«FYRM»

Cross Site Scripting Anonymous Browser

Black Hat DC 2009

Matthew Flick
FYRM Associates
matt.flick@fyrmassociates.com

Abstract

The recent attacks[1] against onion routing[2] tools highlight the need for a new method to maintain some degree of anonymity. The Cross Site Scripting Anonymous Browser ("XAB") addresses this problem for Web browsing by forcing unknowing victims—user browsers—to retrieve targeted Web pages via Cross Site Scripting for the attacker.

True anonymity is not likely to be possible, so any anonymizing algorithms and tools simply attempt to increase the difficulty for others to discover the source, destination, and transmitted content. The XAB improves the user's anonymity by moving the Web server request and response one step away, i.e. to the victim's browser. There are numerous programming techniques that can also be used in order to mask the target and request/response data as well, thereby making it more difficult to track the user's actions.

Synopsis

The goal of the XAB is to provide a tool for attackers to make and receive HTTP requests and responses that will not be easily traced to the attacker. The concept of anonymous browsing via Cross Site Scripting is rather simple:

1. Inject HTML into miscellaneous victims' browsers via Web site(s) vulnerable to HTML injection.
2. Retrieve targeted Web page(s).
3. Send the retrieved HTML and other content to the attacker's system.

The implementation of this concept is, of course, much more complex. The known limits of JavaScript and the security controls implemented by Web browsers drive much of the complexity of the XAB. Specifically, the cross domain firewall restricting content access via JavaScript prevents current browsers from acting alone as a XAB and provided the greatest challenge in developing the tool. This obstacle was followed closely by handling non-text data; this paper will focus on images and leave video and other content for future research.

Design

The XAB is comprised of the following components:

- XABAttacker: attacker's system running Web server listening at (e.g.) http://xab.dyndns.org
- XABProxy: proxy Web server scripts used to fetch target content listening at (e.g.) http://xabproxy.somesite.org
- VulnerableSite: Web server that is vulnerable to HTML injection listening at (e.g.) http://vuln.site.com
    - If VulnerableSite can be used to host and execute the XABProxy server scripts, then the two components can be combined
- Victim: any user that receives the XSS code stored at VulnerableSite
- Target: any URL the attacker wishes to make an anonymous request for, e.g. http://www.fyrmassociates.com/about.htm

Note: These component names will be used throughout the remainder of this document.

The XABAttacker must register a domain name (or use a static IP address) for the victims to contact. It is noted that this requirement may negate the anonymity gained by the XAB, but it is left to the reader to determine how to implement this component anonymously. Alternatively, long DHCP lease times for cable modem providers may enable the use of the same dynamic IP address for an extended period of time.

These components are illustrated in the diagram below along with the inter-component communications displayed as numbered boxes connected by lines. The steps to retrieve content from Target—corresponding to the numbered boxes—are explained on the following page.

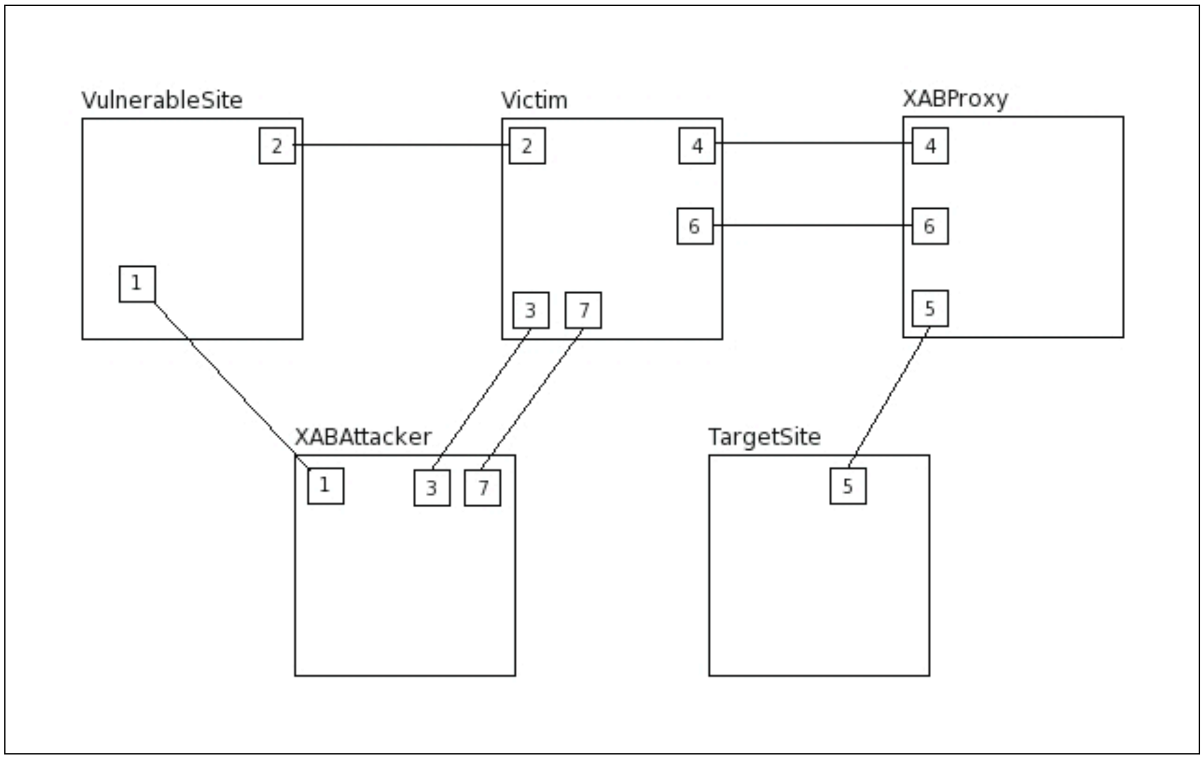

Figure 1: XAB diagram

Step-by-step instructions to implement the XAB:

1. XABAttacker uploads initial payload to VulnerableSite
    a. This can be accomplished via any system so the attacker's system does not actually "touch" VulnerableSite
    b. XABAttacker can also use Tor for this, but that may not be the best idea given the recent "one cell" attack
2. Victim visits VulnerableSite and parses HTML including our initial payload, which requests additional script from XABAttacker
3. XABAttacker sends second payload to Victim; this payload includes:
    a. XABProxy location
    b. Target URL(s) to be retrieved
4. Victim makes another script request to XABProxy (Target included in the request)
5. XABProxy requests HTML and other content from Target URL
6. XABProxy encodes all content as string and sends script back to Victim that includes:
    a. Code to send data back to XABAttacker
    b. Data string (encoded version of Target contents)

7. Victim forwards data to XABAttacker
8. XABAttacker resends second payload to Victim with new Target (optional)

XAB vs. Onion Routing

The XAB should not be seen as a threat to onion routing's reign as the anonymity king, despite other recent threats to the throne. Admittedly, the XAB is limited in its current state to simple Web browsing and does not address any other types of connections. Other limitations and weaknesses will be discussed below, but it is recommended that this tool be used in conjunction with other tools and techniques if a high degree of anonymity is desired.

Using XAB and an onion routing network provides the attacker the ability to replace his own system with a semi-anonymous proxy (XABProxy) as the source in that network, adding a nice degree of separation. Another degree of separation is provided by the many victims acting as proxies between the XABProxy and the true source (XABAttacker).

Technical Difficulties

Perhaps the most daunting technicality to overcome is the Web browser security control preventing JavaScript in one domain to access (read or write) the content of the DOM in another domain. Since the ultimate goal of the XAB is to have minions retrieve content from any domain/Web server of the attacker's choosing and then send that content back to the attacker, a workaround is needed. The next section will discuss three approaches to enable such interaction between attacker and victim, including the use of a proxy that is used in XAB version 1.0.

The second major obstacle involves handling non-text data. Suffice to say browsers do not treat images, audio, and video the same as simple HTML text and markup. A browser can easily retrieve an image from any domain accessible to the host; however, there is no known method to parse that content's data into a transmittable code—or simply retransmit the content—to the attacker's system via JavaScript. This complication pushed the proxy approach into the lead during development of the XAB. By breaking out of the browser/client-side JavaScript sandbox, the XAB enables many more options for better anonymity by allowing more tools and techniques to be used in conjunction with it without sacrificing much in the prerequisites arena.

The last problem to be mentioned in this paper is related to the implementation of HTTP verbs other than 'GET'. Although uncommon, some Web servers will only accept specific HTTP verbs for specific requests; this functionality is typically reserved as a security control for highly sensitive functions within the application, such as authentication or payment submission. In the interest of reading time, the uninformed reader is strongly recommended to search for "xss http post" and examine the results.

Workarounds

Through extensive browser research, there appear to be only two possible techniques available for use in the XAB to allow browsing of any domain: a proxy and DNS spoofing. Another technique will soon be available for use against the most up-to-date browser users and will be described here as well, but does come with its own limitations.

The first technique is DNS spoofing. One may ask, "Why would I go through the trouble of spoofing DNS records just to semi-anonymously browse a few Web sites"? It's an excellent question that is left to the reader to decide. Rather, this paper will focus more on the technical options available to the user. The design of this technique includes the Victim's DNS server(s) in addition to the same components described. Here are the required steps:

1. XABAttacker spoofs the Target domain name to redirect traffic to the XABAttacker system. The Web server running on XABAttacker system must also be configured to accept connections for that domain.
2. XABAttacker loads initial payload to VulnerableSite.
3. Victim retrieves initial payload from VulnerableSite.
4. Victim makes request for content from the Target domain, which is sent to XABAttacker system due to DNS spoofing.
5. XABAttacker sends full payload to Victim that includes a second request for the attacker's desired URL from the Target domain.
6. The full payload executes a denial of service attack against the browser so that the user must restart the application. This action will forcibly reset the DNS pinning of the browser.
7. XABAttacker resets the DNS settings to the original (true) state.
8. When Victim visits the VulnerableSite again, the initial payload is executed a second time but the full payload is retrieved from local browser cache. The second request is sent by the browser for content in the Target domain, which is sent to the real Target server.

9. Victim encodes retrieved data and transmits to XABAttacker via HTTP GET request(s).

The "DNS rebinding"[3] technique described here is simple enough to recreate in a controlled test environment, but is likely to fail or become far too noisy in a real world environment (especially for our simple goal of anonymous Web browsing). Whereas this approach successfully enables access to the Target content, it fails to address the second problem described in the previous section: how to handle non-text content. For these reasons, the proxy approach becomes the logical choice for implementation.

The basic XABProxy is a simple request and response forwarder; in other words, a proxy. It receives a request for script content that includes a target URL, retrieves and encodes the content from the target Web server, and responds with the encoded content and instructions for the requester to pass the data to the final destination (i.e., the XABAttacker). However, since the XABProxy is implemented as server-side technology, its functionality is no longer limited to client-side JavaScript. This not only allows cross domain content access, but also affords the proxy code author the ability to implement such potential functionality as non-text content encoding, response splitting, victim chaining, and even data encryption or obfuscation to further hide data traffic to/from the attacker's system.

The prerequisite list for the XABProxy is short and is likely to be available at any one of thousands of appropriately vulnerable Web servers in the world. In its simplest form, the attacker must configure the Web server to execute server-side scripts. The desired functionality can very likely be implemented in any such language/framework, so portability is not a problem. If the VulnerableSite can be exploited to host this functionality in addition to the initial payload, then the attacker can merge the two components and remove the need to identify or create a separate XABProxy host. Of course any Web server allowing users to upload executable script will suffice, but limiting the number of involved hosts will decrease latency as well as the probability of the attacker being identified as the source.

Note: The XABProxy component may be eliminated or replaced in the event browser bugs are discovered or user error permits cross-domain requests.

The final technique involves the new "Access-Control-Allow-Origin" HTTP response header[4] and an expected response to its widespread implementation. In relation to the XAB, this proposed mechanism would allow the Target Web server (the real one) to specify which other domains may access its content. Although this mechanism has been on many developers' wish lists for a long time, it is likely to expect a reasonable percentage of Web site/application developers to insert a quick "`Access-Control-Allow-Origin: *`" into the standard operating procedures of the Web server, either for ease of implementation, as a forgotten test setting, or due to a lack of proper training and oversight. It is very likely that Web servers implementing the new header in this fashion would allow XAB payloads to request, access, and forward the server's content to the XABAttacker without a proxy.

Enhancements

With the basic functionality completed, the focus now shifts to improving the XAB by adding additional functionality, reconfiguring components, or incorporating other options for various goals. The following list serves as a discussion board as well as implementation plans for future XAB versions (in no particular order).

Note: The following functionality has not been tested and should be considered only theoretically possible until proper testing can be completed.

- Response splitting – Content filters can be quite bothersome, especially when trying to research information security vulnerabilities and attacks (i.e. the "hacking sites"). Some quick modifications to the XAB code can instruct the victim browsers to only send a particular segment of the Target's content to XABAttacker. With this approach, the monitoring systems will have to dynamically identify the splitting algorithm and the connected pieces, and then reconstruct the original content. A slightly more advanced customization can ensure no single victim transmits multiple segments of the same content.
- Request splitting – For XAB users that wish to never transmit a whole target URL akin to the response splitting technique, the XABProxy and Victim components can be modified to handle split Target URLs. The XABProxy will act as the manager in order to recombine the split URL and redistribute the full URL when all segments are received. Note: This technique increases the likelihood of errors due to Victim users browsing away from the VulnerableSite, exiting the browser, or severing the connection for some other reason.
- XABAttacker splitting – Establishing multiple recipients for content retrieved by victims can add yet another layer of separation between the true source of the request and response destination. The candidates for this option would include hosts similar to the XABProxy list of candidates since Web server functionality is required.
- Multiple requests – The XABAttacker component can be configured to send

multiple requests in single or multiple payloads that instruct the browser to make the requests simultaneously or sequentially. The Victim and XABProxy can instruct each other on which responses should be sent back to the XABAttacker: none, all, the final response only, or any error response.

- Binary data transfer – Transmit non-text data from XABProxy and/or Victim to XABAttacker. Images and audio have been shown to be transmittable as encoded strings and interpreted correctly in client-side technology using the data URI scheme [5,6]. With similar tools available to the XABProxy component, the XAB can be [somewhat] easily configured to handle HTML text and markup, images, and audio files. Larger and more complex content, such as video, will very likely require a more complex solution that does not yet exist.
- Victim [mind] tricks – One yet-unmentioned requirement for the XAB is the page containing the initial payload (from VulnerableSite) must remain resident in the Victim's browser until at least one round of the process is completed; otherwise, no content will be sent to the attacker. Although every XAB Victim payload will be hidden from normal browser view, there is no measure available to prevent the user from destroying the window and breaking the process. However, the social engineering empowered XAB user could implement any number of techniques to convince the user not to destroy the payload window.
    - Authors of other Cross Site Scripting exploit tools [7,8] have implemented useful techniques that attempt to maintain control of the browser and will be incorporated into future versions of the XAB.
- XMLHTTPRequest and Access-Control-Allow-Origin – When the major browsers and Web servers begin implementing the new cross domain access control HTTP response header (e.g. for XMLHTTPRequest or "XHR" calls), the Victim can be configured to attempt an XHR call directly to the Target, bypassing the XABProxy. This approach will, of course, negate the benefits of the proxy approach mentioned above, but it can be implemented as a per-request or per-target option for the XAB user.
- Data encryption – If Rijndael has already been implemented purely in JavaScript [9], then surely the advanced and JavaScript-savvy XAB administrator can implement some form of data encryption to better protect data communications from analysis. However, this technique may not provide the intended return on investment due to the large amount of JavaScript code required for implementation when compared to other techniques listed above.
- Onion routing XABProxies – With an army of established and known proxies, an XAB infrastructure can be created akin to the Tor network, even implementing a similar (or the same) onion routing algorithm within the

proxy army. Or better yet, just install Tor on the XABProxy. The benefit of using the XAB in this fashion is the extra degree of separation between the true source (attacker) and the network of onion routers, courtesy the pseudo source (victims).

Known Weaknesses

As stated above, the goal of XAB and other anonymizing tools is to increase the degree of one's anonymity. No tool is perfect and the XAB has its own share of weaknesses:

- Registering XABAttacker – The likely targets for VulnerableSites are those Web servers vulnerable to persistent Cross Site Scripting, which means the attacker must store the XABAttacker system's publicly accessible address—DNS name or IP address—in the attack. If/when the payload is identified by law enforcement authorities or other interested parties, the necessary ISP records will likely lead them to the attacker's location. Some potential solutions include:
    - Use some common techniques for hiding a system's identity: change the MAC, use free and open networks, use a live CD distro (e.g. BackTrack), use public access systems, etc.
    - Run the XABAttacker component from another source, similar to the XABProxy component. The attacker could then simply interface with the XABAttacker as a client.
- Targeting XABProxy – If an attacker-controlled system has been configured as the XABProxy component, then identification or compromise of this system could lead back to the attacker, thereby negating the benefits afforded by the victims. This potential weakness may drive the attacker to identify a VulnerableSite that can act as the XABProxy in addition to simply an initial payload distributor. Free CGI hosting Web servers may also be targeted as XABProxies, though configuring Tor on these systems may be more difficult.
- Malicious victims – Though unlikely, it is possible for a victim to inject malicious HTML or execute other attacks against XABAttacker since the Victim component has access to all of the retrieved content. Mitigating controls for this weakness would include all of the usual steps used when attempting to safely browse in a normal user fashion.
- Corporate firewalls – Corporate network environments are largely configured with security controls preventing any internal, non-administrator user from adding a publicly accessible server to the network. Adept attackers may be able to exploit the many vulnerabilities plaguing today's armadillo corporate networks and install the XABAttacker component; however, the less-skilled (or simply more reserved) attacker must host the XABAttacker system from some other location.

Conclusions

The goal of this paper and the XAB is to illustrate techniques that may be used to increase a user's degree of anonymity, specifically when browsing Web content. Onion routing—and specifically Tor—will likely remain the first tool of choice in the anonymous network user's arsenal. However, additional tools and techniques will become even more crucial as additional attacks targeting onion routing networks are discovered. The XAB demonstrates one such technique by turning unknowing Web browsers into drones that retrieve content for the XAB user. Similar to the defense-in-depth approach for protecting sensitive data and systems, it is recommended to implement an anonymize-in-depth design for protecting an individual's identity.

References

[1]Fu, Xinwen. "Black Hat DC 2009 Briefings Speaker List." One Cell is Enough to Break Tor's Anonymity. 15 Jan 2009. Department of Computer Science, University of Massachusetts Lowell. <http://www.blackhat.com/html/bh-dc-09/bh-dc-09-speakers.html#Fu>.

[2]"Onion routing." 5 Jan. 2009. Wikipedia. <http://en.wikipedia.org/wiki/Onion_routing>.

[3]"DNS rebinding." 26 Dec. 2008. Wikipedia. <http://en.wikipedia.org/wiki/DNS_rebinding>.

[4]van Kesteren, Anne. "Access Control for Cross-Site Requests." . 12 Sept. 2008. World Wide Web Consortium. <http://www.w3.org/TR/access-control/>.

[5]Resig, John. Embedding and Encoding in JavaScript. 9 Apr. 2008. <http://ejohn.org/blog/embedding-and-encoding-in-javascript/>.

[6]Masinter, L. "The data URL scheme." IETF RFCs. Aug. 1998. IETF. 5 Dec. 2008 <http://tools.ietf.org/html/rfc2397>.

[7]Rager, Anton. XSS-Proxy. 9 Feb. 2005. <http://xss-proxy.sourceforge.net/>.

[8]Mavituna, Ferruh. "XSS Shell." . 11 Oct. 2008. Portcullis Labs. <http://labs.portcullis.co.uk/application/xssshell/>.

[9]Veness, Chris. "AES Advanced Encryption Standard." . 1 Aug. 2008. Movable Type Scripts. <http://www.movable-type.co.uk/scripts/aes.html>.