CookieMonster: Automated Session Hijacking Archival and Analysis

Joshua J. Pauli
Dakota State University
Josh.Pauli@dsu.edu

Patrick H. Engebretson
Dakota State University
Pat.Engebretson@dsu.edu

Michael J. Ham
Dakota State University
mjham@pluto.dsu.edu

MarcCharles J. Zautke
Dakota State University
mjzautke@pluto.dsu.edu

Abstract

We introduce a process-driven experiment named "CookieMonster" that can be ran against any cookie granting (i.e. session identification generation) application to test for strength of the cookie generation algorithm. The CookieMonster processes are applicable to any operating system, web server, and web application as long as session identifiers are granted to requesting client machines. Our goal is to decipher how likely session hijacking attacks may be successful strictly because of weak session identifier generation by the web application. These processes and necessary infrastructure setup can be followed for future generations of web application and web server products because of the universal approach we created. Setup for the experiment includes a web server running a web application that grants session identifiers (cookies), an attack machine running our rapid request software (Bockscar) and a database for archival and analysis of the cookies.

1. Introduction

Our Bockscar tool was developed to automate the entire process of being assigned a unique session identifier that was then analyzed to see how many consecutive sessions would need to be requested before a duplicate was assigned by the application. Our CookieMonster process is introduced in the following list executed automatically by Bockscar.

1. Log into web application with the attacker machine running Bockscar using forms authentication and receive a generated cookie from the application.

2. Retrieve the assigned cookie from the attacker machine and store in a runtime variable within Bockscar.

3. Archive the assigned cookie in a database where the cookie and timestamp are stored.

4. Destroy the session between attack computer and the web application so a new unique cookie can be issued to the same attacker machine on the next web request.

These four steps were repeated millions of times until a large sum of cookies were archived and we could then look for duplicates of cookie values, patterns in cookie generation, and other signs of weakness in cookie creation. Our initial experiment was with Microsoft's .NET 3.5 framework where cookies have an index length of 160 characters. In order to generate enough cookies, we virtualized the entire experiment both for these performance factors and to ensure a sandboxed environment where we could control what attack machines were issued a cookie from the web application.

We concluded the setup and execution of the entire experiment in a 90 day window as introduced below.

Day 1: Web server (Microsoft's IIS 7), database (Microsoft's SQL Server 2008), and web application (developed in Microsoft's Visual Studio 2008) developed.

Day 15: Bockscar development for core functionality completed to carry out steps 1-4 of CookieMonster process.

Day 15-60: Research cookie generation algorithms.

Day 60-90: Research into cookie prediction attacks.

Our paper is organized as follows. Section 2 reviews prior work related to web application security specific to cookies and session IDs and the attacks that plague these technologies. Section 3 presents a detailed discussion of the software tool that was created to execute our tests. Section 4 introduces the virtual laboratory environment that was created to execute the tests. Special attention is paid to lab changes made in order to minimize the test runtime. Section 5 covers the lessons learned from the experiment relative to the learning experience and conclusions of .NET 3.5 cookie prediction. We introduce future work areas in section 6 related to cookie prediction and the appropriate virtualized environment to best execute similar tests on different technologies.

2. Related Work

Eaton and Memon introduce how to evaluate web applications on empirical data from fielded systems with multiple client configurations including data on positive (correctly executing) and negative (incorrectly executing) instances of filed web applications [1]. Popular web applications have serious client-configuration-specific flaws [1].

Endler discussed with little ease that session IDs can be brute-forced and generating session IDs in linear or predictable manner, which includes a custom-made automated programs to be used in these experiments [2]. When a session ID can be forged or guessed, it saves time from an attacker's point of trying to brute force a session ID [2].

Juels, et. al investigated the use of active cookies vrs. ordinary cookies that are released to any server based on a domain name that can be vulnerable to pharming, or spoofing of domain names [3]. This includes a brand new type of protocol with IP-tracing integrated to defend against pharming attacks and demonstrated the use of active cookies and how they are very practical for web applications [3].

Kolsek provides detailed information about exploiting vulnerable systems to web-based applications about session management, including specifics on session IDs, session fixation, fixation attack, URL arguments, hidden form fields, and cookies [4].

Miyazaki examines the implications of three studies: 1. Longitudinal examination of the online environment from 2000 to 2007, finds that both cookie use and disclosure have increased, but the covert use of cookies is still a concern; 2. Consumers' negative reactions to cookie use are significantly reduced by a priori cookie disclosure by the visited Web site; and 3. Consumers' online experience and desire for privacy act as additional moderators of reactions to cookie use [5].

Nalneesh identifies security issues in a web application as well as areas to test by testing several common applications and provides measures to minimize the issues [6].

Noiumkar and Chomsiri present the results of 10 most popular free web-mail applications where you can session hijack them, illustrating the vulnerabilities with the use of one cookie and lists the web mail servers with the highest security level of AOL, GMX and Yahoo! Email and lower security of Gmail, Inbox Mail, and Hotmail [7].

Overcash discusses software, know as an agent, on every web server to catch malicious user based on their payload, traffic or input [8]. It includes a list of acceptable behavior and if it suspects any suspicious traffic, it will be dropped and the user will be logged out; it provides similar functionally of an IDS that filters out bad traffic in a way that the agent is a routed application in a security module [8].

Park and Krishnan go into great detail about the federated identity management system and .NET Passport framework [9]. The federated identity management system brings the service providers closer to their customers creating an increased attack vector and they identify potential security weaknesses in .NET services using Secure Cookies [9].

Samar discussed single sign-on solutions that depend on PKI/Kerberos via client side infrastructure, and relates the single sign-on to HTTP cookies and how security issues are proposed [10].

3. Bockscar Test Software

Bockscar was developed in C# in the .NET 3.5 framework to automate the process of requesting cookies, archiving these cookies, terminating the session with the server, and repeating the request for a session to the application. Bockscar is a multi-threaded application capable of three distinct functions:

1. Sending ICMP requests in such a fashion to create a denial of service (DoS) attack against a web application and web server. This is the tradition PING flood that is detected and prevented by intrusion detection systems (IDS) and intrusion prevention systems (IPS) as introduced in Figure 1.

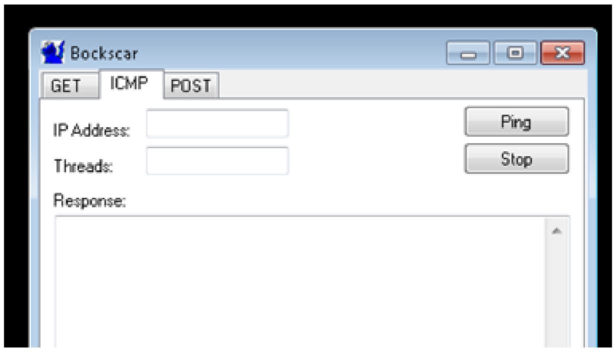


Figure 1. ICMP input tab in Bockscar.

The ICMP tab mandates the specific IP address and how many threads of ICMP requests to execute. The response from the application is also noted in the GUI.

2. Sending GET requests to a specified URL address in such a fashion to create a denial of service (DoS) attack against a web application and web server. This is traditional GET requests for a web resource such as a page or image as introduced in Figure 2 with number of threads and number of requests each thread should execute.

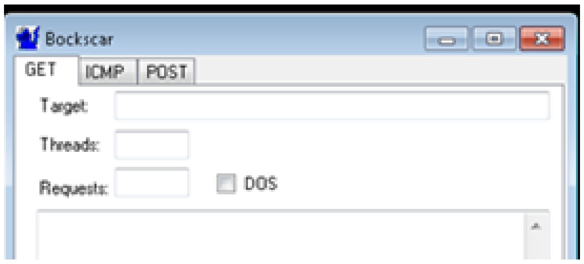


Figure 2. GET input tab in Bockscar.

3. Sending POST requests with log-in functionality via HTTP form-based authentication to retrieve unique cookies. The POST input tab is introduced in Figure 3.

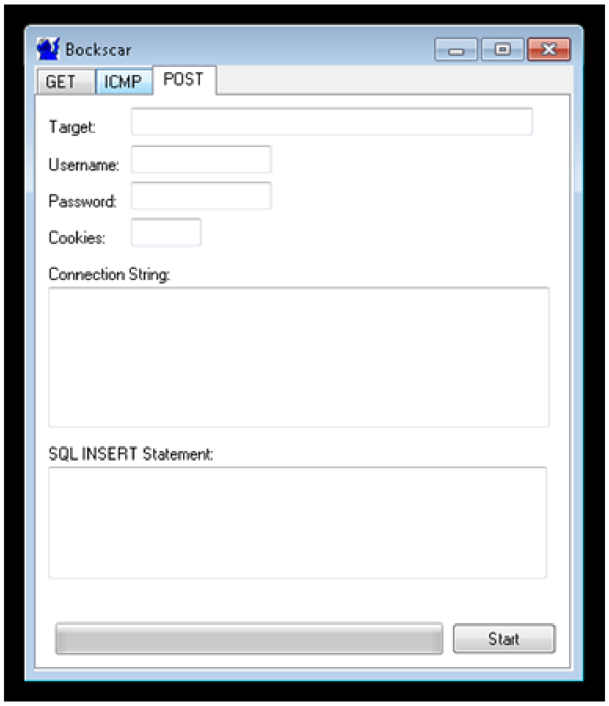


Figure 3. POST input tab in Bockscar.

These requests include parameters for username and password to complete the login procedure in addition to specifying how many cookies you want retrieved total. This is the count for how many times Bockscar should carry out the CookieMonster processes. The POST tab also allows for the exact database connection string and SQL syntax needed to insert the newly assigned cookie into the database for later analysis.

4. Virtual Test Environment

Once Bockscar was developed and the three functions tested against a live application, a testing environment had to be constructed to best suite the experiment. The environment had to allow us to restrict what client machines (attackers) were issued cookies so we would be assured that all issued cookies were, in fact, cataloged in our database as part of the CookieMonster processes.

Regardless of exact configuration, our testing environment had to include fully configured installations of: IIS 7.0 web server, SQL Server 2008 database to hold generated cookies (Cookie Jar), and our login website created in Visual Studio 2008 that issued cookies.

Our testing environment had three iterations before settling on a best practice. Each iteration was constructed virtually in a VMWare environment as introduced in the following list.

1. Version 1 of the testing environment ("Working Configuration") was the simplified proof-of-concept arrangement to make sure all hardware and software were correctly communicating with each other as introduced in Figure 4.

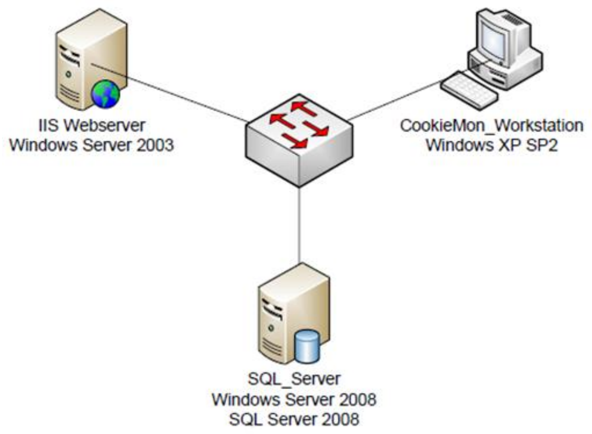


Figure 4. "Working Configuration" of CookieMonster.

This version worked as functionally specified, but because of the long index of .NET 3.5 cookies (160 characters), it was taking too long. Also, this version has the web server housing the web application and the SQL Server housing the cataloging database on the same machine.

2. Version 2 of the testing environment ("Cloned Configuration") was created to test if the exact same infrastructure could be run at same time. We created four web servers with four attacker workstations and grouped them into groups of two; one web server and one attacker workstation per group as introduced in Figure 5.

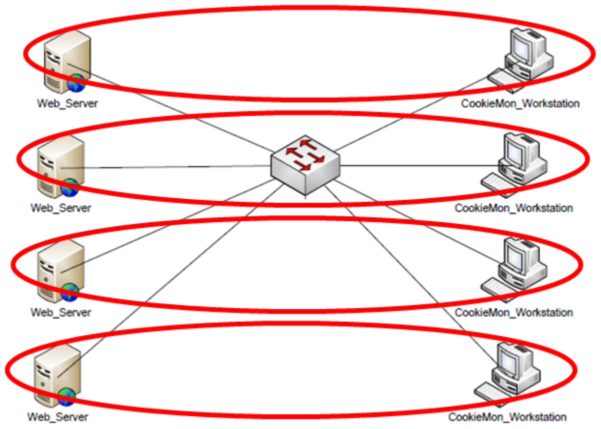


Figure 5. "Cloned Configuration" of CookieMonster.

The purpose of this cloned configuration was to test if four independent web servers would give out the same cookies in the same order. We ensured that each of the four web servers and web applications were started at the same exact time and that each of the attacker machines accessed the corresponding web application at the same time. None of the web applications served up any duplicate cookies when compared to the group of four.

3. Version 3 of the testing environment ("Running Configuration") had four web servers and four clients all running at the same time and dumping assigned cookies to a central database as introduced in Figure 6. This enabled approximately 12,000 unique cookies to be cataloged per minute.

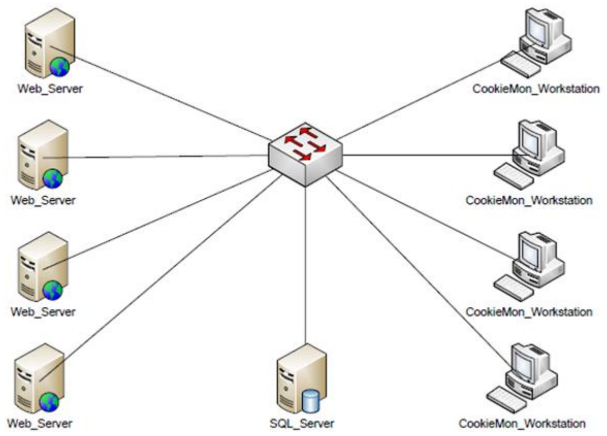


Figure 6. "Running Configuration" of CookieMonster.

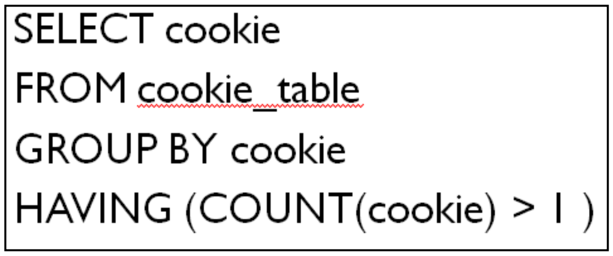
This was the finalized configuration that we used to harvest cookies looking for duplicate or non-random session identifiers from a .NET 3.5 application.

5. Lessons Learned

There are simply too many theoretical cookies generated by the current .NET 3.5 development framework even with Bockscar archiving 3,000 unique cookies per minute per the CookieMonster process as introduced in the paper.

Cloud computing and other distributed infrastructures would not provide the necessary requesting, archiving, and analysis needed to find duplicate .NET 3.5 session identifiers.

The exact SQL syntax for selecting the duplicate cookies is straight-forward as introduced in Figure 7.



```
SELECT cookie
FROM cookie_table
GROUP BY cookie
HAVING (COUNT(cookie) > 1 )
```

Figure 7. SQL syntax to retrieve duplicate session identifiers.

Because of the strength of the session generation algorithm in .NET 3.5, other session attacks can be leveraged to use these strongly generated cookies by exploiting vulnerable cookie storage such as cross-site scripting (XSS), cross-site request forgery (CSRF), and similar session fixation attacks.

Thanks for generating strong cookies – we can't predict them – so we'll just steal them from an unknowing user and use them to masquerade as that user.

6. Future Work

The Bockscar tool needs extensive needs exception handling work to be a legitimate testing tool. There were too many unexpected errors that delayed testing until a new iteration of cases could be executed.

In order for the our testing process to succeed in finding weaknesses in session identifier generation, older HTTP form-based authentication mechanisms should be investigated. This would allow proof-of-concept findings to be investigated on how to best attack current and future session identifier generation standards. The current .NET 3.5 is too robust in generating cookies to achieve any sort of success because of the vast quantities of session identifiers that were generated.

Other attacks such as a denial of service (DoS) via GET requests is also possible with Bockscar; every network defensive infrastructure blocks ICMP (ping requests), but must allow GET requests access to the webserver in order to adequately serve legitimate web resource requests. Functionality for spoofing IP address and MAC address would be necessary in order to avoid network intrusion detection (IDS) and intrusion prevention (IPS) systems.

7. References

[1] C. Eaton and A.Memon. "An empirical approach to evaluating web application compliance across diverse client platform configurations." *International Journal of Web Engineering and Technology 2007* 3.3 (2007): 227-253.

[2] D. Endler. *Brute-Force Exploitation of Web Application Session ID*. iALERT, 2001.

[3] Juels, M. Jakobsson, and S. Stamm. *Active Cookies for Browser Authentication*. Diss. University of Indiana, 2006. Bloomington: RSA Laboratories and RavenWhite, 2006. Print.

[4] M. Kolsek. *Session Fixation Vulnerability in Web-based Application*. Acros, 2007.

[5] Miyazaki. "Online Privacy and the Disclosure of Cookie Use: Effects on Consumer Trust and Anticipated Patronage". *Journal of Public Policy & Marketing* 27.1 (2008): 4.

[6] G. Nalneesh. "Assessing the Security of Your Web Applications." *Linux Journal* 2000.27es (2000)

[7] P. Noiumkar and T. Chomsiri. "Top 10 Free Web-Mail Security Test Using Session Hijacking," *Proc. of the 2008 Third International Conference on Convergence and Hybrid Information Technology*. 2008.

[8] K. Overcash. "System And Method For Detecting Security Defects In Applications". PROCOPIO, CORY, HARGREAVES & SAVITCH LLP, assignee. Patent AG06F1130FI. 18 May 2009.

[9] J. Park and H. Krishnan. "Trusted Identity and Session Management Using Secure Cookies". *Proc. of the Data and Applications Security XIX 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security.* Storrs, CT, USA. August 7-10, 2005.

[10] V. Samar. "Single Sign-On Using Cookies for Web Applications". *Proc. of the IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. 1999.