# Defending Against Application Denial of Service Attacks

Version 1.6
Released: December 20, 2013

**Securosis, L.L.C.** 515 E. Carefree Highway Suite #766 Phoenix, AZ 85085 **T** 602-412-3051
info@securosis.com www.securosis.com

## Author's Note

The content in this report was developed independently of any licensees. It is based on material originally posted on the Securosis blog, but has been enhanced, reviewed, and professionally edited.

Special thanks to Chris Pepper for editing and content support.

## Copyright

# Defending Against Application Denial of Service Attacks

## Table of Contents

# Introduction

As we discussed last year in [Defending Against Denial of Service Attacks](https://securosis.com/research/publication/defending-against-denial-of-service-dos-attacks)[1], attackers increasingly leverage availability-impacting attacks, both to cause downtime (which costs site owners money) and to mask other kinds of attacks — usually involving data theft. These availability-impacting attacks are better known as Denial of Service (DoS) attacks. Our research has identified a number of adversaries who increasingly use DoS attacks, including:

- Protection Racketeers: These criminals use DoS threats to demand ransom money. Attackers hold a site hostage by threatening to knock it down, and sometimes follow through. They get paid. They move on to the next target. The only thing missing is the GoodFellas theme music.

- Hacktivists: DoS has become a favored approach of hacktivists seeking to make a statement and shine a spotlight on their cause, whatever it may be. Hacktivists care less about the target than the cause. The target is usually collateral damage, though they are happy to hit two birds with one stone by attacking an organization that opposes their cause when they can. You cannot negotiate with these folks, and public discourse is one of their goals.

- 'CyberWar': We don't like the term — no one has been killed by browsing online (yet), but we can expect to see online attacks as precursors to warplanes, ships, bombing, and ground forces. By knocking out power grids, defense radar, major media, and other critical technology infrastructure, the impact of an attack could be magnified.

- Exfiltrators: These folks use DoS to delay the response by tying up incident responders, who could have an impact on their true goal — data theft. This could be an intellectual property theft or a financial attack such as stealing credit cards. This tactic is also helpful in the event the application is built with logic to scale back input validation and fraud checks when under significant load. So hammering the application makes it easier to attack. Win-win.

Attackers increasingly leverage availability-impacting attacks, both to cause downtime (which costs site owners money) and to mask other kinds of attacks — usually involving data theft.

[1] https://securosis.com/research/publication/defending-against-denial-of-service-dos-attacks

- Competitors: They say all's fair in love and business. Some folks take that one a bit too far, and actively knock down competitor sites for their own advantage. Maybe it's during the holiday season. Maybe it happens after a competitor resists an acquisition or merger offer. It could be locking folks out from bidding on an auction. Your competition might screen scrape your online store to make sure they beat your pricing, causing a flood of traffic on a very regular and predictable basis. A competitor might try to ruin your hard-earned (and expensive) search rankings. Regardless of the reason, don't assume an attacker is a nameless, faceless crime syndicate in a remote nation. It could be the dude down the street trying to get any advantage he can — legalities be damned.

Given the varied adversaries, it is essential to understand that two totally different types of attacks are commonly lumped under the generic 'DoS' label. The first involves the network, blasting a site with enough traffic (sometimes over 300gbps) to flood the pipes and overwhelm security and network devices, as well as application infrastructure. This volumetric attack basically is the 'cyber' version of hitting something a billion times with a rock. This brute force attack typically requires a scrubbing service and/or CDN (Content Delivery Network) to deal with the onslaught of traffic and keep sites available.

The second type of DoS attack targets weaknesses in applications. In Defending Against DoS we described application attacks as follows:

*Application-based attacks are different — they target weaknesses in web application components to consume all the resources of a web, application, or database server to effectively disable it. These attacks can target either vulnerabilities or 'features' of an application stack to overwhelm servers and prevent legitimate traffic from accessing web pages or completing transactions.*

These attacks require knowledge of the application and how to break or game it. They can be far more efficient than just blasting traffic at a network, requiring far fewer attack nodes and less bandwidth. A side benefit of requiring few nodes is simplified command and control, allowing more agility in adding new application attacks. Moreover, the attackers often take advantage of legitimate application features, making defense considerably harder.

This paper will dig into application DoS attacks, offering both an overview of common attack tactics and possible mitigations for each. By the end of the paper you should have a guide and an understanding of how your application availability will be attacked, and what you can do about it.

# Attacking the Application Server

Application denial of service can be harder to deal with than volume-based network DDoS because it is not always obvious what is an attack rather than legitimate application traffic. Unless you are running all your traffic through a scrubbing center all the time, your applications will remain targets for attacks that exploit their architecture, application stacks, business logic, and even legitimate application functionality.

Application stack attacks are the low-hanging fruit of application denial of service, and will remain so as long as they continue to work.

We will start with AppDoS attacks that target the server and infrastructure of your application, what we call the application stack. Given the popularity and proliferation of common application stacks, attackers can hit millions of sites with a standard technique, many of which have been in use for years. But not enough web sites have proper mitigations in place. Go figure. Application stack attacks are the low-hanging fruit of application denial of service, and will remain so as long as they continue to work.

## Exploiting the Server

Most attacks that directly exploit web servers capitalize on features of the underlying standards and/or protocols that run the web, such as HTTP. This makes many of these attacks very hard to detect because they look like legitimate requests — by the time you figure out it's an attack your application is down. Here are a few representative attack types:

- Slowloris: This attack, originally demonstrated by Robert 'RSnake' Hansen, knocks down servers by slowly delivering request headers, forcing the web server to keep connections open, without ever completing the requests. This rapidly exhausts the server's connection pool.
- Apache Killer: This attack exploits how the Apache web server platform handles HTTP range requests. If multiple GET requests contain overlapping address range fields, the server consumes all its memory loading a new copy of the object for each range, and the server then goes down. The patch has been available for a few years now, but many web sites still run old, vulnerable versions of Apache.

- Slow HTTP Post: Similar to Slowloris, Slow HTTP Post delivers the message body slowly. This is another way to exhaust web server resources. Both Slowloris and Slow HTTP Post are difficult to detect because their requests look legitimate — they just never complete. The R-U-Dead-Yet (RUDY) attack tool automates launching Slow HTTP Post attacks via an automated user interface. To make knocking down vulnerable web servers easier, RUDY is included in many penetration testing tool packages.
- Slow Read: Yet another variation of the Slowloris approach, Slow HTTP Read involves shrinking the response window on the client side. This forces the server to send data to the client slowly to stay within the response window. The server must keep connections open to ensure the data is sent, which means it can be quickly overwhelmed with connections.

These techniques are already weaponized and available for easy download and usage. We can expect innovative attackers to combine and automate these tactics into weapons of website destruction. The hacker th3j35t3r's XerXeS DoS tool[2] is one of the first examples of a number of these techniques stacked together to increase the effectiveness of each tactic. Regardless of the packaging, these tactics are real and must be defended against.

Mitigating these attacks typically requires a combination of web server configuration, network-based defenses, and application-based defenses.

Mitigating these attacks typically requires a combination of web server configuration, network-based defenses, and application-based defenses. It is impossible to completely defend applications against these attacks because they take advantage of web server protocols and architecture. But you can blunt their impact with appropriate controls.

For example, Slowloris and Slow HTTP Post require tuning the web server to increase the maximum number of connections, prevent excessive connections from the same IP address, and allow a storage of a backlog of connection requests — to avoid losing legitimate application traffic. Network-based defenses on WAF and IPS devices can be tuned to look for certain web connection patterns and block offending traffic before the server becomes overwhelmed. The best approach is actually all of the above. Don't just tune the web server or install network-based protection in front of the application — also build web applications to limit header and body sizes, and to close connections within a reasonable timeframe to avoid exhausting the connection pool. We will discuss how to build AppDoS protections into applications later in this paper.

Fortunately recent versions of Apache are protected against Apache Killer. You can also limit the number of address ranges allowed within a request to prevent thrashing on overlapping ranges.

[2] www.infosecisland.com/blogview/9865-The-Jester-Hits-WikiLeaks-Site-With-XerXeS-DoS-Attack.html

An attack like Slow HTTP Read games the client side of the connection and requires similar mitigations. But instead of looking for ingress patterns of slow activity (on either the web server or other network devices), you need to scan the egress side. Likewise, fronting web applications with a WPS (Website Protection Service) can alleviate some of these attacks as the WPS insulates your web application server from clients and slow reads. For more on these services consult our paper on Quick Wins with Website Protection Services[3].

## Brute Force

Another tactic is to overwhelm the application server — not with network traffic, but by overloading application features. We will cover one aspect of this when we discuss search engine and shopping cart shenanigans. For now let's look at more basic features of pretty much every website, such as SSL handshakes and common web pages including the login screen, password reset form, and store locator.

These attacks are effective at overwhelming application servers because SSL handshakes and pages which require database calls are very compute intensive. Loading a static page is easy, but checking login credentials against a database of hashed passwords is much more demanding.

If an attacker uses a rented botnet to establish a million or so SSL sessions at the same time, guess what happens?

First let's consider the challenges of scaling SSL. On some pages, such as the login page, you need to encrypt traffic to protect user credentials in motion. SSL is a requirement for such pages. So why is scaling SSL handshaking such an issue? As described succinctly in Microsoft's description of SSL[4], there are 9 distinct steps in establishing a SSL handshake, many of which require deliberately complex cryptographic and key generation operations.

If an attacker uses a rented botnet to establish a million or so SSL sessions at the same time, guess what happens? It is a compute problem rather than a bandwidth issue, but the application becomes unavailable because no more SSL connections can be established. Mitigating an SSL handshake attack involves either offloading SSL termination to devices specifically designed to terminate SSL connections, or having a website protection service terminate the connections for you and then sending the traffic back to your site over a single (low-overhead) encrypted pipe.

In contrast, a database-driven page such as a login form or store locator needs to hit the database for every request. That typically involves 2-10 different database transactions, as well as field-level validation and other techniques for defending against cross-site scripting (XSS), CSRF (cross-site request forgery), and other application-layer attacks.

[3] https://securosis.com/research/publication/quick-wins-with-website-protection-services

[4] support.microsoft.com/kb/257591

Login page attacks are problematic — not just because attackers can use random numbers to waste effort on the server side. They can also submit huge passwords to force compute-intensive hashing functions to check absurd passwords. A side benefit (for attackers) is that legitimate users can be locked out by too many password failures. To get more specific on the store locator attack, an adversary could submit random 5-digit numbers in the zip code field, forcing the application to search the entire zip code database before giving up on each request. Finally, the attacker could flood the database with garbage by scripting the 'create account' function to generate random usernames and passwords. There are many different ways to waste resources on legitimate pages.

How can you defend against these kinds of attacks? Similar to the way you deal with HTTP-based attacks like Slowloris and Slow HTTP: use defenses built into the application, network-based defenses, and/or website protection services.

To dig into the application defenses, it's important to do input validation on the size of passwords and search terms entered into the application. You need to meter the number of requests/transactions from specific IP addresses to those demanding pages. The good news is that many WAFs (as deployed) have these defenses built in, but doing significant input validation and request metering can increase latency for visitors trying to use them during high usage periods. But the alternative is to have your site crater. We'll discuss application-specific defenses later in the paper.

# Attacking the Application Stack

If you think of a web application as an onion, there always seems to be another layer you can peel back to expose more attack surface.

We have already discussed ways attackers target standard web servers, protocols, and common pages to impact application availability. These surface-level attacks are low-hanging fruit because they can be executed via widely available tools wielded by unsophisticated attackers. If you think of a web application as an onion, there always seems to be another layer you can peel back to expose more attack surface.

The next layer is the underlying application stack used to build the application. One of the great things about web applications is the availability of fully assembled technology stacks, making it trivial to roll out infrastructure to support a wide variety of applications. But anything widely available inevitably becomes an attack target. The best example of this, in the context of availability attacks, is exploitation of hash tables to crush web servers.

## Hash Collision Attacks

We won't get into advanced programming, but you need some context to understand this attack. A hash table[5] is used to map keys to values by assigning each key to a specific slot in an array. This enables very fast searching for keys. But multiple values can 'collide' and be assigned the same slot, which requires more complicated handling and significant additional processing. Hash collisions are normally minimized so the speed trade-off is usually worthwhile.

But if an attacker understands the hashing function used by the application, they can cause excessive hash collisions. This requires the application to compensate and consume extra resources to manage the hashing function. If enough hash collisions occur… you guessed it: the application can't handle the workload and goes down.

This attack was weaponized in HashDoS[6], an attack tool that leverages the fact that most web application stacks use the same hashing algorithm, so a generic attack tool works against most web

[5] en.wikipedia.org/wiki/Hash_table

[6] http://www.purehacking.com/blog/josh-zlatin/introduction-to-hash-dos-attacks

applications on the Internet. With knowledge of this hashing algorithm an attacker can send a POST request with many variables to create hash table chaos and render the application useless. Mitigation for this attack requires discarding messages with too many variables — typically implemented within a WAF (web application firewall) — or randomizing the hash function with application-layer logic. One good explanation of this attack[7] uses cats to demonstrate hashing attacks in layperson's terms.

Remember that any capability of the application stack can be exploited. So diligence in selecting a stack, ensuring secure implementation, and tracking security notices and implementing patches, is critical to application security and availability.

## Targeting the Database

As part of the application stack, databases tend to get overlooked as denial of service attack targets. Many attackers try to extract the data from the database and then exfiltrate it, in which case knocking down the database would be counterproductive. But when the mission is to impact application availability or to use DoS as cover for exfiltration, the database can be a soft target — because in some way, shape, or form, the web application depends on the database.

DoS attacks can be divided into network-based volumetric attacks and application-layer attacks. The problem is that databases can be attacked **both** ways.

If you recall our Defending Against Denial of Service Attacks[8] paper, we divided DoS attacks into network-based volumetric attacks and application-layer attacks. The problem is that databases can be attacked both ways. Application servers connect to the database using some kind of network, so a volume attack on that network segment can impact availability. If the database itself is exploited the application is also likely to go down. Either way the application is out of business.

### Database DoS Attacks

If we dig a bit deeper we see that one path is to crush databases using deliberately wasteful queries. Other attacks target simple vulnerabilities that have never been patched, mostly because the demand for uptime interferes with patching so that it takes place sporadically, if at all. Again, you don't need to be a brain surgeon to knock a web application offline. Here are some of the attack categories:

1. **Abuse of Functions:** This type of attack is similar to the slow HTTP attacks mentioned above: attackers use database functionality against the server. For example, if you restrict failed logins, they might blast your database with bad password requests to lock legitimate users or applications out.

[7] www.anchor.com.au/blog/2012/12/how-to-explain-hash-dos-to-your-parents-by-using-cats/

[8] https://securosis.com/research/publication/defending-against-denial-of-service-dos-attacks

2. **Complex Queries:** If the attacker gives the database too much work to do it will fail. There are many ways to do this, including nested queries and recursion, Cartesian joins, and the `in` operator, each of which can overwhelm the database. The attacker must be able to inject a SQL query into the database directly or from within the application for this to work, so you can block these attacks that way. We will discuss defenses further below.

3. **Bugs and Defects:** In these cases the attacker targets a known database vulnerability. This includes queries of death and buffer overflows. With new functionality introduced constantly, database attack surface continues to grow. Even if the database vendor identifies the issue and produces a patch (not a sure thing), finding a maintenance window to apply it remains challenging in many environments.

4. **Application Usage:** Finally, the way the application uses the database can be gamed to cause an outage. The best example of this is SQL injection, but that attack is rarely used just to knock over databases. Also consider login and store locator page attacks, as well as shopping cart and search engine attacks (to be covered later) as additional examples of application misuse that can impact availability.

Consider the flexibility an attacker has when using SQL injection to directly attack a database. They could use injection to circumvent query limits enforced by input validation and subsequently request a huge result set to knock down the database. Similarly, an attacker could game the speed of MySQL's query benchmark capability by injecting a request to perform a specific function tens of thousands of times, all while asking for a benchmark of the average time of the request. This is another way to crush database performance by not only overloading the database with busywork, but asking the database exactly how long the busywork takes.

## Database DoS Defenses

The tactics used to defend against database denial of service attacks closely reflect good database security practices. Go figure.

The tactics used to defend against database denial of service attacks closely reflect good database security practices.
Go figure.

- **Configuration:** Strong database configuration processes include removing unneeded functions, user accounts, protocols, and services. You will be best served by minimizing database attack surface as a matter of practice.

- **Resource Limits:** You can limit the database resources any specific user, process, or query can consume in order to blunt attacks. Unfortunately resource limits can also be used against you to starve legitimate users and queries so this defense is a double-edged sword.

- **Patching:** As mentioned above, DBAs are constrained in fixing database bugs and defects by the availability of vendor-issued patches. Given that enterprises tend to patch only every 14 months or so, availability of a patch doesn't mean you are protected. We understand the need for maximum uptime and minimal operational availability windows, but if the database is knocked down via a known vulnerability everyone loses — except the attacker.

- **Database Activity Monitoring (DAM):** Combining a number of functions, including configuration checking and SQL query monitoring, DAM devices scrutinize every request and can alert on database misuse. But an alert is not very useful when your database is under attack, so to defend against DoS you will want to configure DAM devices to *block* definite attacks.
- **Database Firewalls:** Like a network firewall, a database firewall only allows authorized queries to reach the database, thus blocking many attacks. If you can reliably profile authorized queries this approach makes a lot of sense. But that's harder than it sounds in light of the rapid change inherent to web applications.
- **Web Application Firewalls (WAF):** WAF devices are more generic than database firewalls, focusing on authorized application functionality and blocking known attacks, but they offer a similar function — indirectly protecting databases from attack. Note that WAF is ineffective against attacks that misuse legitimate database or application functionality. WAFs can also be a target of a denial of service attack, since they require computing cycles to do the input validation and enforce the policies. The more rules running on the WAF, the higher latency and more likelihood the WAF will become the bottleneck when under attack.
- **Application and Database Hardening:** Another option is to protect the application stack from misuse by building protections directly into application code, including the logic used to access the database. We will discuss this in detail later.
- **Website Protection Services:** Finally, keep in mind that many website protection services offer WAF-like functionality, and can block or slow down application attacks that impact the database.

For detail on these kinds of database-centric attacks (and appropriate countermeasures), see our Dealing with Database Denial of Service[9] paper.

[9] https://securosis.com/research/publication/dealing-with-database-denial-of-service

# Abusing Application Logic

Now let's turn our attention to attacking the application itself. Every application contains weaknesses that can be exploited, especially when the goal is simply to knock the application offline rather than something more complicated — such as stealing credentials or gaining access to the data for eventual exfiltration. That lower bar of taking an application offline means more places to attack.

Let's bust out the kill chain[10] to illuminate attack progression and start at the beginning: reconnaissance. That is where the process starts for application denial of service as well. Attackers need to find weak points in the application, so they assess it to figure out which pages consume the most resources, the kinds of field-level validation on forms, and the supported attributes for query strings.

For instance, if a web page does a ton of field-level validation or needs to make multiple calls to multiple sites to render the page it is a good target to blast. Serving dynamic content requires a bunch of database calls to populate a page, and each consumes resources. The goal is to be as demanding as possible, in order to impair the application's ability to serve legitimate traffic.

## Flooding the Application

Above we talked about how network-based attacks flood the pipes. Targeting resource intensive pages with either `GET` or `POST` requests (or both) provides an equivalent application flooding attack, exhausting the server's session and memory capacity. Attackers flood a number of different parts of web applications, including:

- **Top-level index page:** This one is straightforward and usually has the fewest protections because it is open to everyone. Blasted by tens of thousands of clients simultaneously, the server can become overwhelmed.

Targeting resource intensive pages with either `GET` or `POST` requests (or both) provides an equivalent application flooding attack, exhausting the server's session and memory capacity.

[10] computer-forensics.sans.org/blog/2009/10/14/security-intelligence-attacking-the-kill-chain/

- **Query string “proxy/cache busting”:** Attackers can send a request to bypass any proxy or cache, forcing the application to handle the request, doing the required processing and then sending new information to the requestor. This serves to weaken the benefit of a cache in front of the application. The impact can be particularly acute when requesting large PDFs or other files repeatedly, consuming excessive bandwidth and server resources.

- **Random session cookies/tokens:** By establishing thousands of sessions with the application, attackers can overload the server’s session tables and impact its ability to serve legitimate traffic.

Flood attacks can be detected rather easily, unlike the slow attacks described above, providing an opportunity to rate-limit attacks while allowing legitimate traffic through. Of course this approach demands high accuracy, as false positives slow down or discard legitimate traffic, and false negatives allow attacks to waste server resources.

To accurately detect application floods you need a detailed baseline of legitimate traffic, tracking details such as URL distribution, request frequency, maximum requests per device, and outbound traffic rates. With this data a legitimate application behavior profile can be developed. You can then compare incoming traffic (usually on a WAF or anti-DoS appliance) against the profile to identify bad traffic, and then limit or block the bad stuff.

Another tactic to mitigate application floods is input validation on all form fields, to ensure requests neither overflow application buffers nor misuse application resources. If you are using a CDN to front-end your application, make sure it can handle random query string attacks and that you are benefiting from the caching service. Some attackers can bypass CDNs (assuming you have one), so you will want to ensure your input validation ignores random query strings.

You can also leverage IP reputation services to identify bot traffic and limit or block it. That requires coordination between application and network defenses, but is effective for detecting and limiting floods.

## Pagination

A pagination attack involves requesting an unreasonable number of results by expanding the PageSize query parameter beyond intended limits. This can return tens of thousands or even millions of records, consuming substantial database resources — especially when serving multiple requests at the same time. These attacks are typically launched against the search page, using a standard dictionary or even random alphanumerics. Either way, the search function is hammered scanning the entire site.

Another tactic for overwhelming applications is to use a web scraper to capture information from dynamic content areas such as store locators and product catalogs. Product search offerings like Baidu and Yandex have been known to be very aggressive scrapers. If those types of sites and other scrapers are not throttled it can overwhelm the application by scraping over and over again.

Mitigation for most pagination attacks must be built into the application. For example, regardless of the PageSize parameter, the application should limit the number of records returned. Likewise, you will want to limit the number of search requests the site will process simultaneously. You can also leverage a CDN or web protection service to cache static information and limit search activity. Alternatively, embedding complicated JavaScript on search pages can deter bots.

## Gaming the Shopping Cart

Another frequently exploited legitimate feature is the shopping cart. An attacker might put a few items in a cart and then abandon it for a few hours. At some point they can come back and refresh the cart, putting off session expiration and forcing the database to reload the cart. If the attacker has tens of thousand of products into the cart this consumes substantial resources.

Shopping cart mitigations include limiting the number of items that can be added to a cart and periodically clearing out carts with too many items. You will also want to periodically terminate old carts to reclaim session space and flush abandoned carts.

## Combination Platter

Attackers are smart. They have figured out that they can combine attacks to amplify their impact.

Attackers are smart. They have figured out that they can combine attacks to amplify their impact without the associated increase in required attack nodes due to the efficiency of application DoS. An attacker might launch a volume-based network attack on a site. Then start a `GET` flood against legitimate pages, rate-limited to avoid being obvious. This combination should slow response time to a trickle, but that's not all. They can follow up with a slow HTTP attack to further consume application resources. Finally they might attack the shopping cart or store locator, masquerading as legitimate customers. All of these attacks can be automated, making the execution of a multi-phased attack rather straightforward.

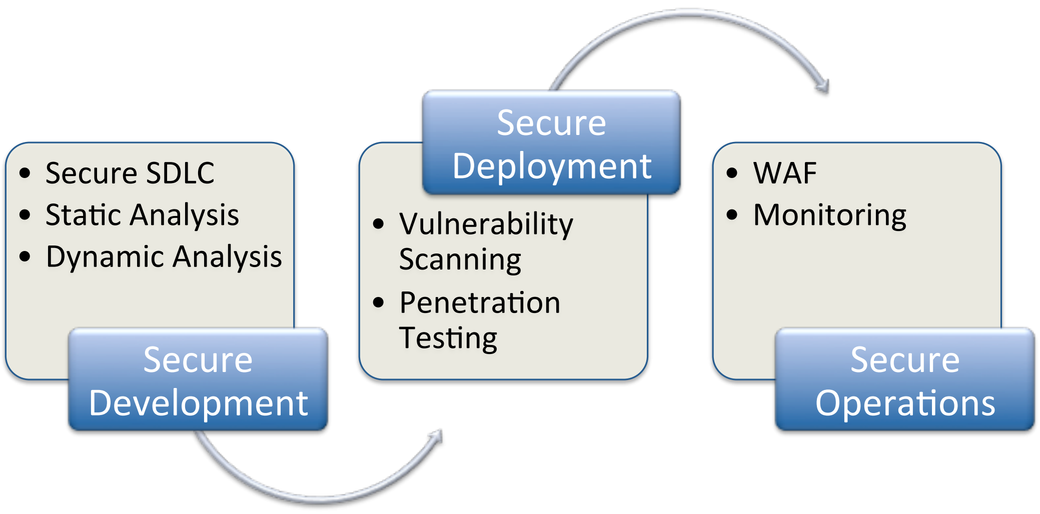
It is hard enough to defend against one of these attack vectors. Combining a number of different attacks dramatically reduces most organizations' ability to defend themselves. Mitigations require the defenses we have been discussing throughout this paper to come together in a coordinated whole. First your scrubbing center needs to blunt the impact of the network attack. Then your on-premise application protection (either a WAF or purpose-built anti-AppDoS device) needs to accurately profile and limit the `GET` flood. Then your web server needs to handle the slow HTTP attack by terminating connections that last too long. Finally application logic must close shopping carts that compromise the PageSize and dismiss stale carts. If any of these defenses fail the application might go down. It is a tightrope act, all day, every day, for applications targeted by DoS.

# Building In Protection

We understand coordinating defenses at multiple levels is a significant undertaking. For instance, security folks have been trying to get developers on board for years, to build security into applications… with little success so far. The operations folks can be difficult at times, as well. That doesn't mean you shouldn't keep pushing, especially given the relative ease of knocking down an application without proper internal defenses. We have found the best way to get everyone on board is by implementing a structured web application security program that looks at each application in its entirety, and can be extended to add protection against denial of service attacks.

Security folks have been trying to get developers on board for years, to build security into applications… with little success so far.

### Web Application Security Process



Revisiting the process described in Building a Web Application Security Program[11], web applications need to be protected across the entire lifecycle:

1. **Secure Development:** You start the process by building security into the software development lifecycle (SDLC). This includes training for people who deliver web applications, along with improved processes to guide their efforts. Security awareness training for developers is managed through education and supportive process modifications, as a precursor to making security a functional application requirement. This phase of the process involves threat modeling up front to identify the pages with the most logic (and thus longest load times), which would warrant additional DoS defenses. The SDLC also leverages tools to automate portions of the effort: static analysis to help engineering identify vulnerable code and dynamic analysis to detect anomalous application behavior.

2. **Secure Deployment:** Once an application is code complete and ready for more rigorous testing and validation, it is time to confirm that it doesn't suffer from serious known security flaws (vulnerabilities) and is configured securely, so it is not subject to any known compromises. This is where you use vulnerability assessments and penetration testing — along with solid operational approaches to configuration analysis, threat discovery, patch levels, and operational consistency checking.

3. **Secure Operations:** The last phase of the process moves from preventative tools and processes to detecting and reacting to events from production applications. Here you deploy technologies and/or services to front-end the application, including web application firewalls and web protection services.

To consider the specifics of what is required to deal with AppDoS attacks, let's look at each step in the process.

The most effective protections are input validation on form fields to mitigate against buffer overflow, code injection, and other attacks that can break application logic.

## Secure Development

In this phase we are looking to build the protections we have been discussing into application(s). This involves making sure the application stack in use is insulated against HashDoS attacks and no database calls present opportunity for misuse or excessive queries. The most effective protections are input validation on form fields to mitigate against buffer overflow, code injection, and other attacks that can break application logic.

[11] https://securosis.com/Research/Publication/web-application-security-program

For example, the recent attack on the Django web development framework[12] involved blasting the login process with very long passwords (millions of characters). This forced the application to hash each long password (which was very processor intensive) and then compare the hash to an existing password. This simple attack was possible because Django's password fields lacked input validation.

But input validation can be a double-edged sword. Heavy input validation impacts application performance, increasing CPU usage and making it easier to attack those pages with a `GET`/`POST` flood or similar attack. Prioritize validating fields that require the least computational resources, and/or re-architect the application to offload some processing requirements to improve scalability. As always you need to balance protection against performance when stress testing the application prior to deployment, to ensure survivability.

Additionally there are tactics, like CAPTCHA and browser validation, that can identify automated attackers and block their requests. These can work well, but can also exacerbate the attack since these challenge/request tactics usually multiple the number of requests involved per session. So if the attacker can handle this defense it accelerates the impact of the attack and takes your site down faster.

You can also add shorter timeouts to database queries and application logic. This could adversely impact normal application behavior, but will also stop many of the "slow" application attacks. Also ensure your application security testing (both static and dynamic) tests robustness against denial of service attacks, including shopping cart and pagination attacks.

## Secure Deployment

When deploying the application make sure the stack incorporates protections against common web server DoS attacks including Slowloris, slow HTTP, and Apache Killer. You can check for these vulnerabilities using an application scanner or during a penetration test. Keep in mind that you will likely need some tuning to find the optimal timeout for session termination.

If you can identify nodes being used in application DoS attacks then you can signal into the network to block those nodes upstream by working with your network, CDN, or scrubbing provider.

## Secure Operations

Once the application goes into production the fun begins — you will be working with live ammunition. You can deploy an anti-DoS appliance or service, or a WAF (either product or service) to rate limit slow HTTP attacks. This is also where a CDN or web protection service comes into play absorbing high-bandwidth attacks and intelligently filtering and caching content to blunt the impact of attacks, including the random query string attacks described above.

[12] arstechnica.com/security/2013/09/long-passwords-are-good-but-too-much-length-can-be-bad-for-security/

One of the key differences between an application DoS and a network DoS attack is that application attacks need to use the TCP protocol to send/receive HTTP requests and that means they can't spoof the sender as in SYN or UDP floods. Thus, if you can identify nodes being used in application DoS attacks then you can signal into the network to block those nodes upstream by working with your network, CDN, or scrubbing provider. This conserves application resources that would otherwise be consumed dealing with the attack.

Finally, during the operational phase, you will monitor the performance and responsiveness of the application, as well as track inbound traffic to detect emerging DoS attacks as early as possible. Since you developed profiles for normal application behavior earlier in the process — you can now use them to identify attack traffic before you have an outage. This may involve using a 3rd party web performance monitoring solution and setting the alerts to possibly indicate an emerging DoS attack.

## Finding the Right Mix

As we have described, you have several options to defend applications against denial of service attacks — so how can you determine the right mix of cloud-based, server-based, and application-based protections? You need to think about each in terms of effort and agility required to deploy at each level.

Building protections into applications doesn't happen overnight — it is likely to require development process changes and a development cycle or three to implement proper security controls. The application may also require significant re-architecting — especially if its database-driven aspects have not been optimized. Keep in mind that new attacks and newly discovered vulnerabilities require you to revisit application security on an ongoing basis. Like other security disciplines, you never really finish securing your application.

Somewhat less disruptive is hardening the application stack, including the web server, APIs, and database. This tends to be an operational responsibility so you need to collaborate with the ops team to ensure the right protections, patches, and configurations are deployed on the servers.

Finally, the quickest path to protection is to front-end your application with an anti-DoS device and/or a cloud-based website protection service to deal with flood attacks and simple application attacks. As we mentioned, these defenses are not a panacea — you still need to harden the stack and protect the application as well. But deploying an inline device or service as a first line of defense does not depend on any other part of the organization, so it is usually the quickest and easiest protection to get operational.

We'd be remiss not to mention the benefit of testing how the application handles an AppDoS attack by attacking yourself. Yes, with real traffic aimed at taking down your own site. Though you probably should be careful about using the production environment, blowing up a staging or test site should be within acceptable boundaries. There are tools and services available to DoS your test environment to see how your defenses hold up. Besides showing you how your defenses hold up under attack, this approach also allows your response team to practice ensuring you are better prepared for the real attack.

# Summary

Application Denial of Service attacks introduce another level of complexity to keeping critical web application available and operational. Not only can adversaries exploit weaknesses in application stack components (web, application, and database servers) to knock infrastructure over, but the application itself can be targeted — both by overwhelming inefficiently coded pages and using legitimate capabilities such as search and login.

Defending against AppDoS requires a multi-faceted approach that typically starts with a mechanism to filter attack traffic, either via a web protection service running in the cloud or an on-premise anti-DoS device. The next layer of defense includes operational measures to ensure the application stack is hardened, including timely patching and secure configuration of components. Finally, developers must play their part by optimizing database queries and providing sufficient input validation to make sure the application itself cannot be overwhelmed using legitimate capabilities.

Keeping applications up and running requires significant collaboration between development, operations, and security. This ensures not only that sufficient defenses are in place, but also that a well-orchestrated response maintains and/or restores service as quickly as possible. It is not a matter of *if*, but *when* you are targeted by an Application Denial of Service attack. We hope our research helps you make sure you are ready.

If you have any questions on this topic, or want to discuss your situation specifically, feel free to send us a note at info@securosis.com or ask via the Securosis Nexus <http://nexus.securosis.com/>.

# About the Analyst

**Mike Rothman, Analyst/President**

Mike’s bold perspectives and irreverent style are invaluable as companies determine effective strategies to grapple with the dynamic security threatscape. Mike specializes in the sexy aspects of security — such as protecting networks and endpoints, security management, and compliance. Mike is one of the most sought-after speakers and commentators in the security business, and brings a deep background in information security. After 20 years in and around security, he’s one of the guys who “knows where the bodies are buried” in the space.

Starting his career as a programmer and networking consultant, Mike joined META Group in 1993 and spearheaded META’s initial foray into information security research. Mike left META in 1998 to found SHYM Technology, a pioneer in the PKI software market, and then held executive roles at CipherTrust and TruSecure. After getting fed up with vendor life, Mike started Security Incite in 2006 to provide a voice of reason in an over-hyped yet underwhelming security industry. After taking a short detour as Senior VP, Strategy at eIQnetworks to chase shiny objects in security and compliance management, Mike joined Securosis with a rejuvenated cynicism about the state of security and what it takes to survive as a security professional.

Mike published The Pragmatic CSO <http://www.pragmaticcso.com/> in 2007 to introduce technically oriented security professionals to the nuances of what is required to be a senior security professional. He also possesses a very expensive engineering degree in Operations Research and Industrial Engineering from Cornell University. His folks are overjoyed that he uses literally zero percent of his education on a daily basis. He can be reached at mrothman (at) securosis (dot) com.

# About Securosis

Securosis, LLC is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services. Our services include:

- **The Securosis Nexus**: The Securosis Nexus is an online environment to help you get your job done better and faster. It provides pragmatic research on security topics that tells you exactly what you need to know, backed with industry-leading expert advice to answer your questions. The Nexus was designed to be fast and easy to use, and to get you the information you need as quickly as possible. Access it at <https://nexus.securosis.com/>.

- **Primary research publishing**: We currently release the vast majority of our research for free through our blog, and archive it in our Research Library. Most of these research documents can be sponsored for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements and conform to our Totally Transparent Research policy.

- **Research products and strategic advisory services for end users**: Securosis will be introducing a line of research products and inquiry-based subscription services designed to assist end user organizations in accelerating project and program success. Additional advisory projects are also available, including product selection assistance, technology and architecture strategy, education, security management evaluations, and risk assessment.

- **Retainer services for vendors**: Although we will accept briefings from anyone, some vendors opt for a tighter, ongoing relationship. We offer a number of flexible retainer packages. Services available as part of a retainer package include market and product analysis and strategy, technology guidance, product evaluation, and merger and acquisition assessment. Even with paid clients, we maintain our strict objectivity and confidentiality requirements. More information on our retainer services (PDF) is available.

- **External speaking and editorial**: Securosis analysts frequently speak at industry events, give online presentations, and write and/or speak for a variety of publications and media.

- **Other expert services**: Securosis analysts are available for other services as well, including Strategic Advisory Days, Strategy Consulting engagements, and Investor Services. These tend to be customized to meet a client's particular requirements.

Our clients range from stealth startups to some of the best known technology vendors and end users. Clients include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors.

Additionally, Securosis partners with security testing labs to provide unique product evaluations that combine in-depth technical analysis with high-level product, architecture, and market analysis. For more information about Securosis, visit our website: <http://securosis.com/>.