Global Information Assurance Certification Paper

Auditing for Availability: The threat of Denial of Service

GSNA Practical v4.0 Option 1, Topic 1 - "Testing"

John Soltys, March 7, 2005

Table of Contents

Abstract 3
Identification of the system 4
Risk Analysis 5
  Reduction of capacity of core content 6
  Denial of Service due to non-throttled programs 8
  Impact on other company applications 10
Testing for vulnerabilities 10
  Testing for reduction of capacity of core content 10
  Testing for Denial of Service due to non-throttled programs 16
  Testing for impact on other company applications 16
Auditing notarealnewspaper.com and notreallyclassifieds.com 18
  Reduction of capacity of core content findings 18
  Denial of Service due to non-throttled programs findings 20
  Impact on other company applications findings 22
Conclusions 25
Appendix A 26
References 28
Endnotes 29

Abstract

As the Internet becomes an increasingly important means to deliver information newspaper web sites must face the challenge of always being available for their readers. This often means being able to handle unpredictable traffic spikes and high loads when news breaks. This paper discusses three risks related to availability when serving the news and methods of mitigating their impact. It describes tests to validate the implementation of safeguards and concludes with an audit of a real newspaper site and its associated online classifieds site.

Identification of the system

In a world where the computer has become ever more important the word "system" has become synonymous with a computer and the software that runs it[1]. More generically, a "system" is defined as "a group of interdependent items that interact regularly to perform a task."[2] Given this definition, think of a major newspaper website. It might have caching proxy servers that the public talks to. It could have web servers that serve stories. It probably has a database that allows for dynamic generation of pages and to generate those pages when a user requests them there are programs, likely using the Common Gateway Interface (CGI).

notarealnewspaper.com[3], the website for a large regional newspaper is such a system.

As the public website for the newspaper the site is an integral part of the overall strategy of the company. In developing the site in the mid 1990s and continuing development into the 21st century the company staked its claim to a part of cyberspace. The site provides a medium that is not limited by column inches and provides a platform for the newspaper to both defend its brand and serve the community. If it happens to make some money so much the better.

For the purpose of this paper only the "core" of notarealnewspaper.com will be audited. The core is made up of those parts of the site that are considered critical and must be maintained above all else even in time of major news event or disaster. The newspaper's management defines the core as

- the front page
- the national news index
- the local news index
- the sports news index
- the content linked to from the front page and national, local, and sports indexes

After several traffic spikes caused by major news events brought the site down it was redesigned with maximum capacity in mind.

Rather than allow the public to connect directly to the newspaper's web servers, the company contracts with a vendor to act as a proxy. Users connect to the proxy servers that, in turn, connect to the newspaper's web servers. The proxy servers also perform content assembly based on special instructions embedded in the pages served from the newspaper's web servers. Furthermore, the proxy servers have the ability to cache content so the newspaper isn't required to repeatedly serve the same content over and over, regardless of how many people want to see it.

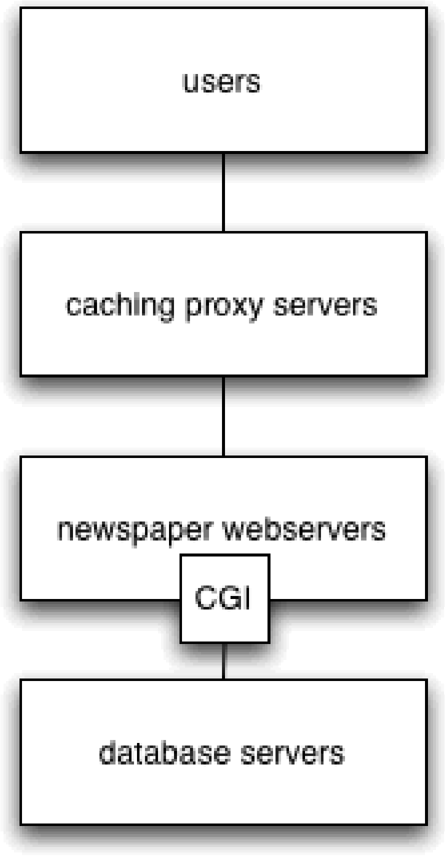
The site is built almost exclusively from static files. In other words, pages are assembled before they are made available through the newspaper's web server.

This is in stark contrast to the concept of a dynamically generated website that must run a program for each page it serves. Static pages are inherently faster and less resource-intensive than dynamic pages. The proxy servers bring a measure of dynamism by combining static files before serving the results to the end user.

While a purely static website promises extremely high capacity there are some business functions that cannot be accomplished with unchanging files. To fulfill these requirements a small number of programs run on the newspaper's web servers.

The final piece is a database that is used to house the content while it's in development and to simplify some of the tasks of the programs generating dynamic content.

A simple diagram of this system might be drawn like this



Such a system performs very well under heavy load, but if there's only one key concept to remember about risk it's that "there's just no such thing as zero risk[4]."

Risk Analysis

In fact, there are several risks to the system. Risk is a function of vulnerability and threat[5]. Exploiting such a vulnerability results in an impact to the system and, usually, to the business that depends on the system.

In the case of notarealnewspaper.com there are three impacts related to availability that are of immediate concern.

1. A reduction in the capacity of the system to serve the core content.
2. Denial of Service (DoS) as a result of non-throttled programs.
3. Impact on other company applications.

During times of high load each of these impacts can result in a loss of the reader's trust. In a time when the newspaper industry is constantly under siege by a dizzying array of new voices maintaining a close relationship with the reader is crucial to the survival of the company.

If readers can't get their news from notarealnewspaper.com they're only a few keystrokes away from another news site. How many times will it take before a reader starts at that other news site before it tries to visit notarealnewspaper.com and that business is essentially lost?

In order to maintain the newspaper's readers and truly serve the community each of these impacts must be fully understood and the vulnerabilities mitigated as appropriate.

Reduction of capacity of core content

As described earlier, notarealnewspaper.com was redesigned to provide maximum capacity and perform well even under heavy load. One of the key aspects of the technical redesign was completely transparent to the user, but had an enormous impact on the site.

The newspaper's classifieds site, notreallyclassifieds.com[6], was originally hosted on the same set of hardware as the news site. The two applications were completely separate, sharing users only through hyperlinks. The only technical link between the two sites was that they competed for the same limited set of resources and would be impacted by traffic to one another.

Early in the redesign project the classifieds application was moved onto a set of dedicated servers while the news application was moved to faster servers to increase its performance. The result of this architectural change was the elimination of periodic news site slowdowns while the classifieds site updated and an end to classifieds application downtime when the news site experienced a traffic spike. Both sites benefited from the modest expense of a few new servers.

As the sites continued to evolve both began making use of the content assembly options provided by the content delivery vendor's proxy servers. The most commonly used feature of the assembly language was a simple include. The include statement instructed the proxy server to insert the result of a secondary HTTP request in place of the statement itself. In other words, if a document included this fragment

```
<table>
	<tr>
		<td>
			<proxy:include src="/components/left_nav.incl"/>
		</td>
		<td>
			<proxy:include src="/components/story1.incl"/>
		</td>
	</tr>
</table>
```

The proxy server would attempt to retrieve /components/left_nav.incl and /components/story1.incl and replace the proxy:include tags with them. The content sent to the end user would appear like this

```
<table>
	<tr>
		<td>
			<a href="/">Home</a>
			<br>
			<a href="/national/">National News</a>
			<br>
			<a href="/local/">Local News</a>
			<br>
			<a href="/sports/">Sports</a>
		</td>
		<td>
			<b>Local hiker completes 2,000 mile trek</b>
			<br>
			Exhausted, but content, a local hiker completed
			a 2,000-mile hike along the ...
		</td>
	</tr>
</table>
```

Assembling pages on the proxy servers provided both productivity and financial benefits. If the navigation was updated in one file (/components/left_nav.incl) all the pages that included it via a proxy:include tag would also be updated. Another feature of the proxy servers allowed for a component to be cached. If the navigation changed very rarely it could be stored in the proxy server's cache for long periods of time (a day, for instance) reducing the resources required to send that content between the company's web servers and the proxy servers. Since every bit crossing the wire between the company and the proxy servers cost the company money this represented a significant savings.

In addition to allowing files from the same server to be included the proxy:include statements could target a URL pointing at another server. To include a component from the classified site a notarealnewspaper.com designer might use a proxy:include tag like this

```
<proxy:include src="http://notreallyclassifieds.com/fragments/ad1.incl"/>
```

This feature allowed an unprecedented ability to collaborate between the two sites and soon its use was widespread. Unfortunately, such collaboration introduced a vulnerability previously unknown.

If the page that contained the proxy:include statement had a capacity of 1,000 pages/second, but the included file had a capacity of only 100 pages/second the effective capacity of the overall page was reduced to 100 pages/second. All the work that had gone into increasing the capacity of the core news content had suddenly been tied to the capacity of a site never meant to handle high traffic volumes.

The impact of this risk is a poor user experience. Although it could be argued that the newspaper's technical ability to serve the content has little to do with the quality of the content that connection is easily made in the mind of the reader. In the span of just a few incidents users begin to favor other news sites.

Mitigation of this risk turns out to be quite simple. The proxy:include tag has several optional parameters that can be used to protect against specifically this type of vulnerability.

- alt - an alternate file to retrieve and include if the main file cannot be retrieved
- else - the action to take if the include request fails
- timeout - the amount of time, in seconds, the proxy should wait before abandoning the request for the included file (overriding the 60 second default timeout)

With the inclusion of these optional parameters the tag now looks like this

```
<proxy:include src="http://notreallyclassifieds.com/fragments/ad1.incl" \
alt="/components/adnotavailable.incl" else="continue" timeout="5s"/>
```

Although slightly more complicated to code the tag provides for a locally stored alternate component, a limit of five seconds to process the entire request, and instructions to continue if any part of the request fails.

Denial of Service due to non-throttled programs

Even though the site was designed to be available under high load, which implied a static file configuration, some business functions could not be implemented as static files. These functions required programs to interact with the end user and these programs had capacities several orders of magnitude less than static HTML files.

If the web server could serve 1,000 static pages/second it might be able to manage only 10 dynamic pages/second. Furthermore, when load increased above this low threshold the system would become more and more unresponsive until it was too busy even to handle the relatively simple task of delivering a static HTML file.

Such a Denial of Service could be instigated intentionally or unintentionally, with or without malicious intent. If the program provided a feature compelling enough it might be linked to by any number of high-traffic sites such as slashdot.com or fark.com. (The effect of being overwhelmed due to links from such sites has introduced slang such as farked[7] and slashdotted[8].)

On the other hand, if a malicious user discovered the low capacity and the high impact of such a script he could easily write an attack agent to repeatedly request the URL corresponding to the program and cause a Denial of Service.

When the site is completely unavailable its users are again left to question their patronage of such a site. One of the greatest promises of the Internet is the immediate availability of information and the knowledge that there are many, many alternate sources of the information. Having problems on notarealnewspaper.com? What about someothernewspaper.com[9]?

Unfortunately, mitigation of Denial of Service is a difficult problem, especially when the traffic is legitimate or the attack is distributed across many hosts (called a Distributed Denial of Service attack or "DDoS"). The programs need to be available on the servers for low-levels of legitimate traffic, but under no circumstance should they be allowed to interrupt the delivery of the core news content.

The solution to such a problem is the introduction of a concept known as "throttling." Under typical load the programs are allowed to run as normal. However, when the number of running instances exceeds some threshold the programs should stop processing and return a simple error message such as, "The site is extremely busy, please try again later."

Unfortunately, this approach is not always successful. Although throttling works for moderate levels of load even the lightweight task of determining how many instances are running cannot return quickly enough under high load to prevent a feedback loop. While the program is trying to shut itself down several more requests come in and soon there are no additional resources to run the program or to serve the core news content.

An additional layer of defense is added as an agent that watches CPU utilization on the server. If utilization goes above a threshold for a certain amount of time the agent disables the programs at the web server level. Subsequent requests result in a web server-generated error message rather than the one displayed when the program can throttle itself, but the ability to serve the core content is protected.

Impact on other company applications

Earlier the impact of a lower-capacity notreallyclassifieds.com site could have on the higher-capacity notarealnewspaper.com site was discussed. If the perspective is shifted to the point of view of notreallyclassifieds.com it becomes apparent that there is a tremendous impact on the classified site when the news site experiences a traffic spike.

notreallyclassifieds.com is designed as a heavily interactive site focused on the business of the newspaper. It accepts advertising content from customers and allows readers to search through several days of classified listings. Neither of these functions can really be accomplished with static HTML files. The site doesn't have the same kind of flash-crowd that can instantaneously appear at the news site in the wake of a major news event such as an earthquake or a terrorist attack so the design of a heavily dynamic site with low capacity is completely within reason.

The exception to this statement comes from the inclusion of components from the classifieds site on the news site. If a dynamic component such as a "featured ad" is generated on the fly and included on a high-traffic page on the news site the classified site will almost certainly suffer a Denial of Service incident at the hands of its sister site.

Even if the news site implements the mitigation techniques outlined above the classified site will still go down, though the impact on the news site will be lessened. Although failure of the news site remains the chief impact, more and more business is driven through the classifieds and the company's other non-newspaper sites. Over time, these sites will rise to the same level of prominence as the news site and will in fact eclipse it in terms of importance to the overall mission.

Unfortunately, the only mitigation readily available is to not use dynamically generated content as an include on a high-traffic page. Whenever possible, dynamically generated components should be converted to static components.

If a component must be generated dynamically the server should enforce a timeout on page delivery slightly longer than the timeout specified in the calling proxy:include statement and load should be monitored closely in order to quickly detect failure. In the event load becomes unmanageable notreallyclassifieds.com could elect to block any requests from notarealnewspaper.com to protect itself.

Testing for vulnerabilities

Identifying vulnerabilities is only one step an auditor must take, but if he stops there his job is left incomplete. The next phase of an audit requires creating tests that will expose the presence of the vulnerabilities.

Testing for reduction of capacity of core content

Testing for a reduction in capacity requires knowing the capacity of each component. This is a daunting task, however, when you consider the number of individual components that make up a site the size of notarealnewspaper.com. Fortunately, it's possible to determine the capacity of a class of components, static HTML, for example, and use that capacity as the basis for risk assessment.

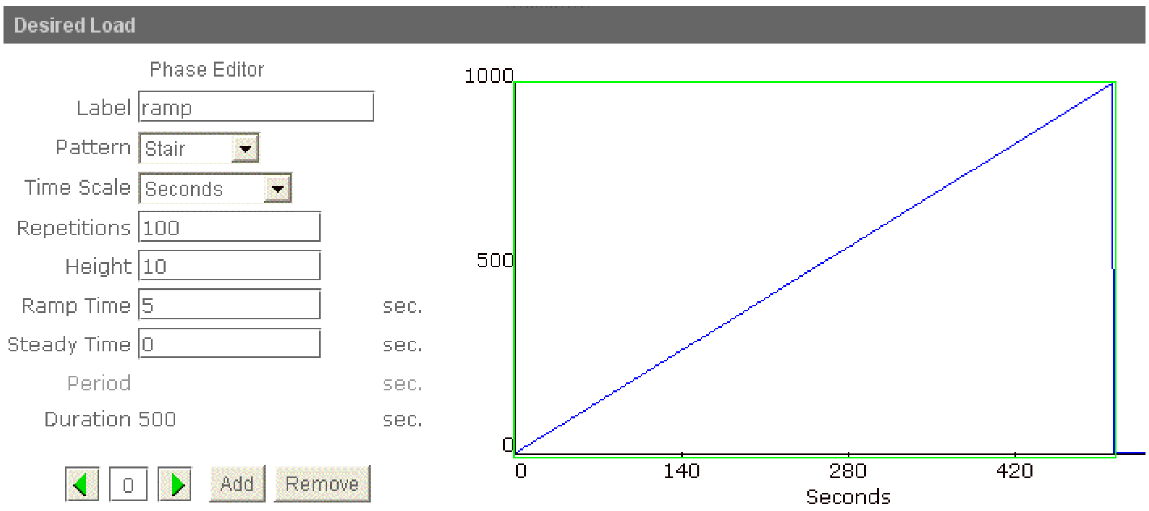
There are many tools that can be used to perform capacity or stress tests. A simple Perl script could be written to generate traffic to the target web server, but with unsophisticated data collection. Paessler's Webserver Stress Tool[10] provides an entry-level, low-budget tool to test medium-capacity sites from a single Windows PC. Mercury's LoadRunner[11] is the proverbial Cadillac of testing tools with a price to match. A typical LoadRunner deployment consists of a Controller and one or more Virtual User Generators.

A compromise between these two extremes is Spirent's Avalanche 220[12]. The Avalanche is an appliance capable of generating up to 200 Mbps of data via two 100 Mbps Ethernet ports and testing a variety of protocols. In order to test the vulnerabilities identified above the Avalanche should be configured to generate standard HTTP traffic.

Detailed instructions for configuring an Avalanche test are beyond the scope of this paper. However, the methodology for determining the capacity of components (and by extension classes of components) will be covered.

The first step is finding the breaking point. The approach below is an iterative one, starting with a coarse test and then refining it to identify the specific amount of load that causes errors.

The Avalanche allows the tester to create a "load profile" that sets up the desired load. In the initial test the goal is to identify the approximate amount of traffic required to cause failure so the load is quickly ramped up until errors appear.



Desired Load

Phase Editor

Label: ramp
Pattern: Stair
Time Scale: Seconds
Repetitions: 100
Height: 10
Ramp Time: 5 sec.
Steady Time: 0 sec.
Period sec.
Duration 500 sec.

A fast ramp is configured by adding 10 transactions/second every five seconds with no pause between steps. (This setup assumes the system is expected to fail at some level of traffic below 1,000 transactions/second.)

A "URL list" is created in order to provide the Avalanche with URLs to test against. The format of the list is

<level> <method> <URL>

for instance

```
1 GET http://www.notarealnewspaper.com/components/story1.incl
```

Requests can be made using the GET, HEAD, and POST methods as well as protocols other than HTTP.

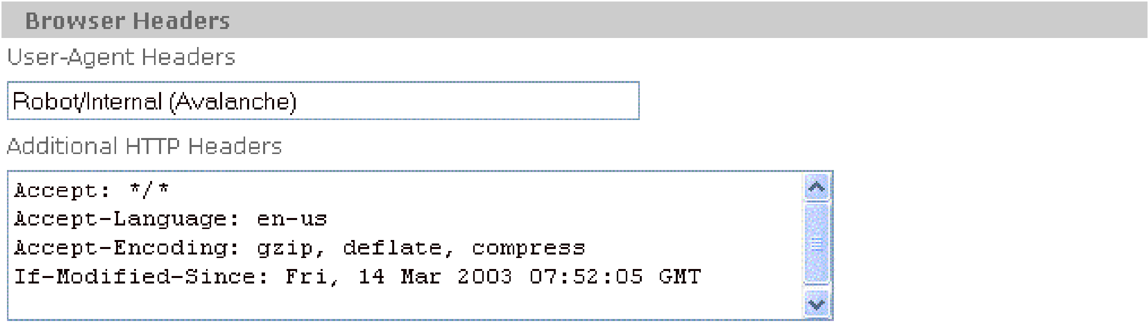
An important consideration when testing high-capacity content has to do with bandwidth. In an HTTP request the amount of data sent from the client to the server is relatively small. It can be as little as the path to the object being requested and the name of the server the client is talking to.

```
GET /components/story1.incl HTTP/1.1
Host: www.notarealnewspaper.com
```

The amount of content that is sent back is almost always much larger. In the case of a news story it can be hundreds or thousands of words long. A common feature of caching systems, whether a web browser or a reverse proxy such as the one in use by notarealnewspaper.com is the ability to ask the web server if content has changed.

Most web servers support the `If-modified-since` header. This header provides a date and timestamp that the server can use to check against its content. If the content hasn't changed since the date provided by the client the server returns an HTTP status code of 304, which tells the client that it's ok to use the content from cache. If the content has changed the web server treats the request as though it had been a standard GET request and serves the content with a status code of 200.

In order to replicate this behavior and accurately simulate load from the proxies the Avalanche's User Profile needs to be modified to include an appropriate If-modified-since header.



User-Agent Headers: Robot/Internal (Avalanche)

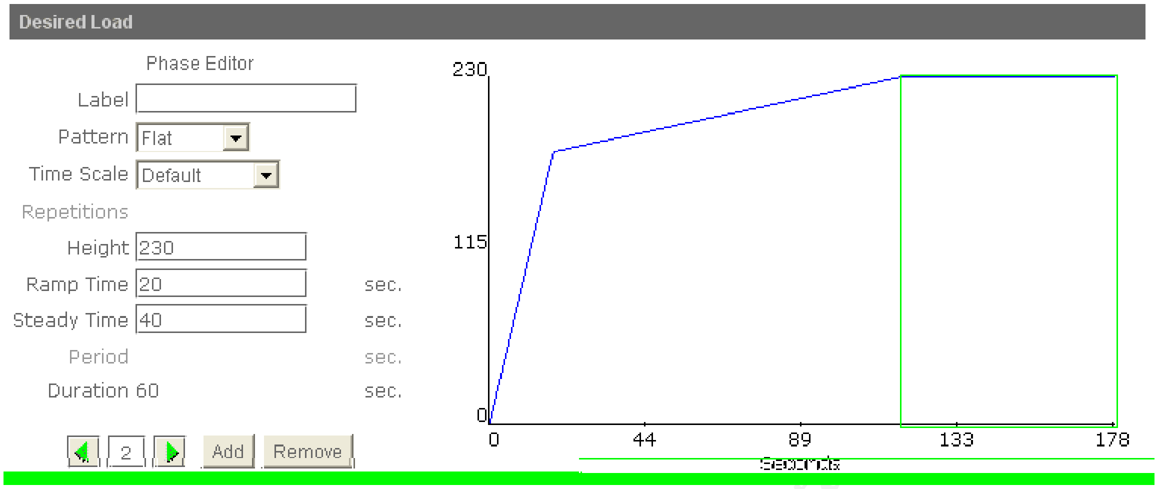
Additional HTTP Headers:

```
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate, compress
If-Modified-Since: Fri, 14 Mar 2003 07:52:05 GMT
```

While it's tempting to set an If-modified-since date in the distant future most web servers will discard an If-modified-since header if it has a date ahead of the server's internal clock.

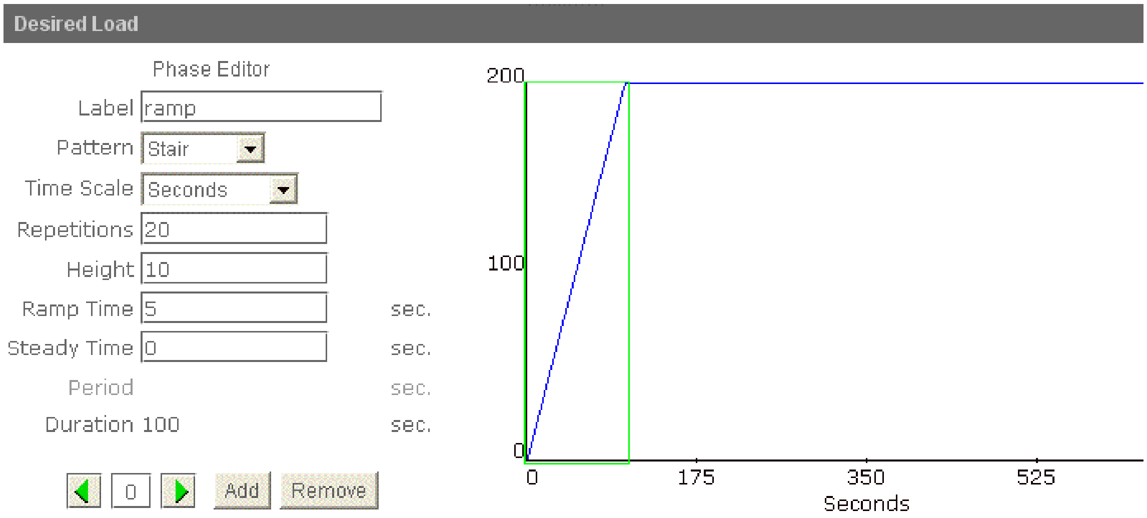
It's also important to set an easily identifiable user-agent within the User Profile. The user-agent is included with the request and most web servers will log it. In order to prevent tests such as these from skewing traffic analysis Avalanche-generated data should be stripped from the logs before they are processed. (Another method of identifying Avalanche traffic is by IP.)

To more accurately define the breaking point this test is repeated with a more gradual increase, but starting at a higher initial load. Assuming failure was seen around 200 transactions/second in the initial test the second test might be quickly ramped to 180 transactions/second, but then slowed to increase by five transactions/second every 10 seconds.



Label
Pattern Flat
Time Scale Default
Repetitions
Height 230
Ramp Time 20 sec.
Steady Time 40 sec.
Period sec.
Duration 60 sec.

Now that an accurate failure point has been identified the next step is to test the web server's ability to sustain a slightly lower level of load. In this example assume failure occurred at 210 transactions/second. To run a "burn-in" test the configuration is set to quickly ramp to 200 transactions/second, but rather than increasing load until failure load is maintained at 200 transactions/second. The purpose of this test is typically to identify memory leaks and other problems that only occur in a time of sustained load.



Label ramp
Pattern Stair
Time Scale Seconds
Repetitions 20
Height 10
Ramp Time 5 sec.
Steady Time 0 sec.
Period sec.
Duration 100 sec.

By running tests such as those described above the auditor will be able to document the capacity of the various classes of content used by a site. To build a list of which classes must be assessed all the proxy:include tags must be extracted and the objects they target added to the list. On a large site this is impractical, if not impossible to do by hand. However, standard tools can be used to assist in this task.

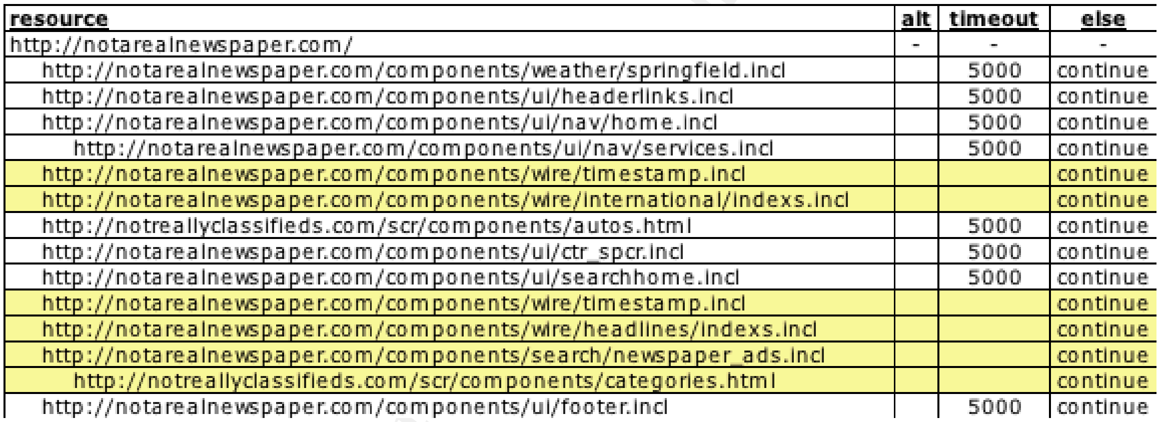
One such tool is GNU grep[13]. Grep is a regular expression matching tool that can search for patterns in files. Grep, of one version or another, is included in most *nix operating systems and several ports are available for Windows[14].

If all the files that might contain a proxy:include statement are stored in /www/content this command will find all the tags and save them to a file called grep.output

```
grep -ir "proxy:include" /www/content > grep.output
```
[15]

Each saved line will include the filename it was found in as well as the complete line. Each source must be put into a class or assessed on it's own.

Another approach is to write a tool to collect the data through the website. While not strictly necessary, knowledge of a scripting language such as Perl[16] can often make an auditor's tasks much easier. Using a simple script of only a couple hundred lines[17] an auditor can produce a report such as the one shown below by simply providing a starting URL.



resource	alt	timeout	else
http://notarealnewspaper.com/	-	-	-
http://notarealnewspaper.com/components/weather/springfield.incl		5000	continue
http://notarealnewspaper.com/components/ui/headerlinks.incl		5000	continue
http://notarealnewspaper.com/components/ui/nav/home.incl		5000	continue
http://notarealnewspaper.com/components/ui/nav/services.incl		5000	continue
http://notarealnewspaper.com/components/wire/timestamp.incl			continue
http://notarealnewspaper.com/components/wire/international/indexs.incl			continue
http://notreallyclassifieds.com/scr/components/autos.html		5000	continue
http://notarealnewspaper.com/components/ui/ctr_spcr.incl		5000	continue
http://notarealnewspaper.com/components/ui/searchhome.incl		5000	continue
http://notarealnewspaper.com/components/wire/timestamp.incl			continue
http://notarealnewspaper.com/components/wire/headlines/indexs.incl			continue
http://notarealnewspaper.com/components/search/newspaper_ads.incl			continue
http://notreallyclassifieds.com/scr/components/categories.html			continue
http://notarealnewspaper.com/components/ui/footer.incl		5000	continue

Once all the classes have had capacities established the next step is to determine if any high capacity pages contain proxy:include statements referring to lower capacity pages. Unfortunately, there is no easy way to do this other than examining the pages in their raw form.

Remember that when the proxy server builds a page it will replace a proxy:include statement with the contents of the included file. If the auditor attempts to view pages after they have passed through the proxy server no proxy:include statements will be found. However, since the newspaper web server makes the raw pages available they can still be retrieved via HTTP and can also usually be retrieved via file transfer.

Take the front page, for instance. The output of the `grep` command we used before shows us there are three proxy:include statements in the front page HTML file. Since each of these include files could contain their own proxy:include statements we must assess each as though it was a stand-alone page. By recursing down the proxy:include tree we can build the component structure of the page.

```
/front_page.html
        /components/index_masthead.incl
        /components/lead_story.incl
                /components/story_masthead.incl
                /components/story_nav.incl
                http://notreallyclassifieds.com/fragments/ad1.incl
        /components/index_nav.incl
```

Knowledge of the site allows us to classify all the components and the front page itself as "static HTML." Testing shows that the "static HTML" class of files have a certain capacity. If the testing showed that the component served from notreallyclassifieds.com had a significantly earlier failure point and the proxy:include statement is missing the alt, else, and/or timeout parameters it indicates that the availability of front_page.html has been compromised. The front_page.html now has the capacity of the content served from notreallyclassifieds.com.

Testing for Denial of Service due to non-throttled programs

Testing for Denial of Service due to non-throttled programs first requires that the auditor assemble a list of all the programs that can be accessed via the web server. Unlike static files on disk, these programs cannot usually be classified as easily. Each program performs a different function and likely consumes different amounts of resources to get the job done. As a result, each program must be assessed independently.

The Avalanche can again be employed to determine capacity and test performance under sustained load, but the levels of traffic the Avalanche generates are likely to be significantly less.

There are two separate failure points that need to be identified for each program.

1. The first is the point at which it can no longer control itself. If the program includes throttling code it will use some method of counting concurrent instances on the server. When a threshold (X) is passed it will abort any new requests it receives. However, at some point the program will no longer be able to determine how many instances of itself are already running. This is the throttle's failure point.

2. The second is the level of traffic that impacts the ability of the server to handle requests for the core content. Remember that the point of throttling is to sacrifice the dynamic functionality in favor of the higher-capacity core content. At this point the agent that monitors CPU utilization should trigger the process of disabling the program. The highest amount of traffic at which the agent can still operate is the agent's failure point.

Once these two failure points are known an auditor can answer the fundamental questions about dynamically generated pages.

- When placed under moderate load (below the throttle's failure point), does the program keep itself in check? Are there only X concurrent instances of the program running? If not, the throttle code is not working properly.

- When placed under heavy load (above the throttle's failure point, but below the agent's failure point) does the agent disable the programs? Once the agent has triggered the shutdown of the programs does the server return to a more stable condition? If not, the trigger may be set to fire too late resulting in a situation in which the server cannot recover.

Testing for impact on other company applications

Testing for the impact of notarealnewspaper.com on notreallyclassifieds.com borrows heavily from the earlier tests looking for reduction in capacity of the core content. The work done building component structures of the high capacity pages will reveal which notreallyclassifieds.com components are in use by notarealnewspaper.com as well as their capacity.

With the tests establishing capacity already completed earlier there remain only a few tests that can be performed to check for compliance.

The first is ensuring that any dynamically generated component really must be generated at the time of the user's request. In many cases it's simpler for a developer to write code that generates content on demand rather than writing files to disk.

In the event there are components that must be dynamically generated their ability to terminate themselves after the proxy server aborts the request must be tested. Even if the proxy:include statement on notarealnewspaper.com contains a timeout parameter and the request is aborted it is no guarantee that the web server on notreallyclassifieds.com will stop processing the request.

To evaluate this ability the Avalanche can be used to generate enough load for notreallyclassifieds.com to push page generation beyond the timeout set in the proxy:include statements on notarealnewspaper.com. For instance, if the timeout parameter is set to five seconds and the classified programs are set to terminate after six seconds the Avalanche should be configured to make requests sufficient to force programs to run for more than seven seconds.

This load configuration is more similar to the burn-in type of test than the breaking point test. The purpose is not to crush the server under load, but rather to evaluate the configuration of the timeout. A follow-up test should be performed that generates heavy load in order to test this same ability when traffic is spiking.

Success for this test could come in the form of log entries showing program termination after six seconds or other empirical evidence of the same.

The final test reveals the impact of a newspaper traffic spike on the classifieds site even if there are no classifieds components being called from the newspaper pages or if the classifieds site is denying requests from for components. Problems in this scenario typically appear in the form of resource contention such as network utilization or access to a shared database.

In order to perform this test the newspaper site must be loaded with care to avoid any requests to the classified site. This can be accomplished by choosing to request only pages with no proxy:includes referring to notreallyclassifieds.com or by configuring the classifieds site to deny access to those components.

With a stable, high level of load on the newspaper site a functional test of the classifieds site is performed. The tests previously described gave little consideration to application functionality than to the ability of the web server to deliver a page. In the case of static HTML pages the two are very closely linked. If the web server delivers the page the expected functionality is also delivered.

However, for an application such as those running on the classifieds site there is more to a success than serving HTML to the end user. For instance, if a user is trying to search the classifieds ads for an automobile listing he might receive an error message from the database embedded within an HTML page. The page delivery was successful, but the search attempt failed.

In order to test the functionality it is critical that the auditor choose 100% reproducible URLs and benchmark the expected behavior before the test begins. An excellent way to do this is using a tool called wget. wget is a *nix[18] and Windows[19] tool that allows the user to capture the raw result of an HTTP request from the command line. The benefit to using wget over viewing a page through the browser is it can be automated and the results can be compared character-by-character to find subtle changes.

The command to capture a page might look like this[20]

```
wget -N -s -O in-test_output.html \
http://notreallyclassifieds.com/cgi-bin/search.pl?q=Land+Rover&cat=autos
```

By running this command before, during, and after heavy load is applied to the newspaper site (modifying the `in-test_output.html` parameter to specify different filenames) results in three separate versions of the same page. To compare the files another GNU utility called diff[21]. diff compares files character-by-character and reports the differences. If the files are identical no output is produced.

```
diff pre-test_output.html in-test_output.html
```

Success is characterized by no output when comparing the pre-test, in-test, and post-test output. Examining the output generated between non-identical files will assist the auditor in identifying what the load on the newspaper site affected. Since the output of wget is actually an HTML file it can be loaded into a web browser for a graphical view of the differences[22]. This is sometimes easier to understand than diff's output.

Auditing notarealnewspaper.com and notreallyclassifieds.com

Using the tests defined in the previous section notarealnewspaper.com and notreallyclassifieds.com were audited for availability. Below are the findings.

Reduction of capacity of core content findings

Analysis of the front page of notarealnewspaper.com found 13 proxy:include statements. Of these seven were missing timeout parameters and none specified an alt parameter. All but one were served from notarealnewspaper.com. All these were static HTML.

The single component included from notreallyclassifieds.com did have a timeout parameter and it referenced a static file.

Two of the static components served from notarealnewspaper.com contained proxy:includes themselves. One referenced a static component on notarealnewspaper.com and one referenced a static component on notreallyclassifieds.com. This last reference did not include a timeout parameter, which would result in a 60 second wait while rendering its parent component.

The story is similar for the local, national, and sports section pages. Between the three section pages there were 24 proxy:include statements. 10 of these were missing a timeout parameter, but all of these referenced static components on notarealnewspaper.com.

All three section pages included a nested proxy:include statement referencing a static component on notreallyclassifieds.com. None of these statements included a timeout parameter. In fact, it was same second-level component on all three pages that made use of this vulnerable proxy:include statement.

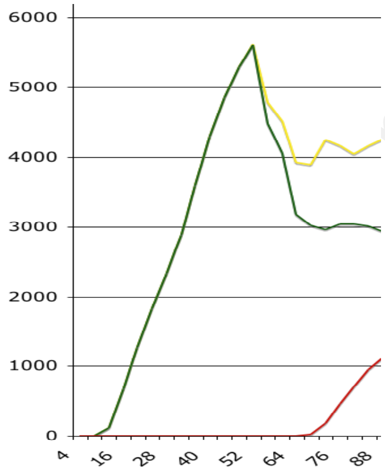
At first blush this appears to show that the same vulnerability is present on multiple pages. However, by modifying the vulnerable component in one location the risk to all three section pages is mitigated in one step.

90 pages were linked to from the front page containing over 600 components. The three section pages linked to over 120 other pages. The pages linked from the section pages contained an additional 532 components.

Although over 1,000 pages and components were identified in the initial analysis only two classes of content emerged requiring testing.

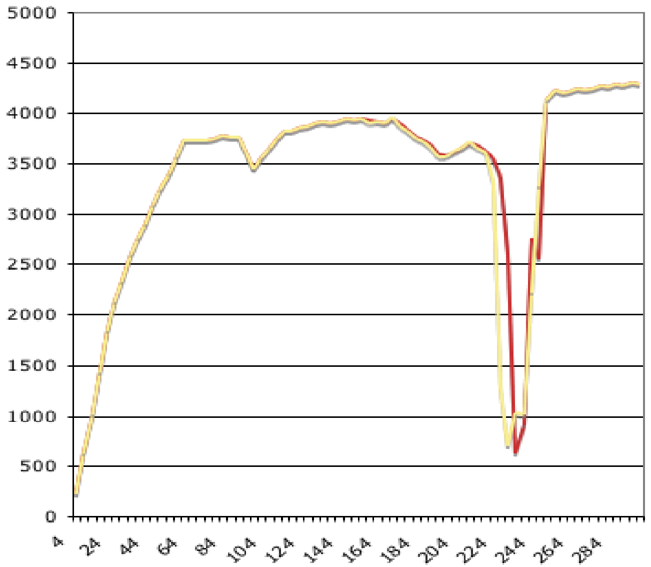
- static HTML on notarealnewspaper.com
- static HTML on notreallyclassifieds.com

Testing against notarealnewspaper.com revealed the system could scale to approximately 5,500 static pages/second before a the web server failed. However, after failure the server continued to serve pages at approximately 3,000 pages/second.



The yellow line shows requests that timed out. The red shows errors.

A similar test against notreallyclassifieds.com showed its web server could handle a fast climb to 3,400 static pages/second before it lost the ability to scale as quickly. After that point it remained fairly stable until approximately 240 seconds into the test when the server restarted. Following the restart the server appeared to have additional capacity.



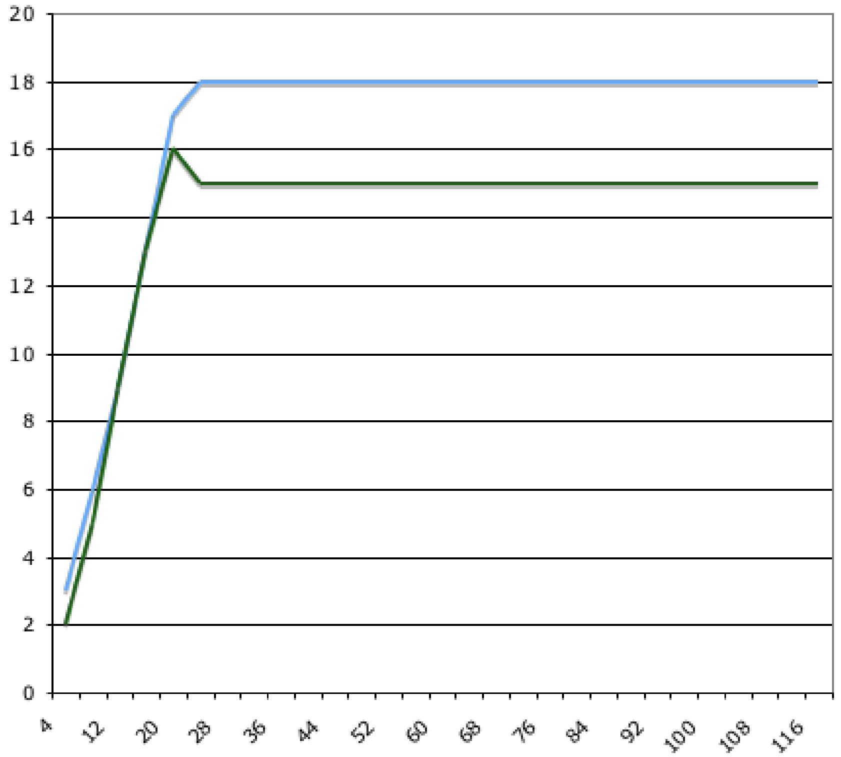
The yellow line represents connections. The red line shows successful requests.

These tests show that a static page on notarealnewspaper.com with a proxy:include statement referring to a static component on notreallyclassifieds.com has an effective capacity of approximately 3,400 pages/second. Further analysis is required to identify the reason for the web server restart in the 240th second of the notreallyclassifieds.com test.

Denial of Service due to non-throttled programs findings

Three programs were identified on notarealnewspaper.com requiring analysis. After interviews with the developers, two of the three were found to not include throttle code. These programs represent a threat to the web server under just moderate load.

The third did contain throttle code with a threshold of 15 pages/second. 15 pages/second was set as the throttle point after several previous tests showed the program's throttle code failed at approximately 20 pages/second. A stress test configured to generate 18 pages/second showed that the throttle effectively limited the number of concurrent instances of the program to 15 after an initial jump to 16.



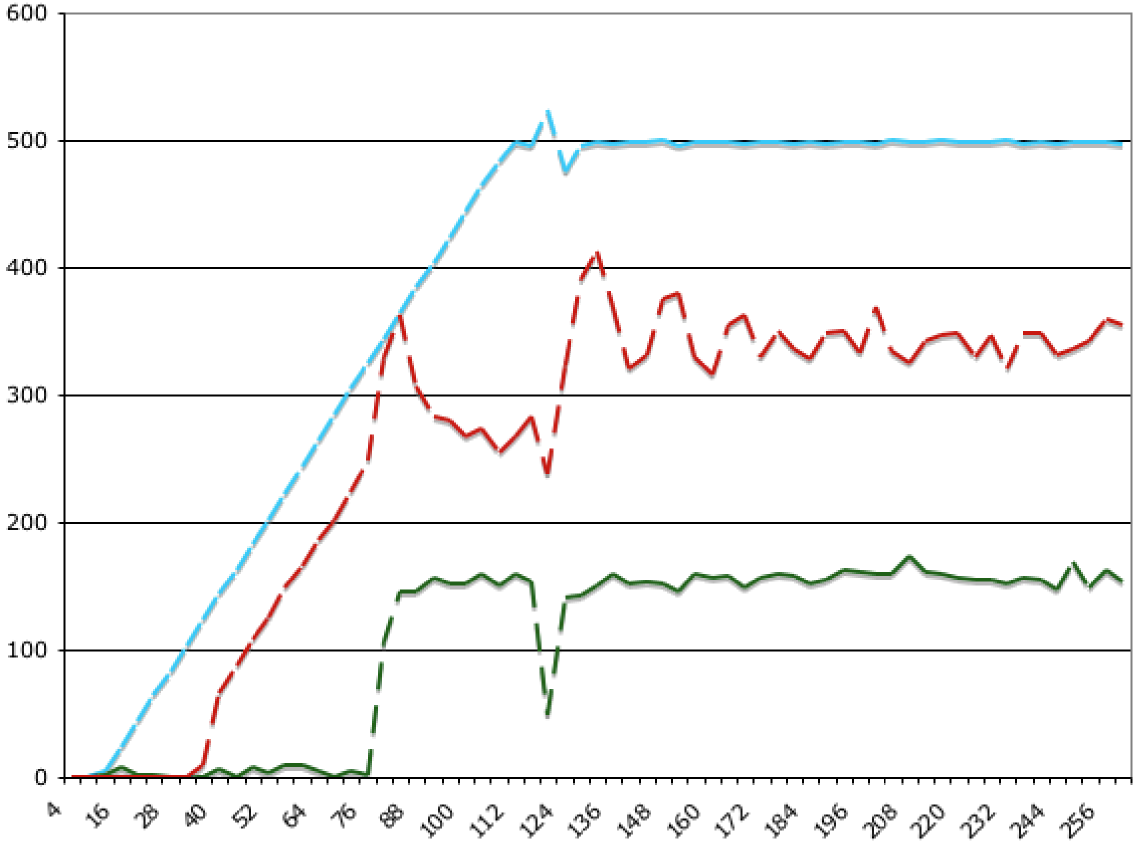
The blue line show attempted requests. The green line shows successful requests.

The agent was configured to disable all programs when the CPU was at or above 80% utilization for at least 60 seconds. Although 20% of the CPU remained idle at this point, it was set as the trigger because no non-threatening activity on the server would drive the CPU to more than 80% for a full minute.

The actual action the agent performs to disable the program not only disables the throttled program, but in fact all the programs leaving only static HTML available through the web server. In this way, the agent mitigates the risk of the two non-throttled programs identified by the developers.

A load of 500 pages/second was chosen to push the server to a high enough CPU load to trigger the agent. Two different classes of pages are involved in this test. The first is a simple static HTML page. This represents the core content of the news site. The second is the program itself.

Almost immediately after starting the test the web server failed to deliver any content, static or dynamic. After approximately 80 seconds the agent disabled the program and static HTML began serving again while requests for dynamic content continued to error.



The blue line shows attempted requests. The red shows errors. The green shows successful requests.

The first test shows that the throttle code, when installed, properly handled moderate load by limiting the number of concurrent requests. The second test shows that the agent was able to disable programs while under high load and restore availability to the static, core content.

Impact on other company applications findings

In developing the report detailing which notreallyclassifieds.com components were in use on high traffic pages it was discovered there are no dynamically generated components served from notreallyclassifieds.com. This finding was confirmed through interviews with the classifieds developers.

After checking that a timeout of six seconds was indeed in place in the configuration of the application server the actual testing of this functionality was postponed until such time as dynamic components are in use. Additionally, the risks associated with providing dynamic components to high traffic sites was added to the classified team's information security guidelines.

The final aspect to testing for impact on the classifieds site during a major news event involved loading shared resources and performing functional tests against notreallyclassifieds.com. Since the news and classifieds sites are hosted on separate hardware they do not compete for CPU, memory, or disk. However, they do contend for access to the shared database as well as bandwidth on the network.

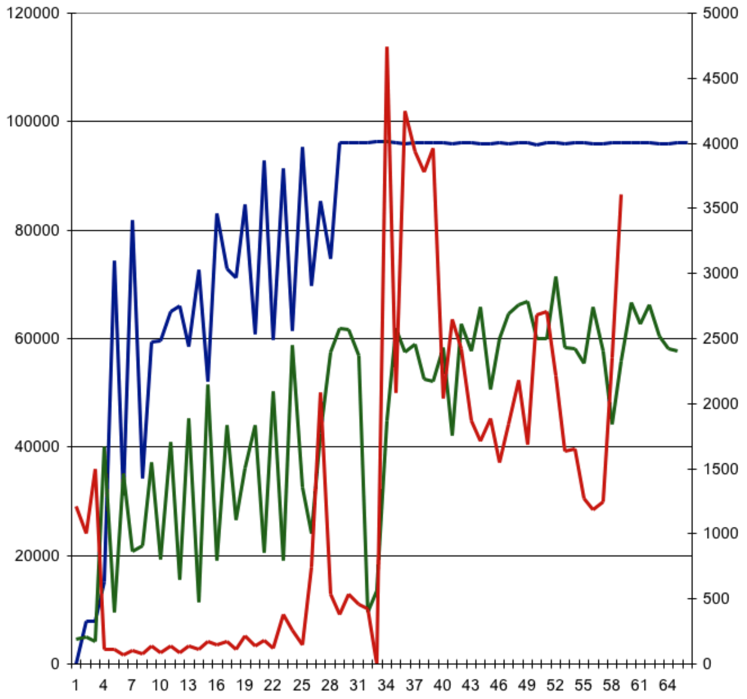
While searching for a way to load the database it was discovered that the news site didn't have the capacity to strain the database. Most of the news site is static and therefore doesn't touch the database. The programs do depend on the database, but attempts to generate enough load on them to stress the database resulted in the agent disabling all the programs and isolating the news site from the database.

Even after the agent was disabled high load on the programs could not affect the database as the programs exhausted the web server's resources before requests could be made to the database server. The conclusion was that the news site could not generate enough load on the database server, under normal or extreme load, to adversely affect the classifieds site.

It was possible, however, to generate high traffic across the network impacting the ability of the classifieds servers to communicate with both the database server and the client.

The scenario in which the network was exhausted is an artificial one. The proxy servers between the end user and the web server make their requests for content with an If-modified-since header. Even though the proxy server checks to see if newer content is available for each request from an end user most of the time the web server sends back a very small response consuming very little bandwidth. The graph below shows the dampening effect a caching proxy server has on bandwidth consumption.

In order to flood the network and evaluate the notreallyclassifieds.com when bandwidth is scarce it is necessary to configure the Avalanche to *not* use the If-modified-since header. Without the special header the web server will send the complete content of the requested page back to the testing appliance.



The blue line represents bandwidth in Mbps and is associated with the left axis. The green line shows successful requests and the red line shows server response time in milliseconds. Both the green and red lines are tied to the axis on the right.

With the network at near 100% utilization wget was used to collect several pages including a static page, a simple search, and a complicated search. These were compared with the same pages retrieved via wget before and after the news site was heavily loaded. (The before and after pages were identical with similar performance so only the comparison between the pre-test and in-test results will be discussed.)

The static page showed no difference in content delivered to the end user between pre- and in-test results. However, the pre-test page was delivered at a rate of over 1 megabyte/second compared to a rate of approximately 264 kilobytes/second. Clearly, congestion on the network dramatically slowed delivery of the page.

The simple search page showed the same delivery slow down, but was otherwise unaffected. Output of diff between the two captured files showed no differences.

The more complex search also showed the same delivery slow down as well as a substantial difference report from diff. By viewing the two files in a web browser the primary difference was immediately clear. Although much of the formatting remained unaffected the search results were replaced by a decidedly user-unfriendly error message from the application server. The error indicated that the web server could not contact the database. Further analysis is required to identify whether a specific component in the network infrastructure caused this error or if it was simply a result of the high network utilization.

Although not specifically what was being tested, the exposure of a database error is in violation of both company policy as well as industry best practices. If a malicious individual were to see the error they could deduce the type of database powering the system and focus their attacks on exploits specific to that type of system.

Conclusions

Over the course of several years significant progress has been made in hardening notarealnewspaper.com. The site routinely handles traffic spikes that resulted in denials of service in the recent past.

Although the proxy:include statements generated by the site's publishing system include the appropriate optional parameters useful in mitigating the risk associated with a traffic spike those coded by hand do not. The poorly coded statements are added outside the publishing system and rules requiring use of the complete syntax cannot be enforced.

Proxy:include statements reference no dynamic components either locally or remotely and only a handful of remotely targeted statements are missing the crucial timeout parameter.

A comprehensive education plan has been put in place to ensure that all members of the notarealnewspaper.com site understand the risks of using proxy:include statements and how to mitigate them.

The throttle code used to limit concurrent instances of programs running on the server effectively handles moderate levels of load. However, the throttle code is not present in all programs on the server resulting in a potential vulnerability at moderate levels of load. The agent that monitors CPU utilization and disables the dynamic portions of the site mitigates this risk for high levels of load.

During traffic spikes to the newspaper site that saturate the internal network the classifieds site experiences problems connecting to the database server. Although this is an artificial scenario it brought to light an error handling issue in a complicated search program. A database error message was exposed to the end user. This information leak could provide valuable assistance to an attacker looking to focus his attacks.

Several areas of additional analysis were identified to fully understand and mitigate the risks exposed in this audit.

Appendix A

Source code for producing a proxy:include report.

```
#!/usr/bin/perl

use LWP::UserAgent;

my ($start) = @ARGV;

$start =~ /^(https?:\/\/[^\/]+)(\/.*)$/;
my $start_host = $1;
my $start_uri = $2;
my $spacer = "";

my %links;

print $start_host . $start_uri . "\n";

build_tree($start_host . $start_uri, $spacer, \%tree);

foreach my $page (sort keys %tree) {
        print " ---\n";
        print "$page\n";

        for (my $i = 0; $i < scalar(@{$tree{$page}}); $i++) {
                my $linked_page = $tree{$page}->[$i];
                if ($linked_page eq $start) {
                        next;
                }
                elsif ($linked_page =~ /^#/) {
                        next;
                }
                elsif ($linked_page =~ /^javascript/i) {
                        print "  HAND ANALYSIS REQUIRED: $linked_page\n";
                        next;
                }
                elsif ($linked_page =~ /^\//) {
                        $linked_page = $start_host . $linked_page;
                }
                print "  link:$linked_page\n";

                build_tree($linked_page, $spacer);
        }
}

exit;


sub build_tree {
        my ($page, $spacer, $hash_ptr) = @_;
        $spacer .= "  ";

        my $content = get_page($page);

        if (! $content) {
                print "error retrieving $page\n";
                return;
        }
        else {
                my @lines = split(/\n/, $content);

                foreach my $line (@lines) {
                        if ($line =~ /.*<proxy:include[^>]+src="?([^"]+)".*>.*/) {
                                my $src = $1;
                                $src =~ s/http:\/\//http:\/\/o./;

                                my $alt;
                                if ($line =~ /.*<proxy:include[^>]+alt="?([^"]+)".*>.*/) {
                                        $alt = $1;
                                }

                                my $timeout;
                                if ($line =~ /.*<proxy:include[^>]+timeout="?([^"]+)".*>.*/) {
                                        $timeout = $1;
                                }

                                my $else;
                                if ($line =~ /.*<proxy:include[^>]+else="?([^"]+)".*>.*/) {
                                        $else= $1;
                                }

                                print $spacer . $spacer . "--" . $src . "\t";
                                print $alt . "\t";
                                print $timeout . "\t";
                                print $else . "\n";
                                build_tree($src, $spacer);
                        }

                        if ($line =~ /.*href="([^"]+)".*/) {
                                push (@{$$hash_ptr{$page}}, $1)
                        }
                }
        }
}       # end crawl_tree


sub get_page {
        # save the uri
        my ($url) = @_;

        # get the data
        my $ua = new LWP::UserAgent;

        # build it into a header object
        my $h = new HTTP::Headers
                Cookie          => $auth_cookie;

        my $request = HTTP::Request->new(GET, $url, $h);
        my $response = $ua->request($request);
        return $response->content;

}       # end get_page
```

References

"Computer System." Internet.com Webopedia. 31 October 2001. 07 March 2005. <http://www.webopedia.com/TERM/C/computer_system.html>

"System." Internet.com Webopedia. 19 February 2003. 07 March 2005. <http://www.webopedia.com/TERM/s/system.html>

Kaplan, Simone. "When Bad Things Happen To Good Companies." CIO. 06 November 2003. 07 March 2005. <http://www.cio.com.au/index.php/id;664166878;fp;4;fpid;18>

SANS Institute, Track 1 - Security Essentials with CISSP CBK. Volume 1.3. SANS Press, April 2003.

"Farked." Urban Dictionary. 30 October 2003. 07 March 2005. <http://farked.urbanup.com/322315>

"Slashdotted." Urban Dictionary. 21 May 2003. 07 March 2005. <http://slashdotted.urbanup.com/134130>

Paessler GmbH Webserver Stress Tool Product Page. 2005. Paessler GmbH. 07 March 2005. <http://www.paessler.com/webstress/?link=menu>

Mercury Interactive LoadRunner Product Page. 2005. Mercury Interactive. 07 March 2005. <http://www.mercury.com/us/products/performance-center/loadrunner/>

Sprient Communications Avalanche Product Page. 2005. Spirent Communications. 07 March 2005. <http://www.spirentcom.com/analysis/product_product.cfm?PL=32&PS=104&PR=521>

GNU Project. "GNU Grep." 04 March 2005. 07 March 2005. <http://www.gnu.org/software/grep/>

GNU utilities for Win32. 30 April 2004. Karl M. Syring. 07 March 2005. <http://unxutils.sourceforge.net/>

Comprehensive Perl Archive Network. 2001. CPAN. 07 March 2005. <http://www.cpan.org>

GNU Project. "GNU wget." 20 February 2005. 07 March 2005. <http://www.gnu.org/software/wget/wget.html>

Heiko Herold's windows wget spot. 2005. Herold, Heiko. 07 March 2005. <http://xoomer.virgilio.it/hherold/>

GNU Project. "GNU Diffutils." 06 November 2003. 07 March 2005.
<http://www.gnu.org/software/diffutils/diffutils.html>

---

[1] "Computer System." Internet.com Webopedia.

[2] "System." Internet.com Webopedia.

[3] Although notarealnewspaper.com does not resolve it represents a real newspaper site.

[4] Kaplan, Simone. "When Bad Things Happen To Good Companies." CIO.

[5] SANS Institute, Track 1 - Security Essentials with CISSP CBK. Volume 1.3. Page 832.

[6] notreallyclassifieds.com does not resolve, but it represents a real classified ad site.

[7] "Farked." Urban Dictionary.

[8] "Slashdotted." Urban Dictionary.

[9] someothernewspaper.com is not a real newspaper site, but could represent any competing online news provider.

[10] Paessler GmbH Webserver Stress Tool Product Page. 2005. Paessler GmbH.

[11] Mercury Interactive LoadRunner Product Page. 2005. Mercury Interactive.

[12] Sprient Communications Avalanche Product Page. 2005. Spirent Communications.

[13] GNU Project. "GNU Grep."

[14] GNU utilities for Win32. 30 April 2004. Karl M. Syring.

[15] This command run with GNU grep 2.4.2 on MacOS X 10.3.8.

[16] Comprehensive Perl Archive Network. 2001. CPAN.

[17] See Appendix A for the code that produced this report.

[18] GNU Project. "GNU wget."

[19] Heiko Herold's windows wget spot. 2005. Herold, Heiko.

[20] This command run with GNU wget 1.8.2 on MacOS X 10.3.8.

[21] GNU Project. "GNU Diffutils."

[22] In order to for the pages to display properly it may be necessary to modify the HTML slightly. In the <head> block adding <base href="http://notreallyclassifieds.com"> will instruct the web browser to request any non-absolute links from the classifieds site rather than the local host.