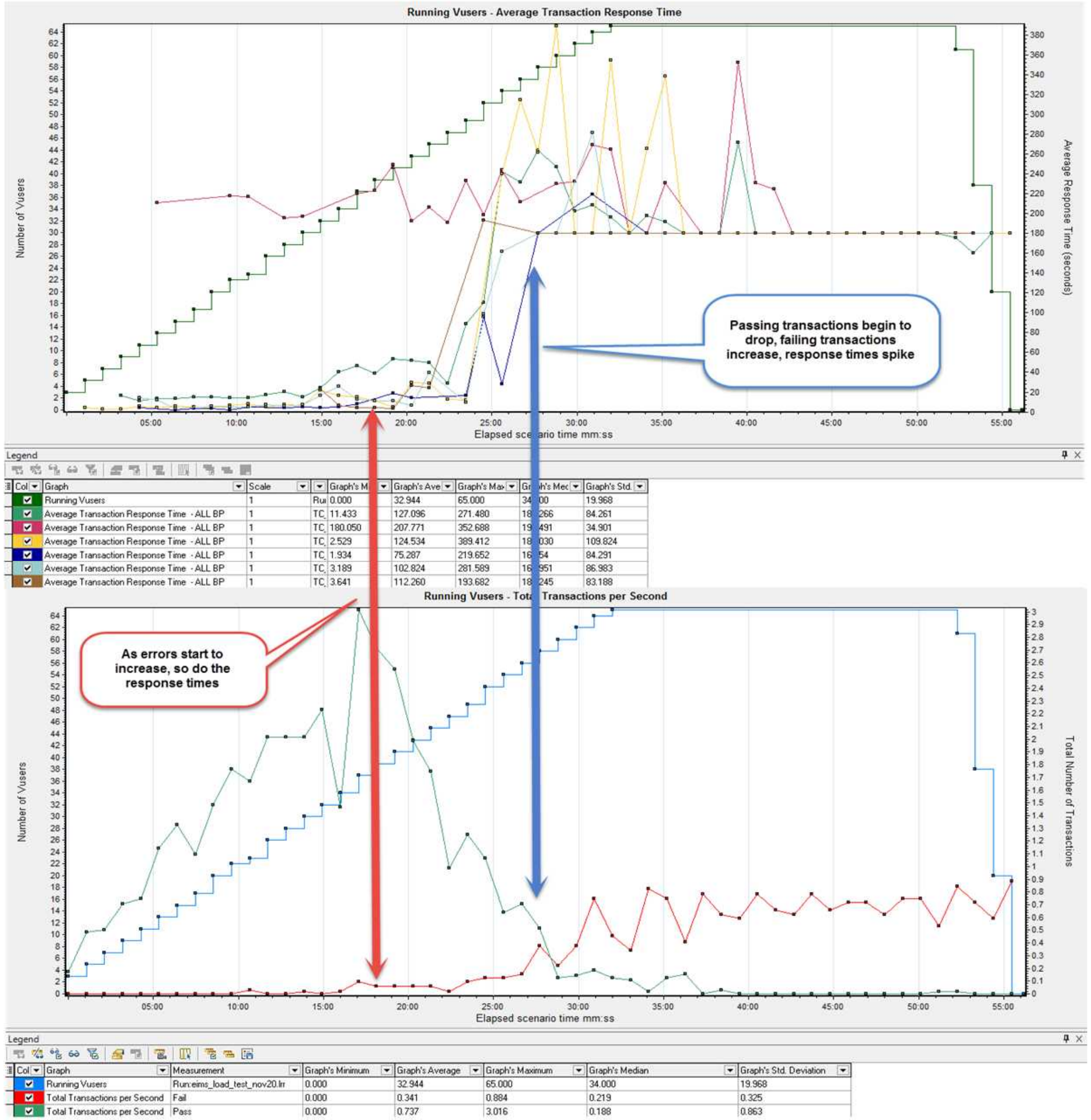
Col	Sca		Minimum	Average	Maximum	Std. Deviation
☑	1	TE	10.140	42.727	131.717	39.861
☑	1	TE	11.896	62.580	175.084	52.384



Col	Sca		Minimum	Average	Maximum	Std. Deviation
☑	1	TE	8.248	39.224	125.157	43.405
☑	1	TE	11.059	65.351	258.597	67.477

Load/Stress Testing Analysis Cont…

- Performance Degradation
  - Performance stable
    - Passing transactions increasing
  - Performance degrading
    - Failing transactions start (red arrow); application response times increase
  - Performance Breaking Point
    - Passing transactions decrease; and failed transactions increase (blue arrow); application response times spike to large amounts
    - At this point, application is not responding


Running Vusers - Average Transaction Response Time
Number of Vusers
Average Response Time (seconds)
Elapsed scenario time mm:ss
05:00
10:00
15:00
20:00
25:00
30:00
35:00
40:00
45:00
50:00
55:00
Legend
Running Vusers - Total Transactions per Second
Number of Vusers
Total Number of Transactions
Elapsed scenario time mm:ss
Legend

Passing transactions begin to drop, failing transactions increase, response times spike

As errors start to increase, so do the response times

Col	Graph	Scale		Graph's M[illegible]	Graph's Ave	Graph's Max	Gr[illegible]h's Mec	Graph's Std.
☑	Running Vusers	1	Ru	0.000	32.944	65.000	34[illegible]00	19.968
☑	Average Transaction Response Time - ALL BP	1	TC	11.433	127.096	271.480	18[illegible]266	84.261
☑	Average Transaction Response Time - ALL BP	1	TC	180.050	207.771	352.688	19[illegible]491	34.901
☑	Average Transaction Response Time - ALL BP	1	TC	2.529	124.534	389.412	18[illegible]030	109.824
☑	Average Transaction Response Time - ALL BP	1	TC	1.934	75.287	219.652	16[illegible]54	84.291
☑	Average Transaction Response Time - ALL BP	1	TC	3.189	102.824	281.589	16[illegible]951	86.983
☑	Average Transaction Response Time - ALL BP	1	TC	3.641	112.260	193.682	18[illegible]245	83.188

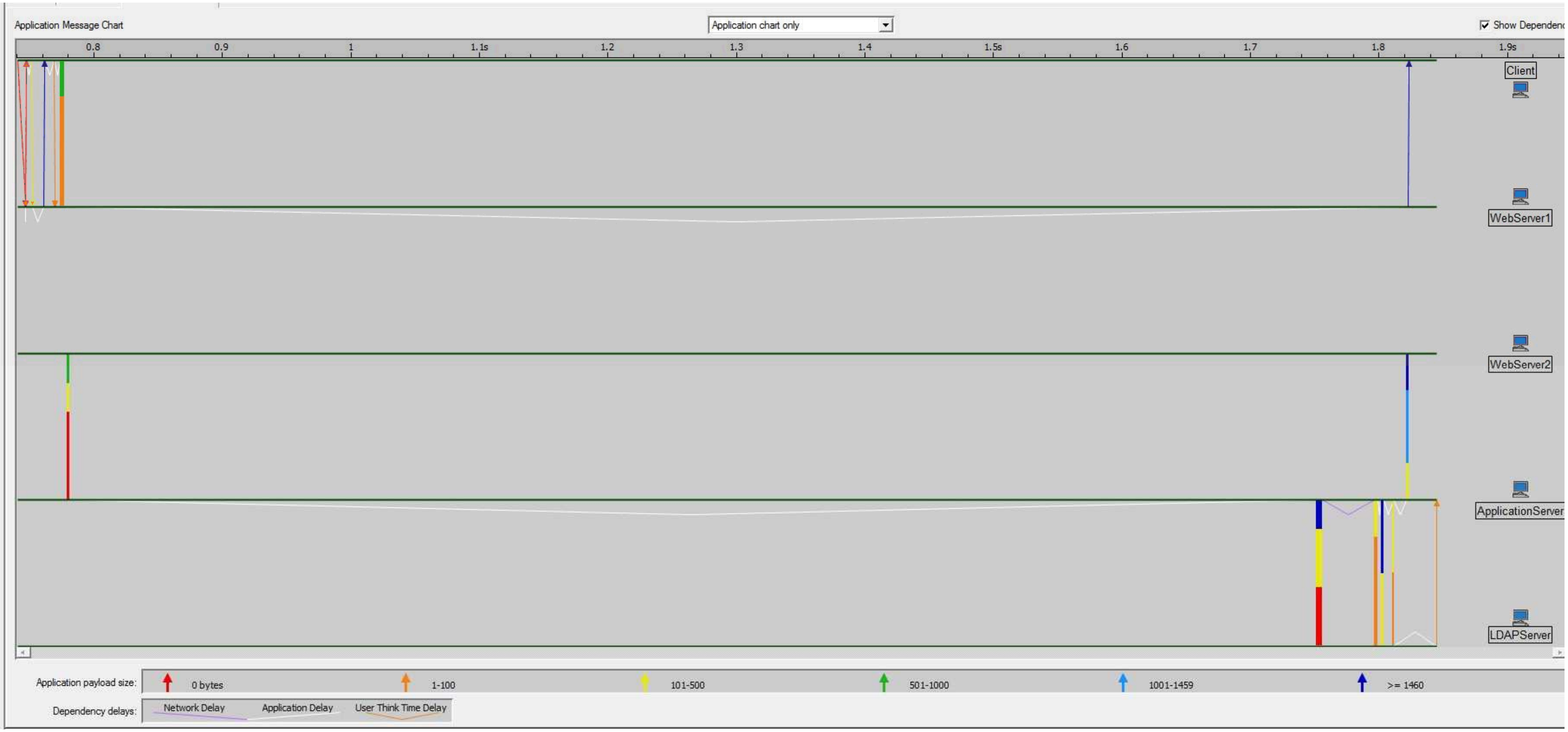
Col	Graph	Measurement	Graph's Minimum	Graph's Average	Graph's Maximum	Graph's Median	Graph's Std. Deviation
☑	Running Vusers	Run:eims_load_test_nov20.lrr	0.000	32.944	65.000	34.000	19.968
☑	Total Transactions per Second	Fail	0.000	0.341	0.884	0.219	0.325
☑	Total Transactions per Second	Pass	0.000	0.737	3.016	0.188	0.863

Transaction Analysis Baselining and Troubleshooting

- Transaction Analysis/Baseline Service
  - Used when a transaction is known to be slow and the slowness is not load-related. Determine how much time is spent at each tier of a transaction.
- When
  - Dev/Qual/Prod – when transaction is known to be slow or for baselining transactions to compare performance over time.
- Limitations
  - Troubleshooting requires collaboration with developer and other service providers
  - Tier Processing is a black box
  - Difficult to isolate transaction on multi-use servers
  - Customer may not know all servers for transaction
  - Sometimes issues are on servers unknown to customer

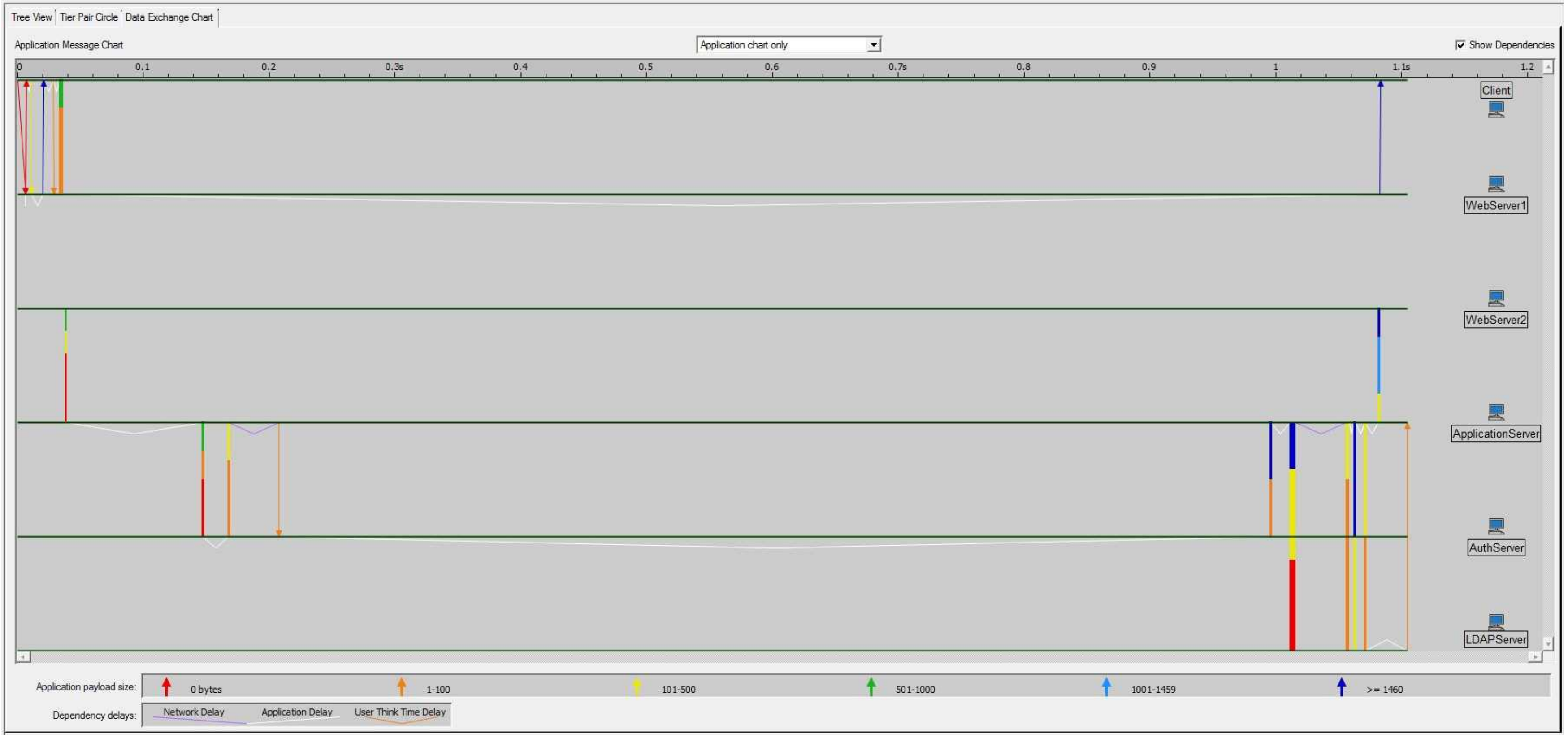
Tool: Riverbed Transaction Analyzer (packet sniffing technology)

Example Transaction Analysis



Initial troubleshooting shows tier processing on the application server
Tool Limitation: Cannot determine the exact issue on the application server
Dig Deeper…

Transaction Analysis – Dig Deeper



Returned to the original packet trace files and determined that Application Server is communicating with Authentication Server.

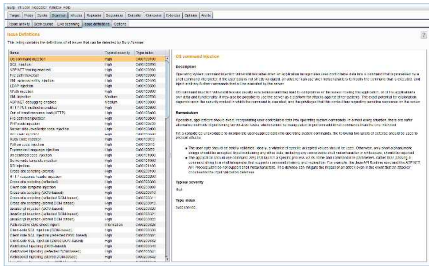
Issue: Authentication is taking too long

Web Application Scanning

- Security needs to be integrated into the all phases of the software development lifecycle

- Service
  - Testing a deployed application for security weaknesses. Tool identifies potential security weaknesses. Weaknesses need to be verified.
- When
  - At baselines during software development in quality environment
- Limitations
  - Tool does not identify design flaws, access control issues, or if external resources (e.g. *.js, *.css from internet) are being pulled into application
  - Security tools have false positive

Tools: Burp Suite, NetSparker



Issue Definitions

This listing contains the definitions of all issues that can be detected by Burp Scanner.

Name	Typical severity	Type index
OS command injection	High	0x00100100
SQL injection	High	0x00100200
ASP.NET tracing enabled	High	0x00100280
File path traversal	High	0x00100300
XML external entity injection	High	0x00100400
LDAP injection	High	0x00100500
XPath injection	High	0x00100600
XML injection	Medium	0x00100700
ASP.NET debugging enabled	Medium	0x00100800
HTTP PUT method is enabled	High	0x00100900
Out-of-band resource load (HTTP)	High	0x00100a00
File path manipulation	High	0x00100b00
PHP code injection	High	0x00100c00
Server-side JavaScript code injection	High	0x00100d00
Perl code injection	High	0x00100e00
Ruby code injection	High	0x00100f00
Python code injection	High	0x00100f10
Expression Language injection	High	0x00100f20
Unidentified code injection	High	0x00101000
Server-side template injection	High	0x00101080
SSI injection	High	0x00101100
Cross-site scripting (stored)	High	0x00200100
HTTP response header injection	High	0x00200200
Cross-site scripting (reflected)	High	0x00200300
Client-side template injection	High	0x00200308
Cross-site scripting (DOM-based)	High	0x00200310
Cross-site scripting (reflected DOM-based)	High	0x00200311
Cross-site scripting (stored DOM-based)	High	0x00200312
JavaScript injection (DOM-based)	High	0x00200320
JavaScript injection (reflected DOM-based)	High	0x00200321
JavaScript injection (stored DOM-based)	High	0x00200322
Path-relative style sheet import	Information	0x00200328
Client-side SQL injection (DOM-based)	High	0x00200330
Client-side SQL injection (reflected DOM-based)	High	0x00200331
Client-side SQL injection (stored DOM-based)	High	0x00200332
WebSocket hijacking (DOM-based)	High	0x00200340
WebSocket hijacking (reflected DOM-based)	High	0x00200341
WebSocket hijacking (stored DOM-based)	High	0x00200342

OS command injection

Description

Operating system command injection vulnerabilities arise when an application incorporates user-controllable data into a command that is processed by a shell command interpreter. If the user data is not strictly validated, an attacker can use shell metacharacters to modify the command that is executed, and inject arbitrary further commands that will be executed by the server.

OS command injection vulnerabilities are usually very serious and may lead to compromise of the server hosting the application, or of the application's own data and functionality. It may also be possible to use the server as a platform for attacks against other systems. The exact potential for exploitation depends upon the security context in which the command is executed, and the privileges that this context has regarding sensitive resources on the server.

Remediation

If possible, applications should avoid incorporating user-controllable data into operating system commands. In almost every situation, there are safer alternative methods of performing server-level tasks, which cannot be manipulated to perform additional commands than the one intended.

If it is considered unavoidable to incorporate user-supplied data into operating system commands, the following two layers of defense should be used to prevent attacks:

- The user data should be strictly validated. Ideally, a whitelist of specific accepted values should be used. Otherwise, only short alphanumeric strings should be accepted. Input containing any other data, including any conceivable shell metacharacter or whitespace, should be rejected.
- The application should use command APIs that launch a specific process via its name and command-line parameters, rather than passing a command string to a shell interpreter that supports command chaining and redirection. For example, the Java API Runtime.exec and the ASP.NET API Process.Start do not support shell metacharacters. This defense can mitigate the impact of an attack even in the event that an attacker circumvents the input validation defenses.

Typical severity

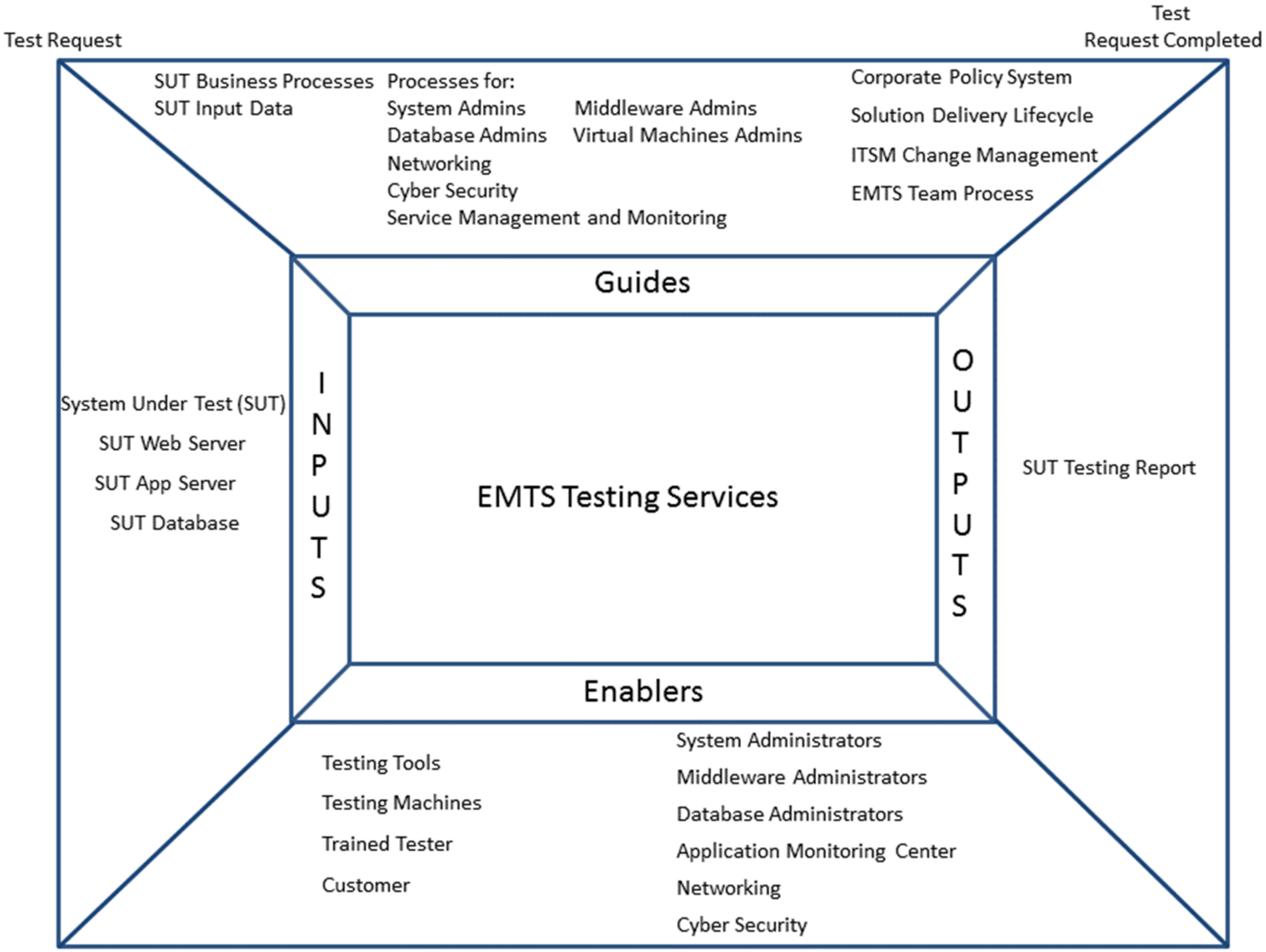
High

Type index

0x00100100

Burp Suite Demonstration

Testing Team Interfaces & Dependencies



Discussion Topics

- Does anyone at your site offer load/stress, troubleshooting, or application security scanning services?
- What types of issues have you encountered? (e.g., funding, training, customer expectations, tools)
- What type of tools do you use?
- How do you communicate your services to customers?
- Any ideas on how National Laboratories can collaborate in testing areas?