Launching Attacks using Steganography

Suba Sah
EECS Department
University of Toledo
Toledo, USA
suba.sah@utoledo.edu

Dr. Ahmad Y. Javaid
EECS Department
University of Toledo
Toledo, USA
ahmad.javaid@utoledo.edu

***Abstract*—An increase in the online presence of the websites worldwide and the gaining popularity of social media and various websites have also exposed us to the verge of getting hacked especially if we are using an old version browser since the cute/innocent-looking images we see into the browser poses threat to us, as it has potential to steal our cookies, perform Distributed Denial of service (DDoSer), performing phishing attacks, and scanning victim completely and so on. To run any website, nowadays basic thing is to be able to run JavaScript by the browser. The content security measure is enabled in all of the browsers so they do not run inline JavaScripts but allows to execute the JavaScript files loaded from the same domain. Almost every site contains a WYSIWYG editor that allows writing posts in HTML form and uploading images, exploiting this working mechanism we will perform various attacks.**

***Index Terms*—stegnography, attacks, content security policy, WYISWYG editor, JavaScript, OWASP Xenotix XSS Framework**

I. Introduction

Unlike an encryption technique, we do not encrypt malicious code or an image in case of stenography attack rather we include exploit within the existing image in a way so that it is not detected passing through the network and going to the victim machine and browser also fails to assess anything harmful with the content to be rendered. For this attack, we will expand the header of the images like GIF and BMP and put in there the serving gateway (URL) so that we can serve our malicious code from the server running in a distant place or from the server ran by attackers and that code will be executed in the browser without a problem. Another concept is like we create IMAJS (image+JS), which has bipolar functionality, when loaded into the image tag for HTML it renders an image and when loaded into the script tag of the HTML it executes the JavaScript which is also we call a polyglot.

II. History

Steganography conceals the fact the message has been conveyed via a carrier. It was found first used in 1499, but the idea has found existed since ancient times from the Roman empire whereby a slave was used to communicate by tattooing information onto the scalp and when hair grows back that slave was used to send to the mission and then the receiver of the slave now can shave his hairs and read the message. Another form of this existed like the art of secret writing and encoding messages into an image and send to the receiver where that image never raises suspicion at all and now receiver can open and read message readily with the help of particular software. These techniques nowadays attackers have started using to hack someone's account or compromise a system, developing the malware and various tools to achieve their goals [1].

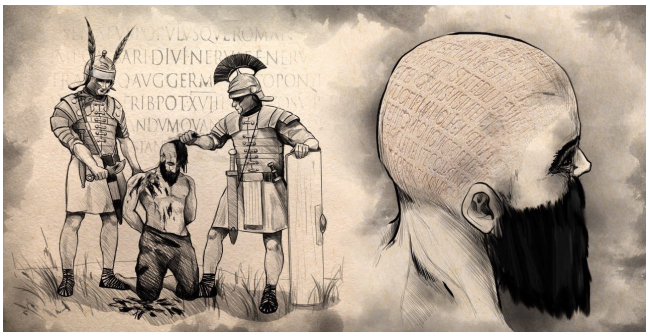

Fig. 1. Tattooing in ancient times

III. Related works

In the past, we have seen researchers coming up with various ideas like researchers Billy Rios and Nate Ncfeter, came up with the idea to embed Java ARchieves into GIF file and upload it to the server and now that GIF plus Java Archives (GIFAR) [5] now can establish a connection to somewhere else which is dangerous since it violated the same-origin policy principle. Another thing, some attackers do is to override or insert additional information into the Exif data of the images and put in the PHP shell ASP shell and own the server by executing commands and reading the server's private information such as passwords. The researcher Saumil Shah has come up with his idea to deliver the exploit that he calls stegosploit and he encodes HTML, CSS, and JavaScript into the single image, creating polyglot and deliver it to the victim so that it runs on the victim machine and that way victim machine now has remote access from the attacker. But here we will expand the header and put out malicious code into the header and upload to the server and run the attacks [2].

IV. Graphics Interchange Format (GIF)

We use the GIF since it has existed among us due to its robustness to produce an image independent to the various platforms. It was developed by an American computer scientist

Steve Wilhite on 15 June 1987. It is lossless bitmap image format and supports up to 8 pixels for each image, allowing a single image reference its palette of up to 256 different colors chosen from 24-bit RGB color space [3]. A typical GIF file format will look like this with the URL to serve the exploits. The use of putting data into the header expanding is inspired by the Saumil shah dealy pixel presentation.

```
WIDTH equ 10799 ; equivalent to 2f2a, which is '/*' in ASCII
HEIGTH equ 100 ; just to make it easier to spot
db 'GIF89a'
    dw WIDTH, HEIGTH
db 0 ; GCT
    db -1 ;  background color
    db 0 ; default aspect ratio
; removing some content for succintness
db 02ch ; Image descriptor
    dw 0, 0 ; NW corner
    dw WIDTH, HEIGTH ; w/h of image
    db 0    ; color table
db 2 ; lzw size
db 0
db 3bh ; GIF terminator
db '*/'  ; closing the comment
db '=1;' ; creating a fake use of that GIF89a string

db 's = document.createElement("script");'
db 's.src = "http://127.0.0.1:5058/xook.js";'
db 'document.body.appendChild(s);'
```

Fig. 2. GIF format with OWASP XSS URL [4]

And similarly, we can do to the other formats of the images, but here I have only worked with GIF and BMP file format and have demoed during the class.

V. Working Mechanisms

I am going to illustrate that what is the recipe for the attack to work and what is the underlying concept for this. We have taken here taken the example of JPEG header just for clarifying the concept that is we will expand the size of the header so that when loaded into the script it will behave as JavaScript and when loaded into the image tag it loads the image.

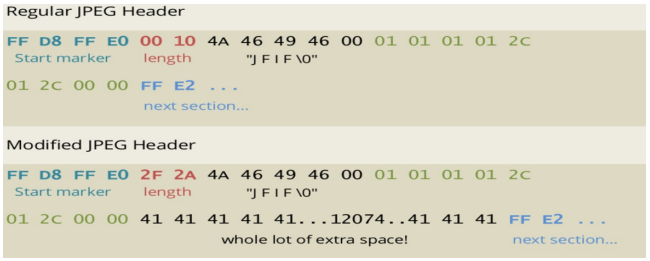

Fig. 3. JPEG header format [6]

Here the length of the header is 00 10 and it is the original length of the JPEG header and now we will increase the size header to 2F2A which is ASCII value of multi-line comment /* in the JavaScript as well and now it will serve two purposes here when loaded in the image tag 2F2A is the length of the header and we in script tag it is a comment. We just created bipolar content.

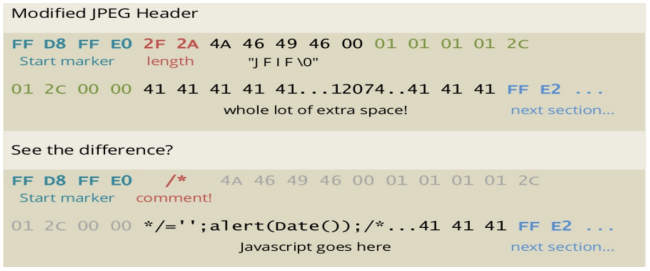

Fig. 4. Difference when loaded as image and JavaScript [6]

Here the same concept has been shown how a single image shows different behaviors when viewed in different containers. Like 2F2A has treated a multi-line comment and FFD8FFE0 = ” acts like a variable of JavaScript and other pixel information of the image has been commented and now alert(Date()); will alert the current date.

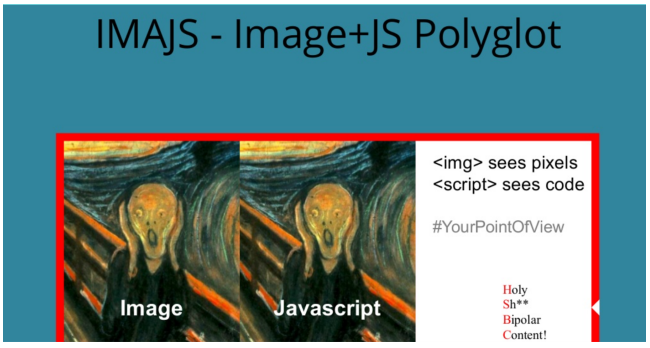

Fig. 5. IMAJS Polyglot [6]

The bipolar content concept in a more visually appealing form here is shown. When the same image is loaded in img tag it sees the pixels all over the place in the image but the same image we load into the script tag it sees the JavaScript code which will be executed upon loading.

VI. Cross-site Scripting (XSS)

The core security feature is the same origin policy (SOP) where the read and write to a site from another site is prevented.

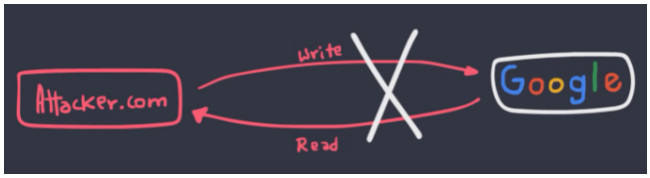

Fig. 6. XSS, Same origin principle violated

But the attacker.com will allow if the same-origin policy is not violated by google.com that states that the site accessing the information should have the same protocol, host, and port if these matches then the attacker.com will allow reading and writing to the site by google.com here in the image.

Fig. 7. Same origin principle obeyed

Which as shown in the image implies that the site itself only can read and write its content and no other site than this, but if they allow read and write from another site the site (google.com) itself has to give the access from content security policy explicitly who can read and write the content. The XSS is JS code injection and what we do with it is up to us. Here in this project, we have a scenario that does not violate the same-origin policy or content security policy by serving the bunch of JS files from the same domain.

VII. Tools and Technology

The ingredients we need to execute this project on the server.

OWASP-Xenotix-XSS-Exploit-Framework is an advanced Cross Site Scripting (XSS) vulnerability detection and exploitation framework which also contains most popular browser engines such as Trident (Internet Explorer), Webkit (Chrome), and Gecko (Firefox) where we can perform various experiments for a site within this framework itself since it contains world's 2nd largest exploits counting to 1500+.

Another a Github repository [9] used for injecting code into the header of images of file format such as gif and BMP file format. We can call it that this tool was used to create bipolar content or we also call it polyglot.

For this project, I have used the MySQL database for the post posted to be persistent such that we can view at any time and PHP for writing that website [10], also the WYSIWYG editor is one of the famous editor called CKEditor, which is a rich text editor and enables to write inside the webpages.

VIII. Project Design

We will discuss how we can set up this attack in your system and perform it at ease. We can perform it with any website containing What You See Is What You Get (WYSIWYG) editor that is available in most of the content management system (CMS) nowadays and embed Xenotix cross-site scripting framework URL to deliver the exploits from the framework, which is loaded with 1500+ exploits ready to be delivered.

1. Embed exploit URL into the GIF/BMP image, an attacker can use his/her server IP or domain to deliver the exploits. In the GIF we can inject as below.

```
db 's = document.createElement("script");'
db 's.src = "http://127.0.0.1:5058/xook.js";'
db 'document.body.appendChild(s);'
```

Fig. 8. Code to be injected into GIF [4]

2. Create a post in WYSIWYG editor, uploading the image we just created injecting the exploit URL.
3. Edit post as source and load this image into the script tag as well and now your image will act as pixels in img tag and JS script in script tag. As shown in the image here.

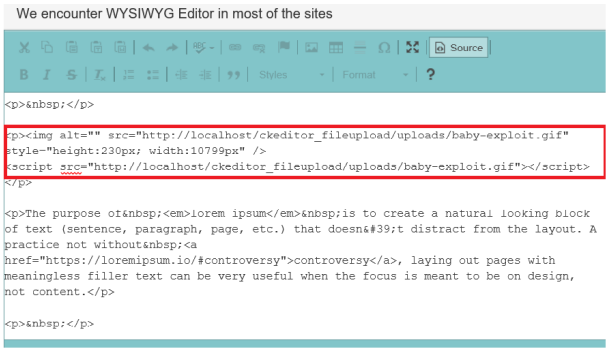

Fig. 9. Adding script tag into the editor [10]

4. Start Xenotix cross-site scripting framework server so that we can deliver our exploits into the victim's browser. Here is the snap of the attacker server started in localhost and this same URL has been injected into our image and now this URL is used to deliver the exploits.

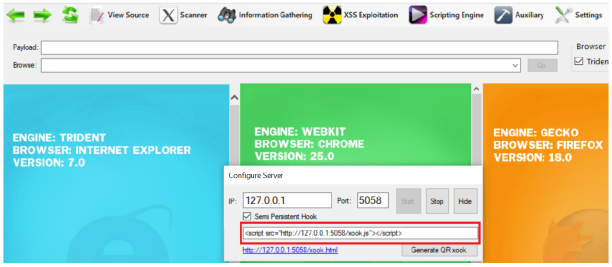

Fig. 10. The Xenotix server ready to deliver exploits [7]

IX. Results

In this project, we will see two attacks from the attackers to the visitors of the post we created with an image acting as bipolar when loaded in different containers. The attacks are Distributed Denial of Service (DDoS) and phishing to harm the user browsing the website [10].

A. Distributed Denial of Service (DDoS)

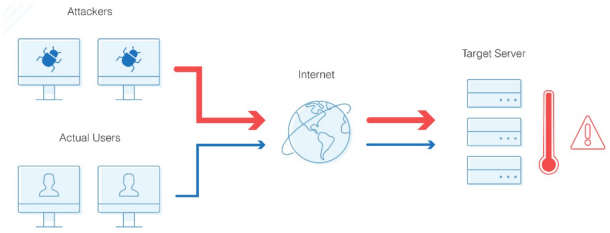

Fig. 11. The big picture of DDoS attack

In the DDoS attack, the attacker can overwhelm the server using a tool, like a bot here. We are using the Xenotix server to do the same. Web services particularly are at risk as the hackers can target services by overwhelming the network with traffic and eventually can stop the server to handle the real user requests.
Now we will show how this I have done using xenotix framework.

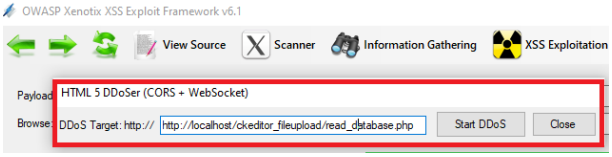

Fig. 12. URL called thousand times in a sec

The URL in this image was called many times until the server run out of resources such as memory, processing and hence the intention of the attacker is achieved with this.

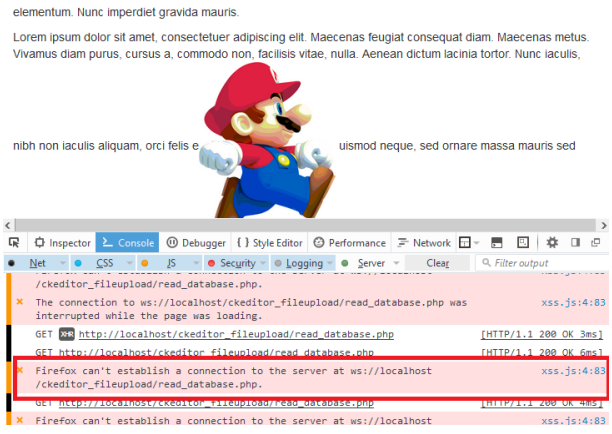

Fig. 13. The server overloaded with the requests from the attacker URL

In the image above, we can see that our localhost server has been bombarded with so many requests.

B. Phising Attack

In the phishing attack, we fool users by sending a legitimate-looking interface so that we could steal the users' private credentials, for example, online banking username and password, Facebook account details, and so many others.

Fig. 14. The big picture of phishing

In this section, we will see that phishing attacks using facebook.com and getting facebook data of the victim to the attacker server in real-time. In this fig 15, the victim has been redirected to the attacker's phishing page so that the attacker can steal the user's private information.

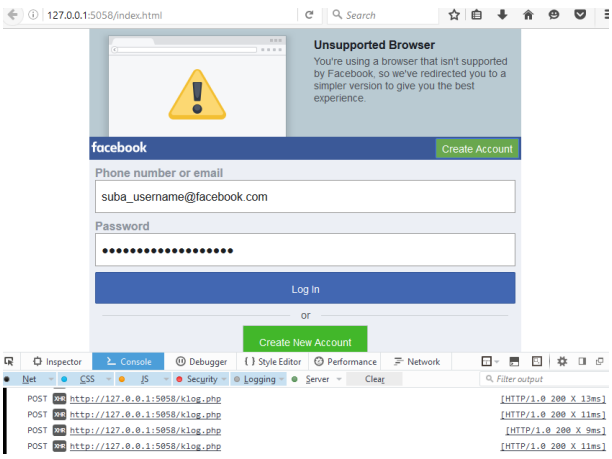

Fig. 15. The victim details recorded to attacker server

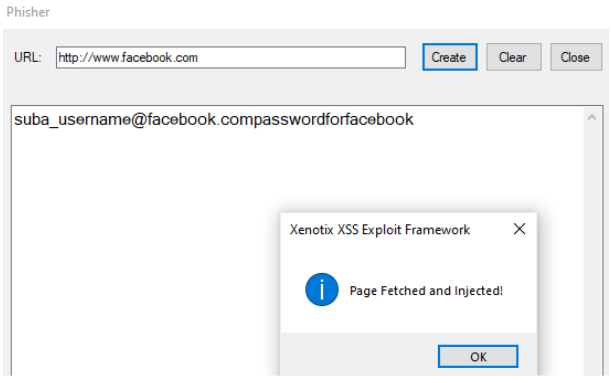

Fig. 16. The attacker recorded to its server

After entering a fake site and injecting into the users reading a post in a website now those users will be redirected to the fake site. Here fake site is facebook.com that will look similar to a real Facebook site but when the user enters its username and password, with each keystroke the entered data will be sent as a POST request to the server, and those users accounts are compromised severely.

Here in figure 15, we see requests made in each keystroke and posting entries to the attacker server that server here is Xenotix framework and whose screenshot is in figure 16.

X. Conclusion and Future works

In this project, we have seen how an image could lead to running a server out of resources (DDoS attack) to stealing someone's private information (phishing) like bank details, Facebook username and password, and so on. The latest version browsers do the right job of not executing an image as JavaScript and vice versa, but still, if someone is using Internet Explorer or older Firefox and Chrome. They are the potential candidate of getting hacked with such sort of attacks, and hence a recommended way is to always upgrade your browser to the latest secure version and not use the Internet Explorer at all.

For future works, XSS attacks can be delivered to the victim machine in a smarter so that even the latest browsers can not raise suspicion and still run the JavaScrip injected into the image. Maybe playing with some modifications in the header information from the server to the browser and exploiting the way the browsers think about the images, we could do this in the future.

References

[1] Alexey Shulmin, "Steganography in contemporary cyberattacks ", [Online]. Available: https://securelist.com/steganography-in-contemporary-cyberattacks/79276/. [Accessed 3 December 2020].

[2] Saumil Shah, "Stegosploit – Drive by Browser Exploits using only Images Presented By Saumil Shah", [Online]. Available: https://youtu.be/zyLxYfGlGZE. [Accessed 30 November 2020].

[3] Wikipedia, "Graphics Interchange Format (GIF)", [Online]. Available: https://en.wikipedia.org/wiki/GIF. [Accessed 1 December 2020]

[4] Jin, "When GIF serve JavaScript!", [Online]. Available: http://iamajin.blogspot.com/2014/11/when-gifs-serve-javascript.html. [Accessed 2 December 2020]

[5] Wikipedia, "Gifar", [Online]. Available: https://en.wikipedia.org/wiki/Gifar. [Accessed 2 December 2020]

[6] Saumil Shah, "Stegosploit - Hacking With Pictures", [Online]. Available: https://www.slideshare.net/saumilshah/stegosploit-hacking-with-pictures. [Accessed 2 December 2020]

[7] Ajin Abraham, "OWASP Xenotix XSS Exploit Framework", [Online]. Available:https://github.com/ajinabraham/OWASP-Xenotix-XSS-Exploit-Framework. [Accessed 2 December 2020]

[8] Saumil Shah, "Exploit Delivery via Steganography and Polyglots ", [Online]. Available:https://stegosploit.info/. [Accessed 4 December 2020]

[9] Github user, "imagejs - Small tool to package javascript into bitmap file ", [Online]. https://github.com/jklmnn/imagejs. [Accessed 4 December 2020]

[10] Suba Sah, " Stegnography attack demo site ", [Online]. https://bit.ly/2VErXk9. [Accessed 4 December 2020]