Don’t Drop the SOAP: Real World Web Service Testing

Tom Eston, SecureState
teston@securestate.com

Joshua Abraham, Rapid7
jabra@rapid7.com

Kevin Johnson, Secure Ideas
kjohnson@secureideas.net

Table of Contents

Abstract

Over the years web services have become an integral part of web and mobile applications. From critical business applications like SAP to mobile applications used by millions, web services are becoming more of an attack vector than ever before. Unfortunately, penetration testers have not kept up with the popularity of web services, recent advancements in web service technology, testing methodologies, and tools. In fact, most of the methodologies and tools currently available either do not work properly, are poorly designed, or do not fully test for real world web service vulnerabilities. In addition, environments for testing web service tools and attack techniques have been limited to home grown solutions or, worse yet, production environments.

This whitepaper will discuss the new security issues with web services, describe a new web service testing methodology, discuss new Metasploit[1] modules and exploits for attacking web services, and introduce a collection of open source vulnerable web services designed to be used within the Damn Vulnerable Web Application[2] (DVWA) that can be used by penetration testers to test web service attack tools and techniques.

Web Service Testing – The State of the Union

Testing of web services is an area into which the security community traditionally has not put much time and effort. Properly testing web services entails more than throwing a tool at a Web Services Description Language (WSDL) to see if there are security issues. Web services have morphed into a complex beast in which mobile applications for enterprise ERP systems like SAP use web services to handle transmitting and receiving sensitive information. Previously documented methodologies and testing techniques seem to only scratch the surface of vulnerabilities that could be identified; and many books on the subject are out of date and do not present the real world threats that are out there. In fact, documented techniques for real world testing remain mostly a mystery where only outdated guides, methodologies, and blog posts exist. Once penetration testers have the tools and knowledge, there needs to be proper testing environments in which penetration testers can test new techniques and learn new skills. Without a feasible and practical way for penetration testers to test new tools and methodologies, web service testing will remain a mystery.

Why Attack Web Services?

Web services offer the attacker a secondary vector to attack the application. Because in many cases these web services are designed to integrate remote systems into business processes or data, these services provide direct access for the attacker. The attacker potentially is able to attack vulnerabilities within the web service to bypass controls within the application. Furthermore, most developers and administrators are not considering the security ramifications of the web service. During penetration tests, it often is found that these services are configured or installed outside the protections within the web application. For example, these services are not behind the web application firewall or do not include the filtering incorporated. This is due to the assumption that the only client for the web service will be another application.

Process Improvements Needed

The majority of web services testing tools have not been built for security testing, but for quality assurance. In the past, the majority of presentations at security conferences involving web services have been from a developer’s perspective and also XML firewalls. The presentations about XML firewalls have been designed to try to reduce the risk of web services based attacks; and the presentations from a developer’s perspective have not been able to gain a wide adoption due to the technical nature of the content around web services because there are a variety of languages, frameworks, as well as usage of the web services.

The best tools for penetration testing web services include soapUI[3] (which makes a single web services request to a web service) and web application proxy tools such as Burp Suite. Currently there are no tools that provide in-depth coverage for web services testing from a security perspective. Over the past year ~14 modules have been added to Metasploit which are based on making SOAP requests to SAP systems. Each of these modules included SOAP XML messages that were built by hand and concatenated. This makes it clear that the work needed to build one of these modules is time consuming and error-prone.

Dramatic Increase in Web Service Usage

Web services have been in existence for quite awhile, but as an industry, we are finding that web services are being used more often every day. With the move to mobile applications and Web 2.0 mash up systems, the web service has become the default technology being implemented. During penetration tests, we commonly find web services being implemented and used, which is a change from as recently as five years ago. This increase is one of the main reasons this work has to be done. As stated throughout this paper, if we do not know how to test or do not test the web services, potentially enormous exposures remain.

The Problem with Testing Web Services

Typically web application penetration tests are not scoped properly to include the related web services. If they are, many penetration testers have no idea how to even begin testing them, nor do they understand the security implications of insecure web services. Once penetration testers begin the testing process, more confusion sets in especially if the web service is using custom and/or proprietary technology such as homegrown authentication mechanisms. Time that could be spent on assessing for flaws and vulnerabilities often is spent building tools or interfaces to test the web service. If proper scoping questions were asked before the engagement was scoped, the penetration tester’s time would be better spent and the client would receive more value from the testing. In our experience, many clients have web services that are customized for specific tasks so it is important for penetration testers to conduct proper pre-engagement scoping.

Once the pre-engagement scope is confirmed often times the next question is, “What type of assessment is most appropriate?” Typically, web application penetration testers use the Black, Grey, and White Box approach for testing web applications; and testing web services should be no different.

For example, in a Black Box assessment the penetration tester might focus more on the authentication of the web service by conducting brute force attacks as well as manually determining how the web service works. On the other hand, a Grey Box assessment typically is more thorough, as any user roles will be tested as well as any data validated through fuzzing techniques. A White Box assessment would be appropriate if the web service is part of a larger application or if there are custom components that should be reviewed from a source code level. BPEL web services are a great example where White Box testing may be appropriate.

The Problem with Current Testing Methodologies

The current “gold standard” of web service testing methodologies is the OWASP Testing Guide v3[4]. Unfortunately, this guide has not kept up with recent advancements in web service technology, not to mention the vast number of custom web services that are encountered by penetration testers. Testing methodologies should include not only technical details on how to test web services, but also non-technical information such as proper scoping as well as pre-engagement requirements, which often are overlooked by penetration testers.

In addition, current methodologies lack information on a complete threat model for web services. Depending on the data being exchanged, threats need to be carefully identified. For example, could web service data exchanges be intercepted via MiTM (Man-in-The-Middle) attacks? And, how does cloud or client based storage concern the security of the web service as well as various authentication requirements?

Testing also is traditionally focused on old technology and older vulnerabilities. For example, basic XML injection as well as XML structure testing is something that still needs to be tested; however, advanced web services such as Microsoft WCF (Windows Communication Foundation) have different testing requirements, as WCF can leverage clients like Microsoft Silverlight as well as use multiple communication protocols such as SOAP over HTTP, SOAP over TCP[5] and SOAP over Message Queues. These new technologies require updated testing techniques.

Lastly, most penetration testers focus on testing web services just like any other web application. While REST web services can be the exception in most cases, traditional SOAP based web services cannot be treated as such. Traditional SOAP based web services require unique approaches to fully cover the scope of the testing required.

The Problem with Current Tools

Current tools for testing the security of web services have been lacking not only in functionality but also in usefulness. Current tools usually do not support features typical web services use, like SSL; or the tools were designed for functional developer testing. Many of the tools currently in existence also are commercial based and offer only a small and incomplete subset of security testing tools.

Another issue involves testing newer web services technologies such as WCF (Windows Communication Foundation) web services. Typically, WCF services are difficult to test with current tools because many WCF web services are configured to use default TCP bindings vs. the less secure BasicHTTPBinding[6]. BasicHTTPBinding allows for HTTP as the transport protocol and provides legacy ASP.NET support. The BasicHTTPBinding configuration is supported in current tools including soapUI[7] and web proxies like Burp Suite[8]; however, many developers do not enable this in production environments. Tools that support native WCF protocols typically are commercial based and provide mainly functional testing.

The Lack of Testing Environments

A development that has enabled penetration testers to expand their skill sets and learn more techniques to test web applications is the creation of vulnerable practice applications. Some examples of these are WebGoat by OWASP[9], Mutillidae[10] by Adrian “irongeek” Crenshaw and Damn Vulnerable Web Application (DVWA)[11] by Ryan Dewhurst. These systems provide a penetration tester a series of pages that contain several vulnerabilities. A penetration tester then can either practice their skill or try out new tools against these vulnerable applications.

Currently, none of these systems, or for that matter any of the publicly available practice applications, contain web services. This means that any penetration tester looking to gain the skill set for testing web services or test any tools has no practical means of performing that testing. Currently the tester has to either use production systems or build their own web service to test against.

Testing environments also have been left up to the developer of the web service to create, and very rarely are these environments utilized for security testing. Most penetration testers have neither the time nor the desire to create a web service from scratch to do customized testing. Even if the tester has the knowledge or desire, the lack of existing model environments makes this process more difficult.

Web Services and the OSI Layers

Web Services are implemented by adding XML onto layer 7 applications such as HTTP. For example, the most common application that is used in web services is HTTP. The XML is constructed in a specific manner so that the sender and receiver of the message can understand its contents. The XML structure is known as SOAP, which stands for Simple Object Access Protocol. The method in which SOAP is used is not the same manner that web applications are used. Rather than the traditional request response, web services should be thought of more like SMTP. SOAP is more like SMTP because the server can easily forward the SOAP message or envelope onto another web service in-order to receive a response.

The Web Service Threat Model

Web Services in Transit

Any data being transmitted between a user and web services should be reviewed to ensure that all data is protected from being intercepted by a malicious attacker. One common misconception is to use BASIC Authentication to restrict access to other authorized users. Depending on the architecture of the web service and the nature of its use, BASIC Authentication may not be acceptable. Let us assume there is a web service that is exposed to the Internet and utilizes BASIC Authentication. If a use of the web service is done from a mobile device, then it would be possible to intercept the credentials being used. This could be done by performing a man-in-the-middle (MiTM) based attack. This will be discussed further to describe when SSL is sufficient, and when message/token based security should be considered.

The following threat model was documented as part of the Hacking Web Services book by Shreeraj Shah [12]and has been included in this whitepaper for the sake of completeness.

1) In Transit Sniffing or Spoofing
2) WS-Routing security concern
3) Replay attacks

Web Services Engine

1) Buffer Overflows
2) XML parsing errors
3) Spoiling Schema
4) Complex or Recursive structure as payload
5) Denial of Service
6) Large payload

Web Services Deployment

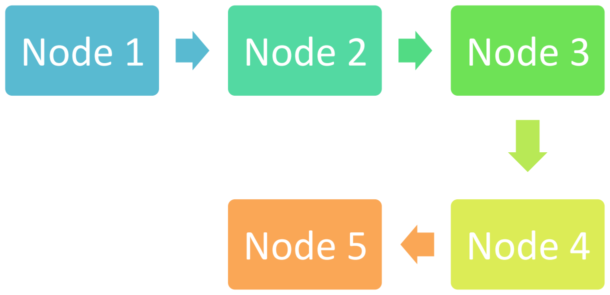
1) Fault code leaks
2) Permissions and Access issues
3) Poor Polices - secure coding / SDL process
4) Customized error messages (reduce info leakage)
5) Denial of Service

Web Services User Code

1) Parameter Tampering
2) WSL probing
3) SQL/LDAP/XPATH/OS command injection
4) Virus/Spyware injection (naughty soap requests)
5) Bruteforce
6) Directory traversal
7) Data type mismatch
8) Content spoofing
9) Sessions tampering
10) Format string
11) Information Leakage
12) Authorization

The SOAP Envelope and Transport Mechanism

As we have mentioned in the previous section, SOAP is more similar to SMTP and email, rather than a traditional request/response web application. This is due to the way in which SOAP messages are transmitted. Imagine the use of a web service that transmits data from Node 1 -> Node 5 as shown below:



Next, let us imagine that the web services use only SSL to protect data being transmitted. Under this scenario anyone on Node 2, Node 3, or Node 4 would be able to read and modify the data contained within the SOAP message. SSL would protect only the data being sent between nodes.

SSL

It is possible that anyone with control of any system that is forwarding a message potentially could read and/or modify the contents for the message. Therefore, it is critical that any message that is passed between web services be secured and that this security is provided by the message itself. SSL only provides point to point security. Anyone using multiple endpoints must be able to ensure that each endpoint will not modify the contents of the message. SSL will not accomplish this task. SSL is recommended when there are only 2 Nodes involved within the web service.

Message Security

It is recommended that Message security (encrypting the SOAP body and signing the SOAP header) be implemented when there is little or no assurance of the security of the Nodes between the first and end node. Message security is difficult and time consuming to implement, but there are certain situations, as shown above, in which message security must be implemented to protect against likely attack vectors.

Web Services Fingerprinting

Web Services can be found and fingerprinted in multiple ways. Simple techniques such as Google Hacking for exposed WSDLs other services are popular ways to do this. In addition, searches for client side applications like Microsoft Silverlight XAP files can lead to exposed web services. Other ways include searching the online device search engine Shodan[13] for exposed web service management interfaces such as Axis2 or GlassFish. Lastly, using the techniques described in the Intelligence Gathering phase of the web service testing methodology, the penetration tester also should search for DISCO/UDDI directories that the client may have exposed to the Internet. Quite often clients do not realize how many existing web services they have exposed to the public.

Differences in Web Service Standards

Over the last several years there has been a departure from the traditional XML based SOAP web services. Traditional SOAP based services (depending on the application) have become a hindrance for some developers because these services can add extra bandwidth and overhead to data exchange. These issues have lead to the emergence of more RESTful[14] web services such as JSON (JavaScript Notation). REST (Representational State Transfer) services like JSON use HTTP methods (GET, POST, PUT, DELETE) for data exchange. However, SOAP based web services remain popular and are found being used for commercial as well as custom created applications. Some examples of SOAP web services used on a large scale include Amazon EC2[15], PayPal[16] and Microsoft Azure[17]. The bottom line is that SOAP based web services generally offer more complex data structures than their REST counterparts regardless of latency and performance issues.

The Importance of Web Service Management Interfaces

Any vulnerability in a web services management interface could allow an attacker to gain control of the system and, in turn, all of the web services that already have been deployed. Often times penetration testers find web service management interfaces exposed to the Internet, and that the security of these interfaces are not configured properly.

One common method used in traditional penetration testing is to use default, weak, and/or reused passwords to gain access. Unfortunately, many vendors do not enforce complexity requirements but instead try to allow for ease of use. In 2010, it was found that Axis2, a common web services interface, had been included within the SAP BusinessObjects product. The Axis2 component was not installed by default, but was required for anyone who wanted to leverage the web services components of SAP BusinessObjects. The SAP module (dswsbobje) included a default Axis2 instance along with the default username and password (admin/axis2). The default username and password allowed anyone who knew this information to log in to the system and completely compromise the default and all web services that were included.

In 2010, a Metasploit module[18] was created for this attack using the default user name and password. This interface was used to deploy a malicious web service and provide remote code execution.

New Web Services Threats

Microsoft Silverlight provides new and interesting threats to the web service landscape as Silverlight applications can use either SOAP based or RESTful services depending on the bindings used. Silverlight applications are run by the client browser and interface with WCF (Windows Communication Foundation[19]) web services. An attacker simply can disregard the client application and interface directly with the WCF services. Silverlight XAP applications also are easy to decompile using tools such as Silverlight Spy[20] to determine how the application works with the web services if the attacker needs more information. Security, in the case of Silverlight applications, truly is dependent on the security configuration of the WCF services[21].

Recently, an online Wiki called WS-Attacks.org[22] cataloged many of the newer attack vectors related to modern SOAP and BPEL based web services. WS-Attacks.org was created by Andreas Flakenberg and should be referenced for attacks when following the web service methodology mentioned in this whitepaper.

It also is possible for a web client to make SOAP based requests to web services that provide that content to the web application. In the past we have seen examples of such interaction with web services by using both AJAX and Flash. Integration between web applications and web services increases the complexity of the testing process. It is critical to test web services components if the web services requests are made from the client.

IMPORTANT NOTE ABOUT SCOPING: When scoping this type of work, it is important to remember that integrated web services within web applications may increase dramatically the necessary scope. Therefore, prioritizing which components to test - and how much - will be important in ensuring a proper scope.

New Advancements in Web Services

Recently there has been an emergence of more client side technologies such as Microsoft Silverlight that utilize web services. With much of the client handling much of the heavy lifting with regard to data processing, this makes these technologies ideal.

One of the most complex tasks in dealing with web services is modeling how information is transferred when multiple web services are needed. Multiple web services within web applications are being utilized more than ever.

One method for accomplishing this is to use an industry standard known as WS-BPEL. WS-BPEL is short for Web Service Business Process Execution Language (BPEL). The goal of BPEL is to separate the business process from implementation logic. A BPEL process is an XML file that is processed at run time. Since BPEL is based using XML and XPATH, it may be possible to perform XML or XPATH injection into a BPEL process depending on the input. Testing BPEL processes traditionally is very difficult without a clear understanding of the business process. Therefore, a White Box approach to testing BPEL is highly recommended.

How Do You Properly Scope a Web Services Penetration Test?

Proper scoping as well as pre-engagement information is very important to properly execute web services testing. The following questions need to be asked prior to any web service testing engagement. This information also is included in the pre-engagement section of the web services testing methodology noted in the next section.

- What type of web service framework[23] is being used? Examples include Windows Communication Foundation (WCF), Apache Axis/Axis2, Zend.

- What type of web services are they? (ex: SOAP, REST or WCF)
- What type of data do the web services provide? What is the importance of this data from a business perspective?
- Is BPEL being used?
- How many web services are there, and how many web methods for each service exist?
- Are you able to provide any developer documentation showing the schema of the web service as well as any documentation on APIs if they are being used?
- Can you provide all DISCO/UDDIs if being used specific to any directory listing of your web service (if publicly available)?
- Does the web service use SSL?
- Does the web service use WS-Security?
- Can you provide all WSDL paths and endpoints? How many WSDL paths are there?
- Are you using non-SOAP web services such as JSON (RESTful services)?
- What type of authentication does the web service use? Examples include: None, HTTP Basic Authentication, NTLM Authentication, NTLM off of Windows (via Ado), Parameter Based Authentication, username/password as parameters (in each call, header/body, etc), custom built or other authentication, and certificate based.
- If authentication is used, will you be able to provide credentials for testing the web service?
- Does the Web Service accept attachments via SOAP requests?
- Will you be able to provide multiple sample SOAP requests that can be used to demonstrate the full functionality of the web service?
- Does the web service have a custom front end that uses the web service i.e., Java app, custom coded desktop application, Microsoft Silverlight? Can you provide the jar, XAP, or installation files?

The New Web Service Testing Methodology

The OWASP Testing Guide v3 provides the following methodology for testing web services:

- WS Information Gathering
- Testing WSDL
- XML Structural Testing
- XML Content-Level Testing
- HTTP GET parameters/REST Testing
- Naughty SOAP attachments

- Replay Testing

Through our research we are modifying this methodology from a high level to follow the PTES[24] (Penetration Testing Execution Standard). PTES still is in alpha development; however, PTES is being discussed throughout the security community as the standard around what a penetration test is and the associated testing methodology. This makes for an ideal foundation for our revised methodology. We are breaking out the new methodology as follows:

- **Pre-Engagement Interactions**
    - Provide scoping questions to client
    - Analyze answers from the client to determine scope/threats
    - Define goals
    - Determine type of assessment (Black, Grey, or White Box)
    - Establish lines of communication
    - Define rules of engagement
    - Determine if highly customized testing will be required and adjust scope
- **Intelligence Gathering**
    - Web Service Information Gathering (active and passive)
        - Identify WDSLs and associated locations
            - WSDL Google Hacking
                - filetype:asmx
                - filetype:jws
                - filetype:wsdl
            - Perform WSDL Enumeration
                - inurl:asmx?wsdl
                - inurl:jws?wsdl
                - inurl:wsdl
        - Identify DISCO/UDDI
            - inurl:Default.VSDisco
            - inrul:default.disco
            - inurl:?DISCO
        - Identify WCF (.svc)
            - Inurl:svc
        - Identify type of authentication in use (Example: HTTP Basic Auth)
        - Identify WS-Security controls
        - Identify client applications that might use the web service (Silverlight)
        - Identify RESTful services if being used
        - Gather authentication credentials
        - Gather sample valid SOAP requests from the client and associated API documentation
            - Prepare SOAPUI, custom SOAP client with test SOAP requests
            - Configure web proxy (BurpSuite)
        - Identify web services interfaces exposed on the Internet
            - Shodanhq: glassfish
            - Shodanhq: axis2
            - Inurl: axis2-admin

- **Threat Modeling**
    - Based on scoping answers, what threats can be determined?
    - What is most valuable, from a business perspective, related to the data used by the web service?
    - Tester should outline threat scenarios to determine realistic attack vectors
    - Determine if threat scenarios are within the scope of the project
- **Vulnerability Analysis**
    - Perform authentication testing of the web service (Example: HTTP Basic Auth Brute Force)
    - Perform transport layer testing if in use (SSL, confidentiality, and integrity checks)
        - SSL Certificate Testing
            - What version of SSL is being used?
            - Self-signed cert and other SSL cert testing
        - WS-Security testing
    - Perform testing on web service management interfaces (Example: Apache Axis/Glassfish default credentials/no credentials)
        - Metasploit Web Service Management Aux Modules
    - Determine if there are existing vulnerabilities for the web service framework
        - Research vendor support site
        - Research ExploitDB for working exploits
        - Search Metasploit Framework
    - Analyze any client application utilizing the web service
        - Silverlight XAP
            - Determine how Silverlight uses the web service
            - Analyze with Silverlight Spy/Watcher plugin for Fiddler
            - Determine insecure domain references (crossdomain.xml)
            - Does Silverlight restrict JavaScript access?
        - Fat Client Application
        - Java/.NET Client or Custom
- **Exploitation**
    - Perform XML Structural Testing
        - Fuzz XML methods via BurpSuite for XPath Injection, OS Command Injection
        - Assess that parameters are being used to validate data
        - Test for Denial of Service
            - XML DOS/Flooding
            - Oversized Tags (XML Attribute,Namespace)
            - XML Entity Expansion
            - Utilize WS-Attacks.org
        - Test for WS-Address Spoofing
            - Utilize WS-Attacks.org
        - Utilize new Metasploit WS testing module
    - Perform XML Content-Level Testing
        - Fuzz content via BurpSuite for SQLi, XSS, XPath Injection
        - Test for Buffer Overflows
        - Test for XML Entity Reference Attack
            - Utilize WS-Attacks.org
        - Test for Denial of Service

                - Oversized SOAP Header/Body/Envelope
                - SOAP Parameter DOS
                - XML Encryption – XSLT/Xpath DOS
                - XML Signature – Key Retrieval/Transformation DOS
                - Utilize WS-Attacks.org
            - Utilize new Metasploit WS testing module
        - Perform HTTP GET parameters/REST Testing
            - This may be no different than traditional Grey Box testing
            - Determine differences and how they relate to the web service
        - Perform SOAP attachment testing
            - Test oversized/undersized attachments
            - Specify different content types (binary/non-binary)
            - Can MSF payload attachment be uploaded (can we get shell)?
        - Perform Replay/MiTMTesting
            - Based on threat model results, is this a feasible attack?
                - Example: Is SSL being used? Is session hijacking a concern?
                - Is XML Signature or XML Encryption used?
                    - Test for Signature/Encryption Redirect
                    - Utilize WS-Attacks.org
            - Utilize Wireshark to gather packets TCPRelay/Custom scripts to replay packets
            - Check for web service session tokens, nonces with MAC addresses, or Time Stamping.
                - Custom session management should be checked for entropy using BurpSuite's Sequencer tool
            - Test for SOAPAction Spoofing
                - Utilize WS-Attacks.org
            - Test for other MiTM Attacks
                - Message Tampering
                - Routing Detour
                - Metadata Spoofing
                - WSDL Spoofing
                - WS-Security Spoofing
                - Utilize WS-Attacks.org
        - Perform BPEL Testing
            - Follow testing guidelines from WS-Attacks.org
            - If web service uses BPEL, test for:
                - BPEL Instantiation Flooding
                - BPEL Indirect Flooding
                - BPEL State Deviation
- **Post Exploitation**
    - Got shell? Perform pillaging if goal requires it
    - Clean up artifacts
    - Prepare screen shots and document findings for reporting
    - Document all steps of test
    - Inform client of completed test
- **Reporting**
    - Provide client steps showing exploitation

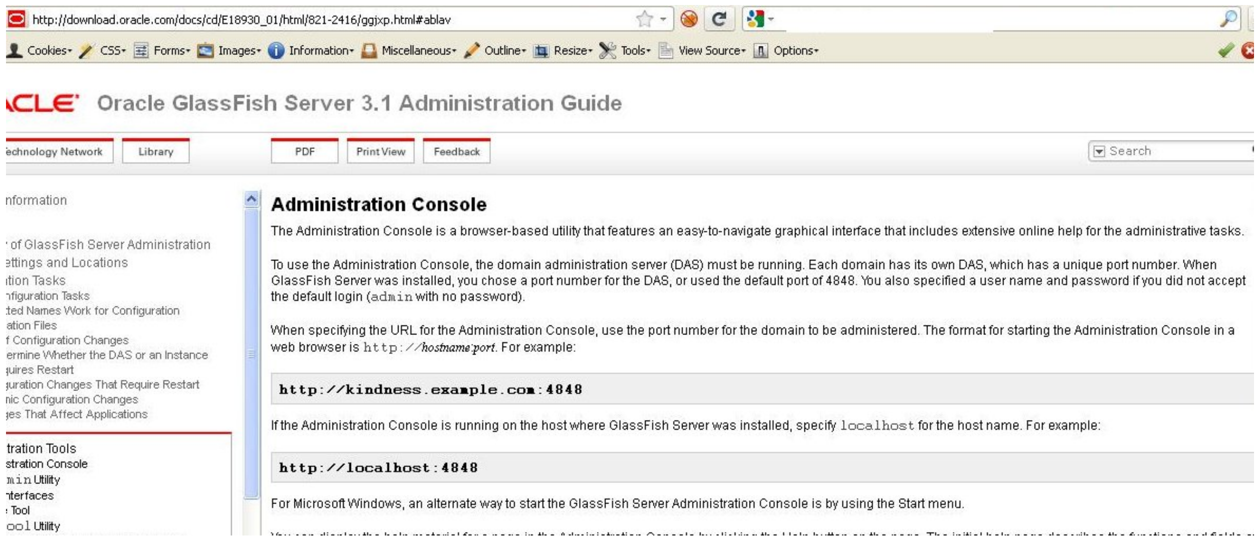
        - Client should have enough information to be able to replicate everything you exploited
    - Prepare report
    - Inform client of report delivery date
    - Schedule debrief

More details to this methodology (including testing examples) will be included in the web services testing section of the upcoming OWASP Testing Guide v4.

New Web Service Testing Modules for Metasploit

In the past few years there have been several attacks released against common web services management interfaces. Recently, there have been two vulnerabilities released in Oracle GlassFish. One was information leakage, and the other was authentication bypass in GlassFish 2.x and 3.0. In addition to these vulnerabilities, GlassFish also has a documented default username and password (admin / *blank password*).

Below is a screenshot of the Administration Guide (http://download.oracle.com/docs/cd/E18930_01/html/821-2416/ggjxp.html#ablav)


http://download.oracle.com/docs/cd/E18930_01/html/821-2416/ggjxp.html#ablav
Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
Oracle GlassFish Server 3.1 Administration Guide
echnology Network
Library
PDF
Print View
Feedback
Search

Administration Console

The Administration Console is a browser-based utility that features an easy-to-navigate graphical interface that includes extensive online help for the administrative tasks.

To use the Administration Console, the domain administration server (DAS) must be running. Each domain has its own DAS, which has a unique port number. When GlassFish Server was installed, you chose a port number for the DAS, or used the default port of 4848. You also specified a user name and password if you did not accept the default login (`admin` with no password).

When specifying the URL for the Administration Console, use the port number for the domain to be administered. The format for starting the Administration Console in a web browser is `http://`*hostname*`:`*port*. For example:

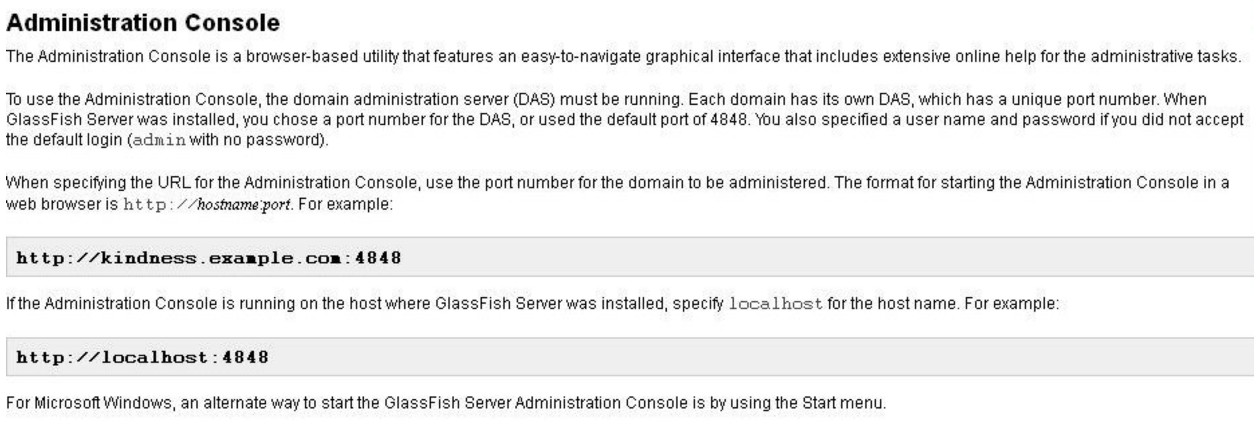
```
http://kindness.example.com:4848
```

If the Administration Console is running on the host where GlassFish Server was installed, specify `localhost` for the host name. For example:

```
http://localhost:4848
```

For Microsoft Windows, an alternate way to start the GlassFish Server Administration Console is by using the Start menu.

This section includes the username/password (“admin with no password”), as well as the unique port on which GlassFish runs (4848).



Administration Console

The Administration Console is a browser-based utility that features an easy-to-navigate graphical interface that includes extensive online help for the administrative tasks.

To use the Administration Console, the domain administration server (DAS) must be running. Each domain has its own DAS, which has a unique port number. When GlassFish Server was installed, you chose a port number for the DAS, or used the default port of 4848. You also specified a user name and password if you did not accept the default login (`admin` with no password).

When specifying the URL for the Administration Console, use the port number for the domain to be administered. The format for starting the Administration Console in a web browser is `http://`*hostname:port*. For example:

```
http://kindness.example.com:4848
```

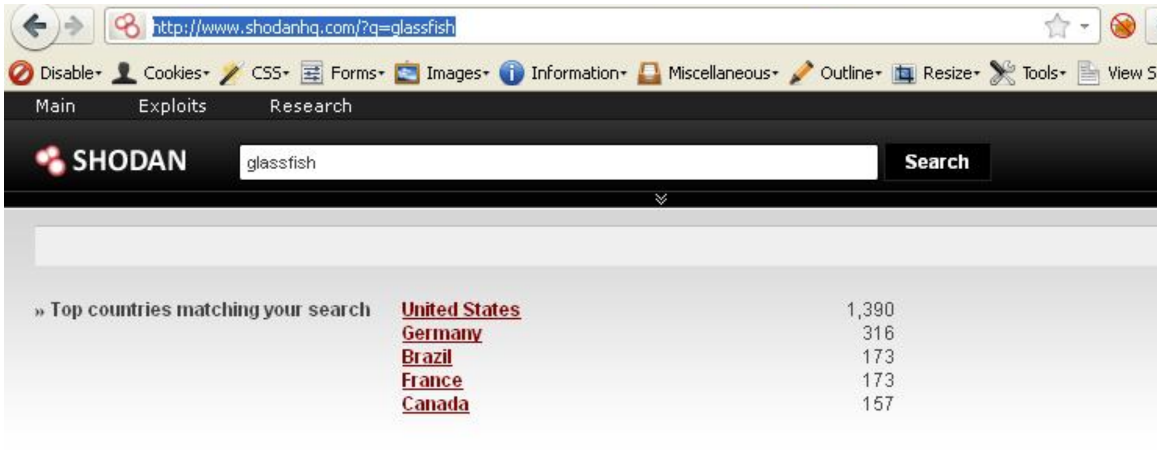
If the Administration Console is running on the host where GlassFish Server was installed, specify `localhost` for the host name. For example:

```
http://localhost:4848
```

For Microsoft Windows, an alternate way to start the GlassFish Server Administration Console is by using the Start menu.

A Metasploit module has been created that includes both the 3.x authentication bypass as well as the default username and password which will provide full access to any one system that is running GlassFish 3.x. It should be noted that the authentication bypass aspect of the Metasploit module works only on 3.0 because 3.1 was found to not be vulnerable to authentication bypass. Therefore, for 3.1 systems, the default username and password should be used to gain full control of the system.

GlassFish instances can be identified easily by using common search engines such as Google or Shodan. Here is an example of a Shodan search which shows that there are ~ 1390 systems on the Internet running GlassFish:



» Top countries matching your search

United States	1,390
Germany	316
Brazil	173
France	173
Canada	157

New Open Source Web Service Testing Environment

As stated before, the lack of test web services that are vulnerable to attack is holding back testers and toolsets. To remediate this issue, we have developed a series of vulnerable web services. This series includes many of the main vulnerabilities found in web applications but exposed through a web service. These web services are released as open source and are freely available. They take two different forms.

The first form is a set of stand-alone services. They are designed to be deployed to a web server that supports PHP. These services were built using the NuSOAP library, and each expose a WSDL to document the functions exposed. The package contains all of the installation and prerequisites needed to deploy the code.

The second form of the web services is part of a larger project. DVWA is an excellent target application set that we have extended. We created Damn Vulnerable Web Services (DVWS), which are packaged as part of the DVWA application. They are built, as are the other vulnerabilities within DVWA, with High, Medium, and Low security levels. They also work within the authentication of DVWA. This allows for the testing of web services alongside vulnerable web applications. Furthermore, we have implemented web services that implement standard features such as WS-Security. The DVWA installation is modified to include menu options pointing to the new services, and to include the documentation to help train users of DVWS.

Conclusion

In this whitepaper we have covered several attack vectors for web services. Several of these attacks have been known in the development community, but not by the security community. Due to the current gap between the development and security communities, we have focused our efforts on raising awareness of and raising the bar on the current state of web services testing.

In the future, it will be more important for the security community to work with developers to identify threat vectors that should be incorporated into the ongoing threat model for web service penetration testing.

References

[1] http://metasploit.com/
[2] http://www.dvwa.co.uk/
[3] http://www.soapui.org/
[4] https://www.owasp.org/index.php/Testing_for_Web_Services
[5] http://java.sun.com/webservices/reference/apis-docs/soap-tcp-v1.0.pdf
[6] http://msdn.microsoft.com/en-us/library/ms731092.aspx
[7] http://www.soapui.org/
[8] http://portswigger.net/burp/
[9] https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project
[10] http://www.irongeek.com/i.php?page=security/mutillidae-deliberately-vulnerable-php-owasp-top-10
[11] http://www.dvwa.co.uk/
[12] http://www.amazon.com/Hacking-Services-Charles-Networking-Security/dp/1584504803
[13] http://www.shodanhq.com/
[14] http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#wsdossoa
[15] http://docs.amazonwebservices.com/AmazonEC2/dg/2007-01-03/using-soap-api.html
[16] http://www.paypalobjects.com/en_US/ebook/PP_APIReference/architecture.html
[17] http://msdn.microsoft.com/en-us/library/gg552871.aspx#SOAPManagement
[18] http://www.metasploit.com/modules/exploit/multi/http/axis2_deployer
[19] http://msdn.microsoft.com/en-us/netframework/aa663324
[20] http://firstfloorsoftware.com/Silverlightspy
[21] http://msdn.microsoft.com/en-us/library/cc197946(v=vs.95).aspx
[22] http://clawslab.nds.rub.de/wiki/index.php/Main_Page
[23] http://en.wikipedia.org/wiki/List_of_web_service_frameworks
[24] http://www.pentest-standard.org/index.php/Main_Page