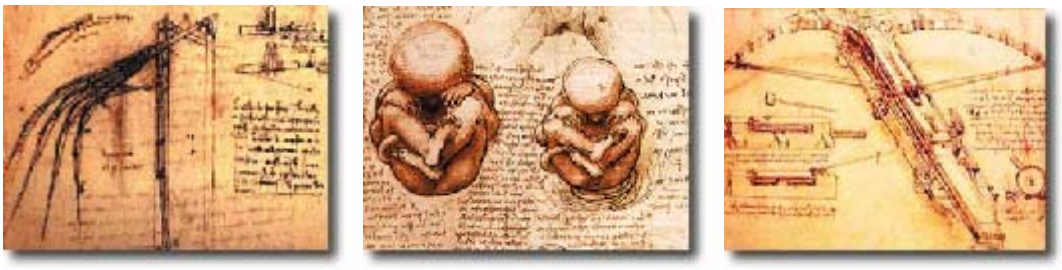
CORSAIRE

The natural choice for information security solutions



A Corsaire White Paper:

Application Denial of Service (DoS) Attacks

Author	Stephen de Vries
Document Reference	03-03-04 Application Level DoS Attacks v0.4.doc
Document Revision	V1.0 Released
Date	1 April 2004

Table of Contents

1. Introduction

In order to achieve business goals, organisations frequently have to develop bespoke application solutions or customise commercial off-the-shelf (COTS) packages. These range from complex back-office database applications, CRMs and asset management systems to customer-facing fat and thin applications. Corporate web-applications offer anything from a simple brochure request to a full e-business implementation.

Availability of these services is important for customers and users of the site, with any disruption directly affecting revenues, negatively impacting confidence in the company or even damaging the brand.

Denial of Services (DoS) attacks aimed at disrupting network services range from simple bandwidth exhaustion attacks and those targeted at flaws in commercial software to complex distributed attacks exploiting specific COTS software flaws. These types of attack are not new and have been used to devastating effect to prevent normal operation of the victim sites. Historically, these attacks by hacktivists and extortionists alike have targeted companies as diverse as eBay and Microsoft, the RIAA and SCO, and a plethora of online gambling companies.

Attackers have not, as yet, exploited the full range of vulnerabilities present in many online services – particularly attacks aimed at the application and data processing layer. With the rise of increasingly targeted and motivated attacks and attackers, these application level DoS attacks will inevitably be exploited for nefarious gains.

1.1 Denial of Service Attacks

Conceptually, DoS attacks are intended to prevent legitimate users, customers or clients of a site from successfully accessing it. Traditional DoS attacks have been aimed at consuming resources or disrupting services at the network or operating system level.

Typical examples are server-based attacks such as SYN floods and bandwidth exhaustion attacks that attempt to saturate the victim's Internet connection with spurious traffic.

2. Application DoS Overview

Application DoS attacks exploit flaws in the bespoke application design and implementation to prevent legitimate access to the victim's services. They represent a subset of potential attacks on such applications, as they are aimed specifically at disrupting operation rather than subverting the application controls.

Attacks based on exploiting these flaws can offer the attacker a number of advantages over traditional DoS attacks:

- *The attacks will typically not be detectable or preventable by existing security monitoring solutions*[1] – Since the attacks do not consume an unreasonable amount of bandwidth and could, in many cases, be indistinguishable from normal traffic.
- *Application attacks are more efficient* – The attacker may not need as much resource at their disposal to successfully complete the attack. Application level attacks target bottlenecks and resources limitations within the application and do not require many compromised "zombie" systems or a large amount of bandwidth. Furthermore, they can be targeted at the weakest link in an environment – for example if a web-farm of a hundred servers relies on a single back-office host to authenticate users, an application attack may be able to directly target it.
- *Application attacks are harder to trace* – Application level attacks normally use HTTP or HTTPS as their transport. Proxy servers can therefore be used to obfuscate the true origin of the attacker; and many are available for an attacker to redirect his malicious traffic. Many of these proxy servers do not keep logs of connection attempts and could therefore successfully hide the true origin of the attacking host.

2.1 Root Causes

Application security flaws exist due to flawed design and implementation, often as a result of incorrect assumptions made during the development process.

2.1.1 Reasonable Use Expectations

Applications are designed and built based on expected use criteria. Capacity planning and load-testing are typically based on these criteria of expected and maximum use.

As with network DoS attacks, this narrow view of how the application may be used exposes it to disruption when those expectations are exceeded.

2.1.2 Application Environment Bottlenecks

Applications are becoming increasingly complex and often reside in distributed environments. The traditional approach to providing adequate capacity has been to focus resources on the client-facing tier, e.g. having a large web-farm. Often, back-office functionality such as authentication and database access have far less resource allocated.

Whilst the front-end may therefore be able to cope with far more than the expected or possible demand, there may be hidden bottlenecks elsewhere within the environment that, while not accessible to an attacker via the network, can be targeted through the application itself.

2.1.3 Implementation Flaws

Most existing DoS attacks have been designed to crash COTS applications or services. This same class of application attack is equally, if not more, likely to work on bespoke applications. An attacker may be able to crash application or middleware components using buffer overflow

[1] E.g. intrusion detection (IDS) and intrusion prevention systems (IPS)

style attacks, for example. This could be due to a poorly implemented software module such as an ISAPI filter failing when sent excessive input data.

2.1.4 Poor data validation

Another common mistake is to implicitly trust input from the client side and user. In many cases, the application acts as a structured portal through which users interact with back-end services and databases. If these functions do not sufficiently validate client-input, it may be possible to disrupt those back-end services through attacks such as Database (SQL) or Script (XSS) insertion.

2.2 Application DoS Attack Vectors

2.2.1 Automated Data Submission

Web application architects and designers frequently assume that the user of the application is a person and not an automated script. This assumption often leads to vulnerabilities that can be exploited by attackers who craft automated tools to attack the application.

A simple example would be a dictionary attack on the authentication process. If a username is known, an attacker may automate the login process using passwords from a dictionary of common words until a match is found. There are many tools available to perform dictionary attacks on various protocols.

Similarly, variations on this theme can be used to attack specific weaknesses in the application's design, for example by overwhelming application functions, invoking race conditions or systematically attacking multiple entities.

2.2.2 Data Manipulation

Any data sent from the client-side may be manipulated by an attacker, from obvious form elements that are completed by the user to hidden fields, cookies and bespoke data channels.

Commonly, manipulation of this data before sending it to the server results in errors or unusual application behaviour – even as a result of simple modification such as sending letters instead of numbers. These back-end errors may in turn lead to exploitable vulnerabilities.

3. Application DoS Attacks

3.1 Attack Classes

3.1.1 Resource abuse and starvation

The simplest way to attack an application is to exceed expected use beyond the capacity of the environment. This is conceptually very similar to a Network DoS attack in that it prevents legitimate users from accessing the application while it is forced to deal with many false requests.

3.1.2 Exploitation

More sophisticated attacks are based on an analysis of the application and its functionality, and identification of susceptible components. This may involve overflows on application buffers to crash the components on the servers themselves, or destruction of back-end data through SQL insertion for example.

3.2 User Based Attacks

In any application that makes use of user based authentication (e.g. a username and some form of password), the authentication system may be vulnerable to DoS attack.

3.2.1 Login Process DoS

It may be possible to overwhelm the login process by continually sending login-requests that require the presentation tier to access the authentication mechanism, rendering it unavailable or unreasonably slow to respond.

3.2.2 Username Enumeration

When a user enters an incorrect username and/or password, the application should respond with a generic error message stating that the information entered was incorrect. If the application explicitly states which component of the username/password pair was incorrect then an attacker can automate the process of trying common usernames from a dictionary file in an attempt to enumerate the users of the application.

Whilst applications may handle authentication failure messages correctly, many still allow attackers to enumerate users through the *forgotten password* feature. Typically, this feature requires the user to enter some form of identification before re-establishing access to an account. Many forgotten password features require only the username to be entered before responding with a message as to its validity. If the application behaves in this manner, as with login messages it is a trivial exercise to automatically deduce valid usernames.

With a list of valid usernames an attacker can attempt to mount automated attacks against the authentication mechanism.

3.2.3 Account Lockout

In order to mitigate dictionary attacks, many applications implement an account lockout procedure to freeze a user's access should they make too many failed login attempts. The lockout facility is designed to deny access to users performing illegal operations, or who are suspected of malicious intent.

Since this mechanism is designed to deny access to the application, if it is not properly implemented it may be abused by attackers to deny access to legitimate users. A number of strategies may be employed by the lockout facility:

1. *Lock the account and require manual intervention to re-enable it* – Any form of permanent lockout should be carefully evaluated to ensure that the process used to

unlock the accounts doesn't require too much resource. Care should be taken when choosing lockout mechanisms that require manual intervention since they may be difficult to action should a large number of users be locked out simultaneously.

2. *Lock out the IP address of the user temporarily or permanently* – Locking out users based on their IP address could create unforeseen problems as many use shared proxy servers[2] or are in NAT environments. Locking out the proxy server's IP address could inadvertently deny service to many legitimate users.
3. *Lock the account out for a certain period of time only* – This is often the best choice of lockout mechanism for as it prevents attackers from performing dictionary or brute force attacks, but still prevents them from locking out legitimate users. It also requires the least resource and interaction from the provider.

Attackers can abuse the first two lockout mechanisms using the same technique:

1. Enumerate usernames through another vulnerability in the application
2. Attempt to authenticate to the site using valid usernames and incorrect passwords
3. Systematically lock-out the accounts after the specified number of failed attempts

At this point legitimate users will not be able to use the site, and will have to follow the account unlocking procedure.

Depending on the business requirements of the application and the account un-locking procedures, automated account locking of a large portion of the application's user base can have an immediate damaging impact on the availability of online services.

3.2.4 Creating multiple false users

Some applications have the requirement of allowing users to sign-up, register or join the application by a fully automated method. This typically involves the user choosing a username and password and entering their required details. If there is no manual interaction required in this process, and if the application does not provide a mechanism to ensure that it is a person entering the data, then it may be susceptible to automated attack.

An attacker could use files containing lists of usernames, passwords, telephone numbers, addresses and even fictional credit card numbers. These lists could be created by manipulating the data from common dictionaries. Once in hand, the attacker could create a program that submits the registration forms repeatedly; adding a large number of spurious users to the application.

This could in itself be a form of attack, since any statistical analysis of the user data will be largely skewed to the side of fiction. Besides the threat to business process posed by the existence of a large amount of fictional data, there is also the risk that the attacker can use these user accounts to perform transactions through the application. With a pool of thousands of fictional users at their disposal, an attacker could develop starvation attacks that would have all the users purchase goods at exactly the same time, for example.

Checks should be built in to susceptible portions of the application to ensure that a person, and not an automated script, is submitting the data. One technique used particularly in high traffic sites is for the application to generate a random sequence of characters, displaying them as images that are slightly distorted and then requires the user to enter them[3]. This will foil most automation attempts as image recognition software has difficulty recognising these characters.

Another technique is to send an email to the user that requires them to click on a link to activate their account. While this may deter casual attackers, those with more determination can setup a system that receives and process those emails automatically.

---

[2] E.g. AOL, cable and DSL users.

[3] For more information on this approach see the CAPTCH project at http://www.captcha.net.

3.3 Transaction Resources

Each transaction performed by the application requires resources such as processing time and storage space. By triggering an excessive number of transactions, the attack may be able to prevent normal operation of the application.

Some transactions may even incur a financial cost, such as when requesting third party services that are billed per transaction. Sending SMS messages or implementing automatic call back facilities also incur costs for each transaction. If an attacker has created thousands of fictional users, then it may be possible, through the use of an automated script to perform thousands of transactions that could incur a substantial cost.

3.4 Attacking the Database

Many web applications use databases extensively to store static data, which is accessed and modified infrequently, and dynamic data which is used to manage the user sessions. Each database operation incurs an overhead in storage space and processor cycles with static limits to both these resources. Simple application functions such as searches, performing user authentication or submitting registration forms often require the database to access data and to consume resources.

An attacker needs access to a large amount of bandwidth should he wish to attempt to flood the victims internet connection. Considerably less resource may be required if the attack is instead directed at a few carefully selected database queries. Complex database queries that can be performed without authentication are the most at risk, since they consume a larger amount of resource and can be accessed by anyone. Search functions that allow users to enter queries in natural language are a good example of a complex query. These require a comparatively large amount processing time for each query, especially if they allow the use of wildcard characters.

Should thousands complex queries repeatedly be submitted, the strain on the application may affect other more critical components, and could even prevent legitimate users from performing transactions.

3.5 Session ID Exhaustion

The HTTP protocol is stateless; meaning that each request to a web server is unique and unconnected to any preceding requests. A requirement of modern web applications is that a users multiple requests to be associated with each other across a "session", requiring unique HTTP requests be tracked and connected. A session typically begins when a user logs in to the application and ends when they logout. This session management is usually implemented through a unique, random identifier, called a Session ID, created at the start of the session. The client transmits this ID with all subsequent HTTP requests so that the server can identify the user.

The session IDs and corresponding session information is usually maintained in a database or at the application level. Each session ID that is generated therefore requires resource in terms of processing time and storage space. Attackers may use this to keep requesting session ID's in an attempt to exhaust the available resources and thereby deny access to legitimate users. The attacker's ability to request that the application generate a new session ID is largely controlled by the chosen session management strategy. As far as the generation of session IDs are concerned, applications can be broken down into three broad categories:

- *The application generates a session ID when the user visits a specific page, without authenticating.* This is the most vulnerable class of application, as the attacker can request many session IDs easily, and quickly.
- *The application generates a session ID only after the user authenticates and allows multiple sessions for the same user*. This requires the attacker has a valid user account on the application before session ID request's can be sent, but since the application allows multiple sessions for a single user it is still relatively easy to generate many session IDs in a short space of time.

- *The application generates a session ID after authentication and only allows one session per user.* This is a more secure approach, since the attacker will have to use multiple user accounts to cause the application to track multiple session IDs. For some applications where user accounts are granted through a manual or manually-assisted process, it will be almost impossible for an attacker to create the required number of user accounts. But, where applications are vulnerable to automated registration scripts (section 2.2.1), it is only moderately more difficult for an attacker to sign-up thousands of fictitious users and have them all log in and request session IDs at the same time.

The generation of session IDs and the resources used to manage sessions can often be overwhelmed if an attacker has the ability to generate a large number of session IDs.

4. Conclusions

Vulnerabilities in web applications can allow attackers to exhaust available resources and thereby deny access to legitimate users. Companies that rely on web applications to provide critical business functions are therefore at risk from attackers wishing to disrupt these functions by exploiting application level vulnerabilities.

Application based DoS attacks require considerably less resource in terms of processing power and bandwidth on the part of the attacker, and therefore present a higher risk to business as the number of possible threat-agents is much greater.

The possibility of a denial of service attack should be considered when designing, implementing and providing applications, and appropriate strategies and mitigation techniques put in place within the application.

Acknowledgement

This White Paper was written by Stephen de Vries and Glyn Geoghegan, Principal Consultants at Corsaire Limited.

About The Lead Author

Stephen de Vries is a Principal Consultant in Corsaire's Security Assessment team. He has worked in IT Security since 1998, and has been programming since 1997. He has spent the last four years focused on Ethical Hacking, Security Assessment and Audit at Corsaire, KPMG and Internet Security Systems. He was a contributing author and trainer on the ISS Ethical Hacking course and Technical Leader for the Automated Perimeter Scanning project co-coordinating a team of six developers in three countries.

Stephen's past roles have included that of a Security Consultant at a leading City of London Financial institution and also Security Engineer at SMC Electronic Commerce. At both positions he was involved in corporate security at many levels and was responsible for consulting on the paper security policies and procedures, conducting vulnerability assessments, designing, deploying and managing the security infrastructure of the organisation.

About Corsaire

Corsaire are experts at securing information systems. Through our commitment to excellence we help organisations protect their information assets, whilst communicating more effectively. Whether they are interacting with customers, employees, business partners or shareholders, our sound advice can help our clients reduce corporate risk and achieve tangible value from their investments.

Privately founded in 1997 and with offices in the UK and Australia, Corsaire are known for our personable service delivery and an ability to combine both technical and commercial aspects into a single business solution. With over eight years experience in providing information security solutions to the UK Government's National Security Agencies, Government departments and major private and non-profit sectors, we are considered a leading specialist in the delivery of information security planning, assessment, implementation and management.

Corsaire take a holistic view to information security. We view both business and security objectives as inseparable and work in partnership with our clients to achieve a cost-effective balance between the two. Through our consultative, vendor-neutral methods we ensure that whatever solution is recommended, an organisation will never be overexposed, nor carry the burden of unnecessary technical measures.

Corsaire have one of the most respected and experienced teams of principal consultants available in the industry and have consistently brought fresh ideas and innovation to the information security arena. We take pride in being a knowledge-based organisation, but we don't just stop there. Through a culture of knowledge-share, we are also committed to improving our client's internal understanding of security principles.

It is this approach to knowledge that differentiates us from most other information security consultancies. As a mark of this, we are known globally through our active contribution to the security research community, publishing papers and advisories on a regular basis. These we share freely with our clients, providing them with immediate access to the most up-to-date information risk management advice available, allowing them to minimize their exposure and gain an instant competitive advantage.

Whilst it is imperative for us to offer a high level of security to our clients, we believe that it is of equal bearing to provide a high level of service. At Corsaire our clients are not only protected but valued too. We work hard at building strong relationships that are founded on the cornerstones of respect and trust. With 80% of our customer base deriving from referrals we are certain that our clients value the quality, flexibility and integrity that partnering with Corsaire brings.

For more information contact us at **info@corsaire.com** or visit **www.corsaire.com**

