

# Web Application DoS/DDoS Testing Methodology

*Gunter Ollman  
Chief Technology Officer  
IOActive<sup>®</sup>, Inc.*

## Abstract

As the goals and motivations of attackers have evolved, so too have their methods and the amount of effort they are willing to expend in taking down a targeted web application or hosting environment.

This white paper outlines a methodology for enumerating the weaknesses in a web application's architecture and service relationships that can be exploited remotely to cause a DoS condition and identifying steps the application owners can take to detect and prevent such attacks.

**IOActive**<sup>™</sup>  
Hardware | Software | Wetware  
SECURITY SERVICES

---

# Contents

- Types of Denial of Service ..... 3
  - Volumetric DDoS ..... 3
  - Layer 3 DDoS ..... 4
  - Layer 7 DDoS ..... 4
- Layer 7 DoS Implications ..... 5
- Web Application Architecture ..... 6
  - Presentation Layer ..... 6
  - Business Layer ..... 6
    - Application Layer ..... 7
    - Service Layer ..... 7
  - Data Layer ..... 7
    - Data Access Layer ..... 7
    - Persistence Layer ..... 7
    - Data Sources ..... 7
    - Domain Objects ..... 7
    - Service Interface Layer ..... 7
    - Network Services ..... 7
- Layer 7 Testing Framework ..... 8
  - Replicating the Standard Attacker Toolset ..... 8
  - Targeted and Unexpected DDoS Conditions ..... 9
  - Pre- or Post-authentication Attacks ..... 9
  - Location, Location, Location... ..... 9
- Layer 7 DDoS Testing Methodology ..... 10
  - Phase One: Application Crawling, Mapping, and Classification ..... 11
  - Phase Two: Pre-authentication Testing ..... 11
  - Phase Three: Authentication, Lockout, and Reset Resilience Testing ..... 12
  - Phase Four: Post-authentication Testing ..... 13
  - Phase Five: Vulnerability Consolidation, Analysis, and Reporting ..... 14
- Conclusion ..... 15
- References ..... 16

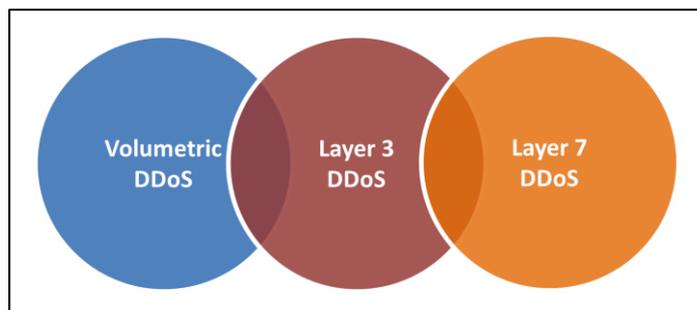
---

## Types of Denial of Service

There exist an increasingly broad array of attack vectors designed to disrupt the proper functioning of Internet applications and websites. These disruptions are commonly referred to as a denial of service (DoS) and are typically associated with a single source of attack. When DoS attacks are initiated from multiple sources simultaneously against the same target, they are referred to as distributed denial of service (DDoS) attacks. DDoS attacks have traditionally been associated with botnets—large networks of rogue and remotely controlled victim computers (ranging from several hundred to over ten million victims) that are centrally managed by a criminal operator.

Over recent years, the distinction between DoS and DDoS has become fuzzy. The number of attackers required to conduct a successful DDoS attack has shrunk, and access to powerful commercial cloud hosting platforms has increased.

DDoS attack techniques and vectors targeted at web applications can be broadly grouped into three major categories:



*Figure 1–Attack Categories*

### Volumetric DDoS

Volumetric attacks are tuned to saturate the bandwidth of the web application’s hosting infrastructure by directing large amounts of network traffic to the target. Historically, the unwanted traffic has been in the form of User Datagram Protocol (UDP) or Internet Control Message Protocol (ICMP) floods as the attacker can disguise their location by spoofing alternative source addresses—making it more difficult to block or filter. Attackers often employ specialist agents when launching an attack and, in more recent years, have exploited vulnerabilities in DNS and other ubiquitous Internet services to reflect and amplify their attacks.

Volumetric attacks are easy to initiate and do not require identification and successful exploitation of application weaknesses. While not particularly sophisticated and representing the lowest bar in the DDoS attack hierarchy, these attacks can be devastating to organizations that have not invested in appropriate network filtering infrastructure or negotiated scalable provisioning contracts with their hosting providers. In recent years, volumetric DDoS attacks have generated hundreds of gigabits of traffic per second and have disrupted some of the largest Internet providers in the world.

---

## Layer 3 DDoS

Layer 3 attacks target nuances and weaknesses in the TCP stacks that govern how data is transported between a web application's infrastructure devices and operating systems. Attackers launch specially crafted packets designed to overflow and disrupt TCP state information, causing additional work for the network processing functions on the target device and slowing down responses.

Historically, the most common Layer 3 DDoS vectors have included TCP SYN floods, TCP fragmentation, Teardrop, and other related low-rate attacks. These vectors are preferred by attackers who wish to target a particular web application without disrupting intermediary service providers. Layer 3 DDoS attacks require fewer machines than volumetric attacks to achieve a DoS status.

Since Layer 3 DDoS attacks tend to consume far less bandwidth than volumetric attacks, they are much easier to combat at the infrastructure level. Simple signature-based filtering systems are often an adequate defense.

## Layer 7 DDoS

Layer 7 DDoS attacks target specific weaknesses in the configuration of the web application and intermediary supporting services—causing them to slow down, hang, or crash. In most cases, Layer 7 attacks manipulate HTTP requests sent to the web server, exploiting vulnerabilities within the web server software or the custom code and business logic of the organization's application.

By targeting the custom code and business logic of the application, attackers seek to cause the application to become slow and unresponsive to legitimate users and customers. Traditionally, most successful attacks have focused on causing the application to perform intensive processing functions or exhausting memory handlers.

Because they target specific vulnerabilities and weaknesses in the web application's logic and code, it is considerably more difficult to combat Layer 7 attacks using filtering technologies. Organizations must focus on both the design of the application architecture and identify and limit access to critical logic blocks that require intensive processing or system resources.

---

## Layer 7 DoS Implications

A Layer 7 DoS state can be achieved using a broad range of attacks. While the tools and methodologies can vary, they typically result in freezes, crashes, and reboots, and have the following major effects:

- CPU Utilization
  - Starvation (99%+ utilization, forces other critical processes to halt or flounder)
- Server Memory
  - Invalid memory allocation, access, or leakage
  - Starvation (other processes cannot run or must swap to disk)
- Processes and Threads
  - Deadlock (process is or can be frozen)
  - Fork bomb (process continually replicates itself to deplete system resources)
  - Race condition (output is dependent on the sequence or timing of other uncontrollable events)
  - Resource starvation (encountered in multitasking applications where a process is perpetually denied necessary resources)
  - Thread starvation
- Disk
  - Disk overflow (capacity is consumed, applications cannot write to the disk)

Post-attack analysis will often reveal a combination of the following root causes:

- Bugs or implementation flaws
  - Poor input filtering and validation
  - Failure to supply required elements or objects
- Session management
  - Limited connection pool
  - Expensive session generation and login processes
- Application logic and hosting environment
  - Application logic open to abuse
  - Time-degrading application actions
  - Bottlenecks in application framework or environment
- Insecure features or unreasonable use expectation
  - Trusted input/action sequence
  - Human actions were expected

---

## Web Application Architecture

Understanding the logical and physical architecture of a web application is key when assessing its resilience against DDoS attacks. Not all web applications are the same, there are a diverse range of architectures employed to meet specific business requirements. Each architecture uniquely effects an application's robustness and the strategies that should be employed during an assessment.

It is important to agree on a common nomenclature in order to limit any potential misunderstandings when defining a DDoS testing methodology. At a high level, most web applications comprise of three tiers: the presentation layer, the business layer, and the data layer. For testing purposes it is important to clearly define the application's layers and interfaces.

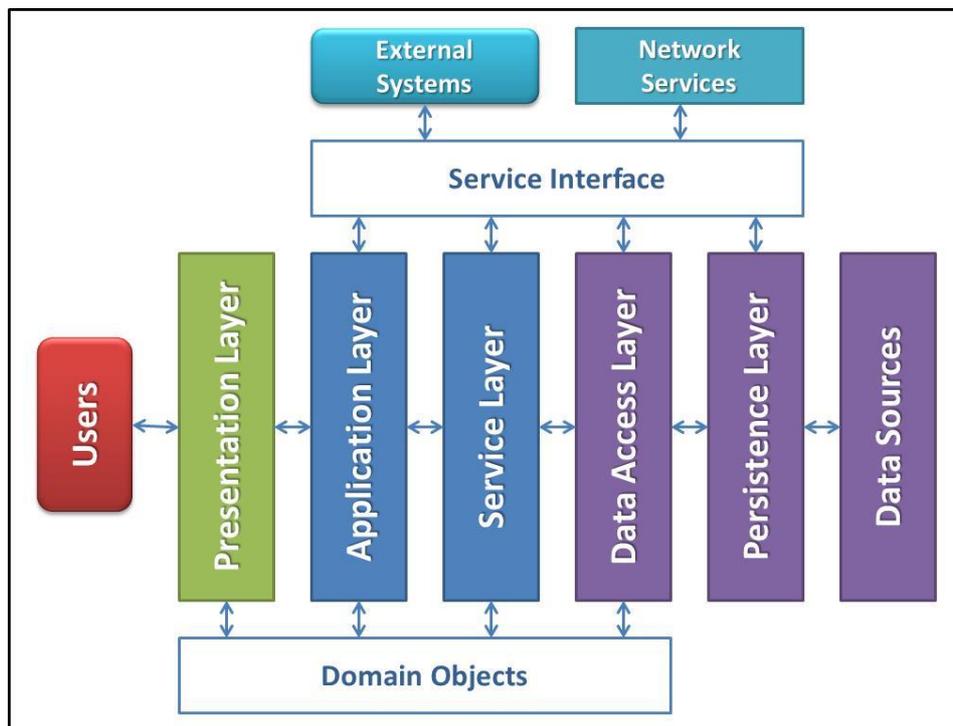


Figure 2–Web Application Architecture

### Presentation Layer

The presentation layer includes the web application components that directly interface with the user's computer system. They encompass the user interface (UI) components and the UI process components.

Some typical technologies used at this layer include HTML, DHTML, JavaScript, AJAX, CSS, Java Applets, and Flash.

### Business Layer

The business layer contains the custom logic and primary processing of user interactions with regards to the business data and objectives. The layer encompasses business workflows, application facades, business entities, and the logical web services.

---

Some typical technologies employed at this layer include Java, J2EE, VB.net, C#.net, Ruby, and Ruby on Rails, etc.

***Application Layer***

Business logic, initial data parsing, flow control, and application facades

***Service Layer***

Web services and interfaces to local architectural components and the data layer

**Data Layer**

The data layer contains dynamic data access and storage sources. It also provides for the management of database records and high-volume performance optimizations.

***Data Access Layer***

Repository interfaces (DAO)

***Persistence Layer***

Object-relational mapping (ORM), Java Persistence API (JPA), and Entity Framework (EF)

***Data Sources***

Data repositories (MS SQL, Oracle DB, Hadoop, and flat files)

In addition to these primary logical layers of an n-tiered architecture there often exist the following additional layers or components:

***Domain Objects***

Domain objects include shared resources and repositories between the logical layers of an application, such as graphic repositories, temporary file creation, and hosting of large reference files.

***Service Interface Layer***

The Service Interface Layer provides the objects for interfacing with external systems (remote repositories, intra-bank fund transfer systems, etc.) and/or locally provisioned network services.

***Network Services***

Network services are things that the physical and logical application components depend upon, such as DNS, LDAP, audit logging, and backups. They may be shared between application infrastructures.

---

## Layer 7 Testing Framework

The primary goals of Layer 7 testing are to enumerate the weaknesses in the application's architecture and service relationships that can be exploited remotely to cause a DoS condition and to identify steps the application owners can take to detect and prevent such attacks.

The assumption is that several instances of the same DoS vector can be used in a botnet or similarly automated toolset to cause the web application to crash or temporarily lose service. The DoS vulnerability consists of a weakness in a web application that could be exploited for DDoS in the absence of alternative traffic blocking or filtering mechanisms.

### Replicating the Standard Attacker Toolset

The vast majority of tools used to orchestrate Layer 7 DoS attacks focus primarily on the presentation layer. There are over 800 different DoS tools which can be easily located using popular Internet search tools that, when used by multiple attackers (as part of an online-protest movement), can constitute a DDoS attack.

Source code for most of these tools is also readily available and is often compiled into distributable software agents by protest movements that seek to target a single organization. In addition, most "commercial" botnet agents have the native capability to craft and launch custom packets, allowing them to successfully engage in any class of DDoS attack.

While there are hundreds of off-the-shelf DoS/DDoS tools, their primary mechanisms of attack are:

- Web Server Connection Starvation
  - Virtual sit-in (coordinated, simultaneous, and repeated web browser visits)
  - HTTP GET or POST Floods (httpflooder, RussKill, DirtJumper, and Brobot)
  - Long HTTP form field submissions (R-U-Dead-Yet (RUDY))
  - Slowly stepping through each line of an individual HTTP request, but never completing the request, in order to keep the session open for an extraordinarily long time (Slowloris, Qslowloris, and OWASP HTTP Post Tool)
  - Slow read attacks that send legitimate application layer requests but read responses very slowly, thus trying to exhaust the server's connection pool (SlowHTTPtest)
- Web Server Vulnerability Exploitation
  - Known vulnerabilities in popular web server applications that, like the "ping of death" of old, cause the server to crash or reboot (Apache Killer and Apache Struts)
  - Hash DoS that exhaust the CPU by forcing a large number of collisions via a single request with many parameters (CCC Hash DoS and Hash DoS Tester)
  - Regular expression implementations that work very slowly under extreme conditions (ReDOS)

---

## Targeted and Unexpected DDoS Conditions

As the goals and motivations of attackers have evolved, so too have their methods and the amount of effort they are willing to expend in taking down a targeted web application or hosting environment. While the previous generation of DoS and DDoS agents were successful in exploiting weaknesses within the presentation layer of the application, skillful attackers are now more likely to perform reconnaissance against the target prior to attack and identify business layer and data layer vulnerabilities.

The purpose of reconnaissance is typically to identify specific URLs and web application features capable of inducing a load on the servers which could lead to resource starvation and ultimately a DoS condition. These vulnerable URLs are most likely to be associated with weaknesses in the business layer or data layer of the application. Once vulnerable URLs have been identified, they can be shared with fellow attackers and used as target information for the toolset the attackers have chosen.

Organizations may also find their web applications suffering from unexpected DDoS effects due to unpredictable user demands following marketing campaigns and incorrectly derived and shared URLs.

## Pre- or Post-authentication Attacks

The vast majority of DDoS attacks are directed at pre-authentication web application content. Pre-authentication content is targeted because access to the specific URLs and content is much easier (not requiring to maintain session state), and it generally allows the attackers to continue to maintain a relative level of anonymity.

Of the pre-authentication content and processes likely to be targeted, the authentication process itself is widely acknowledged to contain more intensive business level or data level interaction, and is often singled out.

Post-authentication can be a viable DDoS route if authentication vulnerabilities or bypasses are uncovered by the attackers, or if the mechanics of authentication can be statically combined with the DoS payload. In many cases, post-authentication DoS requires fewer attackers to be successful, in part because they are already operating within a trusted element of the application.

## Location, Location, Location...

Short of infrastructure components and servers within the web application's hosted environment being compromised and used interactively by an attacker, all DDoS attacks will be initiated external to the application—inevitably from across the Internet.

While DDoS attacks can be initiated external to the web application during an assessment, testing speed and accuracy can be increased by using appropriate instrumentation throughout the environment. In particular, monitoring the dynamics of the application between each of the major architectural layers (as well as access to appropriate server/service logs) can help to identify DoS test cases that have, or are likely to yield, the greatest effect on the entire application.

---

It is often advantageous to observe how a particular DoS test case is interpreted and relayed between layers, and then to launch new test cases from between the application layers (replaying original and modified traffic) in order to speed up testing and provide greater focus on root cause. This approach also enables testing of secondary DoS cases whose results were obscured by earlier “upstream” vulnerabilities—due to application components in previous levels failing for other reasons.

## Layer 7 DDoS Testing Methodology

Testing web applications’ susceptibility to DDoS attacks requires a multi-faceted strategy in order to identify and understand the vulnerabilities inherent in the application layer, business layer, data layer, service interfaces, and critical network features.

All testing should be conducted in a mirrored environment that accurately reflects the configuration and physical architecture of the final deployed application. Also, the appropriate application QA and UAT organizations should be available for consultation throughout the testing period.

At a minimum, gray box testing is recommended. The speed of testing can be increased and the load on the test environment can be reduced if the consultant has ready access to event logs, process monitors, and disk storage information.

Layer 7 DDoS testing is typically divided into the following key phases:

1. Application crawling, mapping, and classification
2. Pre-authentication testing
3. Authentication, lockout, and reset resilience testing
4. Post-authentication testing
5. Vulnerability consolidation, analysis, and reporting

Phases 2–4 each require tests to be generated that reflect the following characteristics:

- A high rate of traffic that emulates the attack profiles generated by standard DDoS attack tools known to target the application layer
- Emulate DoS agents impersonating legitimate services and browsers
- Emulate DoS agents with Cookie and session management support

---

## Phase One: Application Crawling, Mapping, and Classification

The purpose of this phase is to map the pages, objects, and resources of the web application in order to identify, prioritize, and tune the later testing phases. One critical objective of this phase is to observe and time the application's responses to each request.

Mapping the application can be achieved using a variety of standard web-crawling tools (WGET) to enumerate publicly-accessible page content. Authentication and post-authentication application content can be enumerated using automated web-crawling tools (Black Widow Crawler) or personal proxy services (Burp Proxy).

Each page or object should be requested between 20 and 100 times and page return times recorded. Application content or objects with statistically significant longer responses are primary candidates for targeted DDoS investigation.

---

**Note** If the web application is live and/or under use or testing by other departments while page response time is being investigated, then the same tests should be repeated at different times of the day.

---

## Phase Two: Pre-authentication Testing

The purpose of this phase is to identify vulnerabilities and assess the web application's resilience to DoS attacks that target content and objects accessible to unauthenticated users.

Web application URLs and objects identified and prioritized in Phase One serve as the initial order of testing.

1. Target slow performing pages and objects using flooding techniques such as HTTP GET Flood and HTTP POST Flood DDoS attack vectors
2. Target and test slow performing pages and objects for resilience to malformed HTTP request attacks such as Slow Read and Slow Step
3. Target forms and user-definable fields using Long HTTP Form Field Submission DDoS (RUDY class of attacks), Hash DoS, and Regular Expression DoS attack vectors
4. Target page variables and objects not ordinarily edited or visible by the user using Long HTTP Form Field Submission DDoS (RUDY class of attacks), Hash DoS, and Regular Expression DoS attack vectors
5. Subject forms, user-definable fields, and other non-user page variables to business layer and data layer vulnerability discovery techniques; subject any uncovered vulnerabilities to increasing volumes of manipulation while monitoring system responsiveness, including:
  - a. SQL Wildcard Attacks (Ref: OWASP-DS-001)  
SQL server LIKE operator supports extra wildcards such as "[", "[^]", "\_", and "%

- 
- b. Buffer Overflows (Ref: OWASP-DS-003)  
Client-side validation is bypassed to submit unexpected length strings
  - c. User Input as a Loop Counter (Ref: OWASP-DS-005)  
Server-side applets are executed multiple times based upon user-defined integers
  - d. Writing User Provided Data to Disk (Ref: OWASP-DS-006)  
Long values or large files are submitted to the application and logging or inspection fails
  - e. Storing Too Much Data in Session (Ref: OWASP-DS-008)  
Overloaded and complex session variables cause the business layer to overflow
6. Test state management system (session tracking) for overflows, tardy validation, injection vectors, saturation, and reuse by multi-location attackers
  7. Test data upload mechanisms under high load, incomplete and high CRC failure transfers, and oversized files
  8. Test and evaluate built-in DoS detection and blocking rules against automated DDoS attack tools

### **Phase Three: Authentication, Lockout, and Reset Resilience Testing**

This phase of testing focuses on the methods in which users can login, authenticate, lockout, and reset their accounts within the web application. This component is singled out for special attention due to the fragile nature of the process and the complex dependencies between the business layer, data layer, and other infrastructure services.

URLs and objects identified and classified as being associated with the authentication and account management processes during Phase One are singled out for additional, specific DoS testing.

1. Identify thresholds for simultaneous authentication requests, test thresholds, and monitor the load placed on each logical layer of the application
2. Identify vulnerabilities within the authentication process that could cause it to be bypassed or encapsulated into a reduced number of HTTP GET or HTTP POST requests which would make it easier for common DDoS tools to target post-authentication URLs and objects
3. Identify overflows and computationally intensive account variables that increase the demands upon the authentication processes
4. Perform tests that quantify the scope and thresholds at which an attacker can cause a DoS against web application users seeking to login
  - a. Check account lockout thresholds and lockout durations

- 
- b. Enumerate accounts and test rapid submission of legitimate accounts with incorrect credentials
  - c. Test password reset issuances (repeated against a single account, all known accounts, and all possible accounts)
  - d. Test spam delivery consequences of forced “forgot my password” reset processes
5. Assess strength of any CAPTCHA used as a part of the authentication process to reduce the effectiveness of automated attacks
  6. Assess processes for automated or manual reset of locked accounts
    - a. Test password reset issuances (repeated against a single account, all known accounts, and all possible accounts)
    - b. Verify onerous and computationally intensive “forgot my password” reset processes

#### **Phase Four: Post-authentication Testing**

Targeting and ultimately exploiting DoS vulnerabilities that are only accessible post-authentication is beyond the capabilities of most common DDoS tools used by cyber-protestors and managed DDoS service providers.

To successfully launch a DDoS attack against post-authentication content, the attacker will first have to complete reconnaissance of the web application and identify probable problematic URLs or objects. Armed with this intelligence, the attacker will then have to obtain a current session management “token”, and incorporate it into the preformatted DoS requests.

Web application URLs and objects identified and prioritized in Phase One serve as the initial order of testing.

1. Identify post-authentication URLs and objects that can be accessed (either partially or fully) with fake or impersonated session credentials
  - a. Conduct standard DoS tests (as defined in Phase One) against the accessible content, using the fake or spoofed session credentials
2. Conduct standard DoS tests (as defined in Phase One) against all post-authentication accessible content, using legitimate user credentials
3. Identify and test thresholds that restrict simultaneous use of authenticated session management tokens and processes that could result in a DoS for legitimate (authenticated) web application users

---

**Note** The attacker may not need to have direct knowledge of any user credentials to successfully target post-authentication content. Attackers that leverage botnets to launch a DDoS attack could have simply acquired and duplicated an active session management “token” from an existing botnet victim (or victims).

---

## **Phase Five: Vulnerability Consolidation, Analysis, and Reporting**

The final phase focuses on consolidating all previous test results, reviewing system and network logs, analyzing performance metrics, and examining application source code. The goal is to identify the next steps in remediating any DoS and DDoS vulnerabilities.

A critical component in identifying DoS and subsequent DDoS vulnerabilities lies in correlating the datasets that were gathered throughout each of the prior testing phases. Since the purpose of testing the web application is to identify potential and probable DoS conditions within a QA or UAT environment, without necessarily making the entire system unresponsive, the data related to performance metrics and log events must be married to the individual test cases and interpreted.

The key stages to this final phase are:

1. Correlate event logs and performance metrics to specific DoS test cases or categories of test cases, divided by:
  - a. Pre-authentication, no session management
  - b. Pre-authentication, web browser
  - c. Pre-authentication, with session management
  - d. Authentication, no session management
  - e. Authentication, web browser
  - f. Authentication, with session management
  - g. Post-authentication, web browser
  - h. Post-authentication, with session management
2. Examine event logs, disk storage metrics, and other logged information for saturation and unrestricted overwriting of file content
3. Identify and rank test cases (and the specific parameters associated with the exploit that targeted the vulnerability):
  - a. Caused a DoS condition  
A test case that caused the web application to stop performing during testing
  - b. Will eventually cause a DoS condition  
A test case that would have caused the web application to stop

---

functioning if parameters had been adjusted and was observed to have a detrimental effect on the application during testing

c. Would likely cause a DoS condition

A test case that, based upon log and performance metrics, would likely cause DoS conditions, but had no noticeable detrimental effect on the application during testing

d. May cause a DoS condition in certain circumstances

A test case that caused secondary application effects that could cause a DoS condition if the application architecture had been slightly different, but did not during testing

e. Needs further investigation

A test case that was inconclusive

4. Examine the web application source code associated with the vulnerable conditions classified in the ranges 3.a to 3.b (caused or will eventually cause a DoS condition), and recommend improving or optimizing code quality and integrity to remove or prevent the test case from being successful in the future
5. Identify optimizations and thresholds that could be added to the web application architecture and source code to prevent vulnerable components from being exposed to DDoS attacks
6. Identify monitoring points and detection rules (host, server, and application code) that could be used to detect when a DDoS attack is being attempted
7. Identify application features that could be leveraged by attackers to launch attacks against other web applications (reflection attacks) or users (alert spam)
8. Provide packet captures that can be replayed by QA and UAT tools in the future for regression testing purposes

## Conclusion

Enumerating the weaknesses in a web application's architecture and service relationships that can lead to a DoS condition requires careful research and execution. The process begins by understanding the implications of a Layer 7 DoS condition and the logical and physical architecture of a web application. Building a testing framework by selecting toolsets, performing reconnaissance, and determining probable attack locations is the next step. Finally, exhaustive testing in a mirrored environment is performed in five phases. The goal of this effort is to clearly communicate how the application's owner can detect and prevent Layer 7 attacks.

---

## References

[https://www.owasp.org/index.php/Testing\\_for\\_Denial\\_of\\_Service](https://www.owasp.org/index.php/Testing_for_Denial_of_Service)

<https://code.google.com/p/slowhttptest/>

[https://www.owasp.org/index.php/Regular\\_expression\\_Denial\\_of\\_Service\\_-\\_ReDoS](https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS)

[https://www.owasp.org/index.php/Testing\\_for\\_Denial\\_of\\_Service](https://www.owasp.org/index.php/Testing_for_Denial_of_Service)

### **About IOActive**

*IOActive is a comprehensive, high-end information security services firm with a long and established pedigree in delivering elite security services to its customers. Our world-renowned consulting and research teams deliver a portfolio of specialist security services ranging from penetration testing and application code assessment through to semiconductor reverse engineering. Global 500 companies across every industry continue to trust IOActive with their most critical and sensitive security issues. Founded in 1998, IOActive is headquartered in Seattle, USA, with global operations through the Americas, EMEA and Asia Pac regions. Visit [www.ioactive.com](http://www.ioactive.com) for more information. Read the IOActive Labs Research Blog: <http://blog.ioactive.com/>. Follow IOActive on Twitter: <http://twitter.com/ioactive>.*