XSS Cookie Injection Covert Channel

Kyle Feeney and Daryl Johnson
Department of Computing Security
Rochester Institute of Technology
Rochester, NY USA
Email: kdf5506@rit.edu, daryl.johnson@rit.edu

Abstract—This paper describes a method of covert communication by way of HTTP Cookie injection using Cross-site scripting into vulnerable website's. Website's susceptible to Cross-site scripting can be used as a medium for covert communication using Cookies as the message carrier between two or more nodes. Due to the necessity of Cookies in Web development and the format options of a Cookie, this covert channel offers a variation of implementation techniques which can achieve reasonably high data rates.

I. INTRODUCTION

Research into covert channels is still fairly new and for that reason, defining a covert channel is difficult. Butler Lampson first used the term covert channel in his paper, The Confinement Problem [4] presenting techniques for safeguarding against information leakage in computing processes which may lead to other processes accessing the leaked data. It has sense been recognized that covert channels exist in many different technologies and are used legitimately and illegitimately. The Department of Defense’s Trusted Computer System Evaluation Criteria (TCSEC) recognizes two types of covert channels: storage channels and timing channels. Storage channels include all vehicles that would allow the direct or indirect writing of a storage location by one process and the direct or indirect reading of it by another. Timing channels include all vehicles that would allow one process to signal information to another process by modulating its own use of system resources in such a way that the change in response time observed by the second process would provide information [1]. Although there are many different definitions for what constitutes a storage channel this implementation of a covert channel, based on the TCSEC definition, would be described as a covert storage channel.

Exploration in the area of HTTP Cookie covert channels is not new either. Past work includes Huba’s HTTP Cookie Covert Channel using Google Analytic web Cookies [2]. These Cookies can be encoded to carry a predefined message which is passed to the web server in an HTTP request. Using a Man-in-the-middle style attack, a node can sniff traffic sent to the server and strip the Cookie content from the HTTP header and decode the message. The proposed covert channel in this paper will use Cross-site scripting to inject Cookies with encoded messages into website’s which do not safely validate their form data. According to XSSed [6], one of the largest archives of Cross-site scripting vulnerable sites, there are over 45,000 website’s listed which provides large a number of sites for this covert channel implementation.

II. HTTP COOKIE

Though HTTP Cookies are a controversial topic due to tracking Cookies and Cross-site scripting exploits, they are largely misunderstood and are an integral part of the Web and serve an important purpose. HTTP is a stateless protocol, meaning the protocol does not save any information about its current state or the client and it’s state is renewed with each invocation of an HTTP request. There arose a need to create a way for a website to save information about it’s state such a session information or client information. This led to the advent of the HTTP State Management Mechanism [3] otherwise known as the HTTP Cookie or juts Cookie. Originally designed and patented by Lou Montulli in the early 90’s. A Cookie contains small pieces of data passed from a website to a client’s browser and depending on the browser it may also be stored on the client’s file system. A Cookie can serve several purposes. It can be used to store information about user’s preferences or browsing habits such as past website’s visited. They can be used to collect information, otherwise known as tracking Cookies which obtain information such as the user’s network address, location, and other information. Google Analytic’s is an example of a tracking Cookie implementation. The most common use of Cookies is for storing user’s login credentials. These different uses server the purpose of making Web browsing a more intuitive and enjoyable experience. Cookies are widely used and without them many of the popular website’s would not work. For that reason, they offer an excellent vector for exploit.

The HTTP State Management Mechanism was originally documented in RFC2091 and later superseded by RFC2965 and again by RFC6265. An HTTP Cookie has 6 fields, 5 of which are optional. The number of optional fields leaves room for modifying and improving this covert channel in future development. The available fields include:

- Name - Required. The name given to the Cookie by the origin server.
- Expire - Optional. Specifies the date and time for which a Cookie should expire.
- Domain - Optional. Specifies the domain for which a Cookie is valid. This must always begin with a dot in a fully qualified format.
- Max-Age - Optional. Specifies the lifetime of a Cookie in seconds. It must be a non-negative integer. A zero
means the cookie should be discarded immediately. If specified along with Expire, the Max-Age attribute should take precedence.

- Path - Optional. Specifies a subset of URL’s for which the Cookie applies.
- Secure - Optional. Indicates that securing the Cookie is in the best interest of the user agent.

According to RFC6265, the general implementation considerations for browsers should at least include:

- At least 4096 bytes per Cookie (as measured by the sum of the length of the Cookies name, value, and attributes).
- At least 50 Cookies per domain.
- At least 3000 Cookies total.

This covert channel will use HTTP Cookies as the message carrier. Therefore, these considerations are important as they will be used to derive the potential data rate this cover channel offers. This will be laid out in the Cover Channel Method.

III. CROSS-SITE SCRIPTING

Cross-site scripting, also referred to as XSS remains the largest vulnerability on the Web. According to Symantec, in 2007 over 84% of web exploits were related to Cross-site scripting attacks and still in 2012 Cross-site scripting remained the most common vulnerability on the web [6]. The vulnerability is typically the result of poor development and configuration. Website’s which do not properly validate input on their form fields are susceptible to Cross-site scripting. An attacker can use JavaScript to manipulate the website to act differently than intended. One way to test for Cross-site scripting is to simply browse to a predefined website such a message board or Blog site and enter the following script as seen in an OWASP example [5].

```javascript
<script type="text/javascript">
alert("<script src="http://example.com" /></script>"
</script>
```

Fig. 1. Cookie Grabber Attack Example

This script will send the Cookie to the evil.php page where the PHP script can use it for whatever it wants. If the website accepts this as input, when the HTML is rendered it will not display what is in the script tags, however visiting users of the website will now have their Cookie information passed to an attacker. Because this vulnerability is so frequent, it makes for an excellent means for persisting our communication over the network.

IV. METHOD

The design of this covert channel is based on three entities. Firstly, the web server hosting a website which is vulnerable to Cross-site scripting. The server can either be owned by the communicating parties or it can be a third party server. Second is the sender which runs a simple Perl script to generate the message and encode it into a Cookies content field using the JavaScript `document.Cookie` property. This property acts as the `Set-Cookie` line in an HTTP header. Using this method, there is no perceivable difference between legitimate Cookies passed by the server and the ones we inject. Lastly, the receiver which runs a Perl script to parse all Cookies received and stored on the local file system. The receiver script then verifies whether a message exists and if it does it decodes the message. This method allows for bi-directional communication if both nodes are running the sender and receiver scripts. The figures below depict the message flow from sender to receiver and then from receiver to sender, respectively. The message is first encoded and injected by the sender. The receiver will refresh the page at a specified interval invoking an HTTP request. The Cookie is then passed to the receivers browser in the Set-Cookie header of the HTTP response. The receiver can now decode the message and acknowledge the message by injecting its own Cookie containing the Expire date of the original Cookie incremented by a single second causing the senders Cookie to be overwritten with the acknowledgement.

Fig. 2. Message from Sender to Receiver with Acknowledgment

Fig. 3. Message from Sender to Receiver with Acknowledgment

In our case, we are not concerned with legitimate Cookies passed to the browser from the website, but those that were injected by the sender into the website’s database using our Cross-site scripting attack. The Expire field of the Cookie tells the browser whether or not it needs to accept the new Cookie and overwrite the old. A browser will not accept two Cookies of the same name value from the same website instead it will overwrite an older one with one containing a later expire date. With this in mind, we can produce a TCP like communication between sender and receiver where
the receiver acknowledges each Cookie received before the sender sends the next of the same name value. When the receiver gets a Cookie from the sender, the receiver could theoretically pass back the Expire date to the sender in the content field, telling the sender that it has successfully received that message. The sender can then increment the time on the next message in the Expire field and send the next message, again waiting for an acknowledgement. This way the sender and receiver can be assured the message is being passed in its entirety. If the sender receives an acknowledgement for a time less than the last messages Expire value than the sender can retransmit the message starting from the Expire value given by the receiver. This method presents a reasonable data rate but involves outside variables which can influence the data rate. Given the above example, the bandwidth, measured in bits per second, of a single message sent can be measured as such:

Given $S$ representing the size of the Cookie in bytes. If $s$ represents the number of seconds needed to inject the Cookie into the website, then $t_1$ represents the bit rate.

$$t_1 = \frac{(S \times 8)}{s}$$ (1)

As an example, a Cookie of max size 4096 bytes requiring 5 seconds to inject would have a bit rate of 6553bit/s for injecting the Cookie into a website.

$$\left(\frac{4096 \times 8}{5}\right) = 6553\text{bit/s}$$ (2)

If the receiver is set to automatically refresh the website at a specified interval of say 15 seconds, then $t_2$ can represent the download bit rate. Let $C$ represent the size of our injected Cookie in bits and $n$ represent the sum of all bits for additional Cookies passed by the website. Let $s$ represent the time in seconds needed by the website to push the Cookies to the browser.

$$t_2 = \left(\frac{(C + n)}{(s + (15/2))}\right)$$ (3)

The maximum theoretical throughput in bits per second can be calculated by subtracting $t_2$ from $t_1$.

$$\left(\frac{4096 \times 8}{5}\right) - \left(\frac{32768 + 0}{5 + (15/2)}\right) = 3932\text{bit/s}$$ (5)

To effectively test this formula, both the sender and receiver modules should be automated. Currently, testing this theory requires manually injecting the scripts into the website while the receiver manually refreshes the page. Parsing the cookies is automated, however the receiver would need to manually inject an acknowledgement Cookie back into the website. Further development of this covert channel would include improving automation of both the sender and receiver modules.

V. DETECTION AND PREVENTION

The implementation of this covert channel utilizing Cross-site scripting and HTTP Cookies does face some setbacks which have to be considered. Firstly, as the entire transmission relies on an Cross-site scripting vulnerable website as the channel medium and Cookies as the message carrier, it is critical that the sender and receiver agree on a vulnerable website to use, and that both the sender and receiver use a browser which has Cookies and Scripting enabled. From a security perspective, administrators can prevent this covert channel from being successful by disallowing Cookies to be pushed to the browser which can kill the message carrier portion of this channel. Furthermore, a well developed and configured website would prevent the sender and receiver from injecting scripts into its pages. This can be done by correctly validating data persisted through the website to its storage.

As previously defined, a successful covert channel hides the fact that two or more entities are communicating and that both the sender and receiver should not be identified even if the channel were to be. One flaw in this implementation is the use of injecting scripts into the website’s form fields, which are then persistent on a storage medium of some sort. This means that the script itself resides local on the server in a database or flat file. This could also mean that the sender may be logged by the web server when data is posted to the website. The receiver or receivers however, can remain undetected because by merely visiting the page the receiver remains ambiguous to the server. Only a receiver who knows that a Cookie has been pushed to their browser and who knows how to decode the message would find the Cookie useful. Future development may include techniques for safeguarding the senders identity.

VI. CONCLUSION

In conclusion, though the detection and prevention techniques may hinder the success of this channel, it is still a viable covert channel if implemented correctly. Knowing that a sender may be logged we can use techniques such as spoofing an IP address to protect the senders identity. Enabling Cookies and scripting in the browser is a simple obstacle to overcome because it would be safe to assume the sender and receiver have access to the machines for which they are using for communication. Lastly, as noted covering Cross-site scripting; due to the number of Cross-site scripting vulnerable sites, there would be no shortage of communication mediums to choose from.

REFERENCES
