



Security Report on WebGL

Context Information Security

blogs@contextis.com

June 2011



Contents

| | |
|--|-----------|
| Blog 1 – WebGL - A New Dimension for Browser Exploitation | 4 |
| Summary | 4 |
| WebGL | 4 |
| Quick Overview of 3D Graphics Pipeline | 5 |
| The Trouble with WebGL | 7 |
| Denial of Service | 8 |
| Cross-Domain Image Theft | 9 |
| Conclusions | 10 |
| Blog 2 - WebGL – FAQ | 11 |
| “Am I vulnerable to WebGL based attacks?” | 11 |
| “How can we be safe from WebGL based attacks?” | 11 |
| “You said we should consider disabling WebGL, how exactly would you go about doing that?” | 11 |
| Firefox 4 | 11 |
| Chrome | 12 |
| Safari | 12 |
| “No Script Provides WebGL Security for Firefox” | 12 |
| “Would the use of the GL_ARB_robustness extension eliminate the Denial of Service issues with WebGL?” | 13 |
| “The proof of concept does not seem to work correctly; is WebGL not supported on Nvidia/ATI/Intel graphics cards?” | 13 |
| “Is there a way of blocking the cross-domain image issue without removing support for cross-domain textures or imposing CORS?” | 13 |
| “What responses have you had from the Browser vendors?” | 14 |
| Blog 3 - WebGL – More WebGL Security Flaws | 15 |
| Summary | 15 |
| Background | 15 |
| Cross-platform testing against the Khronos solution | 16 |
| The Case of the ATI Failure | 20 |
| Exploiting the Vulnerability | 21 |
| Conclusions | 23 |
| Contributions | 24 |
| About Context | 25 |



Introduction

This report is a consolidation of the blogs written by Context on the subject of WebGL security in browsers. Context found serious fundamental security issues with the specification, design and implementation of WebGL. The findings of our research are contained within this document.



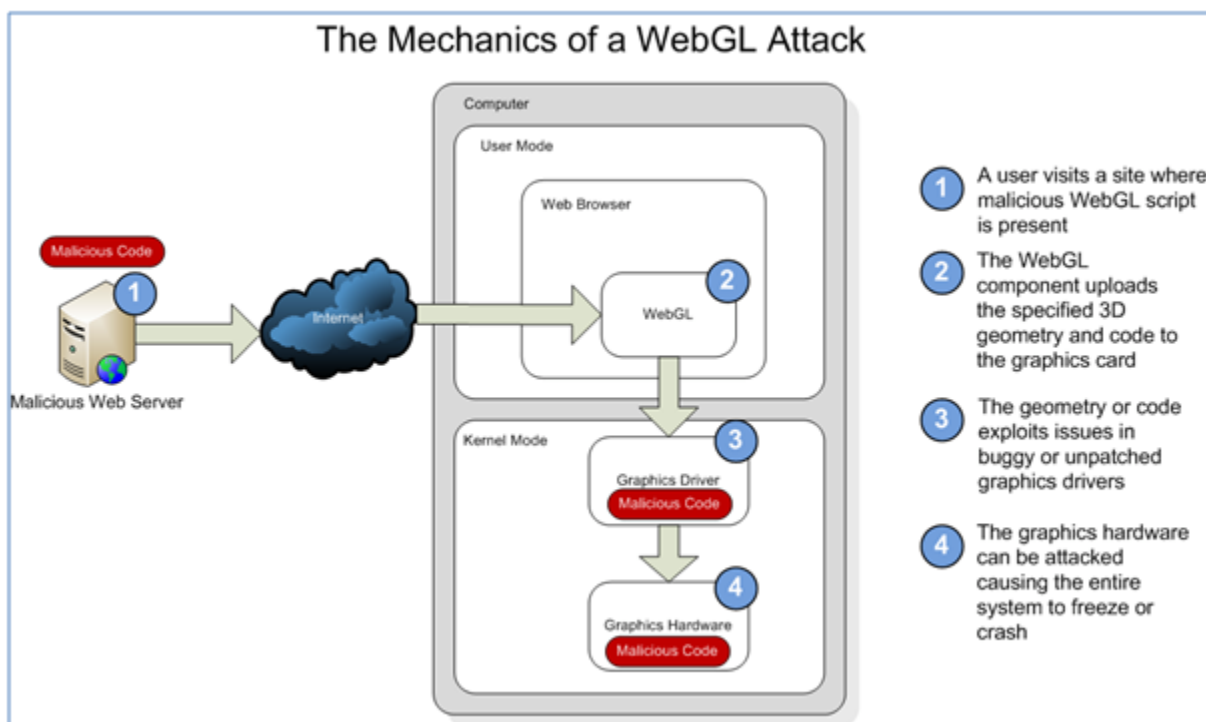
Blog 1 – WebGL - A New Dimension for Browser Exploitation

9th May 2011

Summary

WebGL is a new web standard for browsers which aims to bring 3D graphics to any page on the internet. It has recently been enabled by default in Firefox 4 and Google Chrome, and can be turned on in the latest builds of Safari. Context has an ongoing interest in researching new areas affecting the security landscape, especially when it could have a significant impact on our clients. We found that:

1. A number of serious security issues have been identified with the specification and implementations of WebGL.
2. These issues can allow an attacker to provide malicious code via a web browser which allows attacks on the GPU and graphics drivers. These attacks on the GPU via WebGL can render the entire machine unusable.
3. Additionally, there are other dangers with WebGL that put users' data, privacy and security at risk.
4. These issues are inherent to the WebGL specification and would require significant architectural changes in order to remediate in the platform design. Fundamentally, WebGL now allows full (Turing Complete) programs from the internet to reach the graphics driver and graphics hardware which operate in what is supposed to be the most protected part of the computer (Kernel Mode).
5. Browsers that enable WebGL by default put their users at risk to these issues.



WebGL

Throughout the history of the Web there has been a drive to allow greater interactivity and expressiveness in web content. Starting with the initial forays into scripting,



extensive plugin capability and ActiveX through support for HTML5 functionality such as the video or canvas tags, more and more complexity has been provided in the browser by default.

At each stage in the evolution of the modern browser existing security tenets have had to be re-evaluated to ensure new functionality does not open up any serious attack vectors. As an example, before scripting was introduced there was no easy mechanism for a malicious page to gain access to another site's content; therefore there would be no need for implementing a same-origin policy. Security decisions made during the early days of the browser may no longer be appropriate to modern advancements, especially ones regarding this cross-domain access of content.

While the theft of data is a serious issue the integrity of the browser and the host operating environment should also not be forgotten when introducing new technology. Sometimes the benefits may prove to be more of a curse. As an example take binary browser-plugin support (e.g. ActiveX or the Netscape Plugin Application Programming Interface). The support makes it very easy for third parties to extend the functionality of the browser and provide callable interfaces for web pages. This in turn opens the attack surface of the browser to potentially a larger corpus of code, some of which is almost certainly badly written. It might then become difficult for a browser vendor to secure the platform, as it might not even be their code which is an issue, leading to such band-aids as IE's Killbits and Firefox's plugin checker to block or inform the user an update might be necessary. In the end the only secure way to use such wide ranging native content is probably not to have it in the first place, but the general consensus at the moment is the benefits outweigh the security risk.

This leads to the topic of this blog post, WebGL. If it is something you have yet to hear of you almost certainly will soon. WebGL's goal is to introduce an OpenGL derived 3D API to the browser, accessible through JavaScript from any Web page which wants to use it. The recently released Mozilla Firefox 4 browser has enabled support by default, as has Google's Chrome browser since version 9 and Safari (WebKit) 5.

This in itself should not really be controversial, however the way in which it is implemented, coupled with the way current PC and Graphics Processor architectures are designed has led some to question the security of the approach. Context has performed some initial investigations into possible security concerns which seem to be inherent in the specification, leading to questions as to whether it should be currently available on supporting platforms, and if its benefits actually outweigh its risks.

Quick Overview of 3D Graphics Pipeline

First let's start with a very simplified overview of how 3D graphics is implemented in most modern PC style architectures.

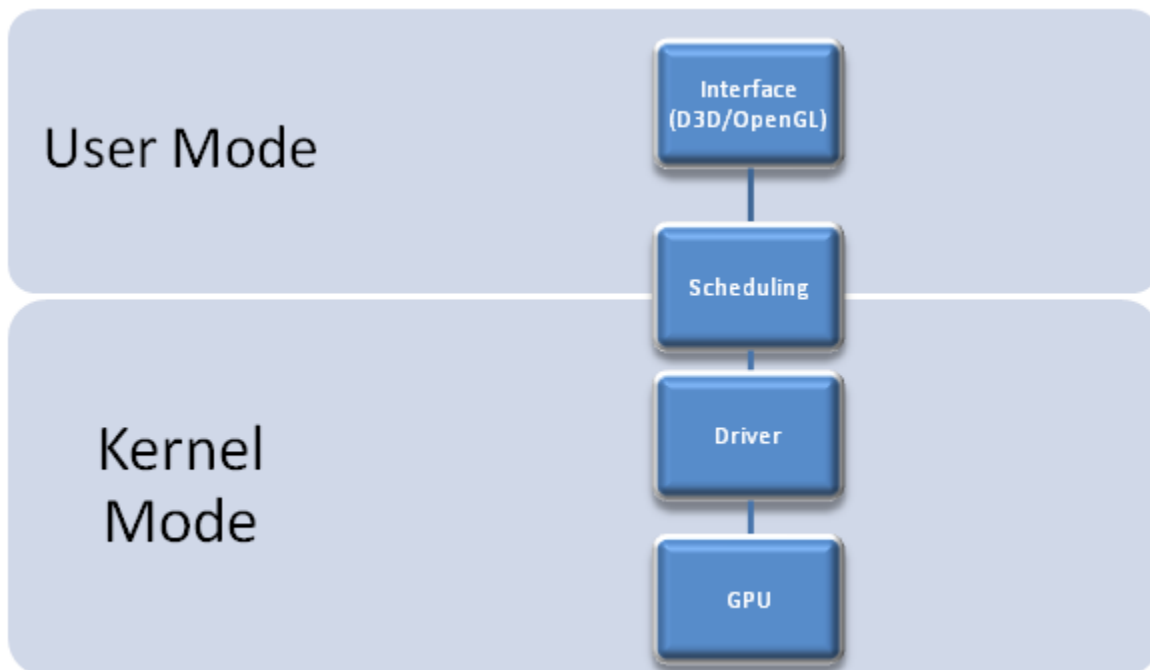


Figure 1 - Simple Diagram of Graphics Pipeline

At the lowest level is the Graphics Processor (GPU) hardware itself, this does not necessarily implement any specific API (almost certainly it is a proprietary interface developed by the manufacturer), however it should at least support all the functionality expected at the programming API level. Almost all modern 3D hardware contains individual programmable units (usually referred to as shaders), these can be individually programmed by the user-mode processes. The native format of the shader code is generally specific to the hardware vendor; however common languages exist to permit cross platform code to be developed.

Above the hardware is a driver which tends to run within kernel mode on the operating system; its job is to handle the low-level hardware aspects and to provide a standardised interface (e.g. WDDM) through which other components of the operating system can access the GPU.

Next is the scheduling, this could be implemented in a number of different locations, for example in the kernel driver itself, by the OS or entirely in user mode. Its responsibility is to share access to the GPU between individual programs running on the same machine. In a more traditional environment this would not be necessary because only one application (for example a windowing manager) would actually need direct access to the GPU at any one time. In a 3D scenario the requirement to directly access the shaders and memory to upload texture and geometry means this must be managed appropriately.

The final piece in the stack is the interface library, which is the main route through which user processes access the graphics library. This is the final level of abstraction, removing where possible any hardware specific functionality. Common interface libraries are Direct3D (which also has some kernel functionality) and the cross-platform OpenGL. They provide APIs to create the 3D geometry to be displayed, compilers to convert



shader programs into a more suitable representation for the GPU and manage the allocation and uploading of texture information to video card memory.

The Trouble with WebGL

Based on the simplified description it is possible to go into what is currently the issue with the way WebGL is specified, designed and implemented. Traditional browser content would not normally have direct access to the hardware in any form, if you drew a bitmap it would be handled by some code in the browser with responsibility for drawing bitmaps. This would then be likely to delegate that responsibility to an OS component, which would perform the drawing itself. While this distinction is blurring somewhat with the introduction of 2D graphics acceleration in all the popular browsers it is still the case that the actual functionality of the GPU is not directly exposed to a web page. The salient facts are that the content is pretty easy to verify, has a measurable render time relative to the content, and generally contains little programmable functionality (at least which would be exposed to the graphics hardware).

WebGL on the other hand provides, by virtue of its functional requirements, access to the graphics hardware. Shader code, while not written in the native language of the GPU, are compiled, uploaded then executed on the graphics hardware. Render times for medium to complex geometry can be difficult to determine ahead of time from the raw data as it is hard to generate an accurate value without first rendering it; a classic chicken and egg issue. Also some data can be hard to verify and security restrictions can be difficult to enforce once out of the control of the WebGL implementation.

This might not be such an issue, except for the fact that the current hardware and graphics pipeline implementations are not designed to be pre-emptable or maintain security boundaries. Once a display list has been placed on the GPU by the scheduler it can be difficult to stop it, at least without causing obvious, system-wide visual corruption and instabilities. By carefully crafting content it is possible to seriously impact the OS's ability to draw the user interface, or worse. The difficulty in verifying all content and maintain security boundaries also have potential impact on the integrity of the system and user data.

Up to now the manufacturers of graphics hardware haven't really needed to worry about an un-trusted use case for their products. Certainly the issues of integrity and denial of service would be considerations even for native programs, but the developers will generally have a vested interest in making sure that their programs do not cause problems. A malicious actor would need to convince someone to install their bad code, at which point attacking the graphics hardware might be the least of the user's worries. Graphics drivers are generally not written with security as their main focus, performance is likely to be most critical. Security costs a significant amount in both man-hours and monetary terms, there seems to be little incentive for the manufacturers to harden their products (potentially at the expense of performance) to support the WebGL in its current form.

Even if security issues are identified it is unclear what the patch strategy employed by the large GPU manufacturers would be. Searching Security Focus for either ATI or NVIDIA only produces a few publically disclosed vulnerabilities (dating back to 2008); a Google search for related security bulletins also does not bring up any information. Considering the complexity of the drivers and hardware interactions it seems hard to



believe that there has never been an exploitable bug in their software which needed immediate remediation.

Of course the patching situation might not be helped by the typical restrictions on OEM products, especially laptops. Typically in these situations the reference driver provided by the GPU manufacturer is blocked from installing on a laptop, making any security update considerably more difficult to deploy.

During the development of WebGL it seems that all the browser vendors supporting it have encountered issues with certain drivers being unstable or crashing completely. The current work around for this seems to be a driver black list (or in Chrome's case not running WebGL on Windows XP at all). (See https://wiki.mozilla.org/Blocklisting/Blocked_Graphics_Drivers). This does not seem to be a very tenable approach long term.

Denial of Service

The risk of denial of service is one of the most well known security issues facing WebGL, not least because it is even documented in the current standards documentation (see <https://www.khronos.org/registry/webgl/specs/1.0/#4.4>). Basically because of the almost direct access the WebGL API has to the graphics hardware it is possible to create shader programs or a set of complex 3D geometry which can cause the hardware to spend a significant proportion of its time rendering. It is easy to trivialise client denial of service attacks when the only affected component is the browser process (there are numerous ways of doing this already), however in this case the attack can completely prevent a user being able to access their computer, making it considerably more serious.

In certain circumstances Context has observed the operating system crashing (i.e. Blue Screen of Death). These crashes can be benign (from an exploitability sense) to ones where the driver code has faulted causing potentially exploitable conditions. No further details of actual exploitable vulnerabilities or the code used to generate them is to be disclosed at this time.

Windows 7 and Vista seem to fair slightly better in this regard, if the GPU locks up for around 2 seconds the OS will force it to be reset. This stops all applications from using any 3D graphics during this period of reset. However these OSes also have a maximum limit to how many times this can happen in a short time window before the kernel will force a bug check (Blue Screen of Death) anyway (see <http://msdn.microsoft.com/en-us/windows/hardware/gg487368.aspx>).

Of course as it is a known issue there are efforts to mitigate it, for example the ANGLE project (<http://code.google.com/p/angleproject/>) includes a shader validator to eliminate simple infinite loop cases, which is used in Firefox 4 and Chrome. This validation cannot possibly block all cases leading to denial of service, especially when you can create large geometry and shaders which don't contain loops but still take substantial amounts of time to execute.

At this point it would seem to be reasonable to provide a proof of concept; however Context did not need to even write one as Khronos provides one in their WebGL SDK. See <https://cvs.khronos.org/svn/repos/registry/trunk/public/webgl/sdk/tests/extra/lots->



[of-polys-example.html](#). This page has been found to completely lock the desktop on OSX, reliably crash XP machines and cause GPU resets on Windows 7.

Cross-Domain Image Theft

One of the fundamental security boundaries in the specification of the Document Object Model and browser handling of JavaScript is the domain boundary. This is to prevent content served from say, [www.evil.com](#) being able to access authenticated/trusted resources on [www.mybanking.com](#). Whether content is permitted to be accessed across this boundary very much depends on the type of resource being accessed. This is sometimes referred to as "Right to Embed" vs. "Right to Read". For example it is perfectly acceptable to embed an image from outside of your domain because the underlying APIs never gave you a mechanism to read the actual content (outside of image dimensions, and an indication of success or failure to load). On the other hand trying to use the XMLHttpRequest object to pull content from outside your domain (and therefore giving you access to the raw data) is generally not permitted.

Before the introduction of the 'Canvas' element, which is being standardised in HTML5 there were not many options for stealing the raw data of images cross domain. To combat this, an 'origin-clean' flag was implemented. This flag is initially set to true and is set to false if any cross domain image or content is used on the canvas (see <http://www.w3.org/TR/html5/the-canvas-element.html#security-with-canvas-elements>). Once the 'origin-clean' flag is false you can no longer call the APIs such as 'toDataURL' to extract the image content.

The WebGL API is built on top of the 'Canvas' element and so extends the concept of the flag to also encompass the use of cross-domain textures (see <https://www.khronos.org/registry/webgl/specs/1.0/#4.2>). This would be the end of it except for one slight issue. As already discussed with regards to denial of service it is possible to cause shading code and geometry drawing to take a non-trivial amount of time. One of the resources which a shader can directly access is the pixel data of textures, which once it reaches the shading code it no longer has any concept of origin. Therefore it is possible to develop a timing attack to extract pixel values even if we cannot read them directly. This can be done by changing how long a shader runs depending on the colour or brightness of a pixel and measuring the time the drawing process takes in JavaScript. This is a standard attack technique in the security field although it is most often used for breaking cryptographic systems. In relation to WebGL it has already been mentioned in a public mailing list that this could be an issue (see <http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2011-March/030882.html>).

Of course an attacker might not even need to extract the entire pixel data of the image for this to be of use. For example it could be used to compare a cross-domain image to another known image, returning a simple true or false value. As an example imagine a web site which returns a profile picture on a fixed URL, the content determined by the session cookie stored in the browser. An attacker might be able to compare this cross-domain image against a known list of profile pictures to identify when specific a specific person is using the malicious site.

Therefore as part of our investigations into WebGL a proof-of-concept has been developed to demonstrate the attack is practical (if a little slow). To access the PoC go



[here](#). It has been tested in Firefox 4 and Chrome 11, on Windows XP, Windows 7 and Mac OSX. It works best in Firefox. It should be noted that Context do not hold any of the data captured on the page, everything is done on the client. For those without a WebGL capable machine or browser there is also a short video [here](#).

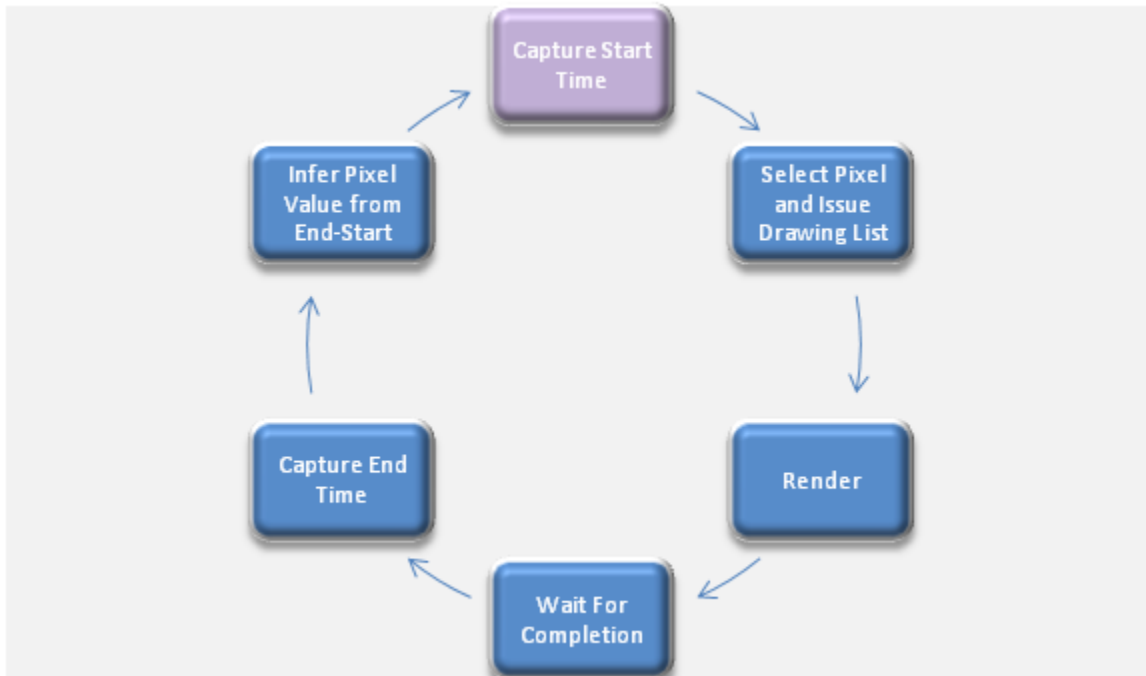


Figure 2 - Flow Diagram Showing Stages of Image Capture

This is something which we believe can only be fixed by changing the nature of cross-domain image access in the specification of WebGL. This could be achievable via blocking all cross-domain images or using something like CORS (<http://www.w3.org/TR/cors/>) to only permit specific image content to be accessed from certain domains.

Conclusions

Based on this limited research Context does not believe WebGL is really ready for mass usage, therefore Context recommends that users and corporate IT managers consider disabling WebGL in their web browsers.

While there is certainly a demand for high-performance 3D content to be made available over the web, the way in which WebGL has been specified insufficiently takes into account the infrastructure required to support it securely. This is evident from the development of ways to mitigate the underlying security issues by introducing validation layers and driver black-lists; however this still pushes much of the responsibility of securing WebGL on the hardware manufacturers. Perhaps the best approach would be to design a specification for 3D graphics from the ground up with these issues in mind.



Blog 2 - WebGL – FAQ

11th May 2011

Due to the high level of interest in Context's blog posting on the Security issues within WebGL we are releasing the following further information to aid in the understanding of the issues.

“Am I vulnerable to WebGL based attacks?”

This depends on the browser you are using, operating system and graphics card. Use this [test page](#) to determine if you have WebGL support and are therefore exposed. WebGL is currently supported on Firefox and Chrome web browsers; if you are using Internet Explorer, Safari or Opera (at the time of writing) you are not vulnerable to WebGL issues.

“How can we be safe from WebGL based attacks?”

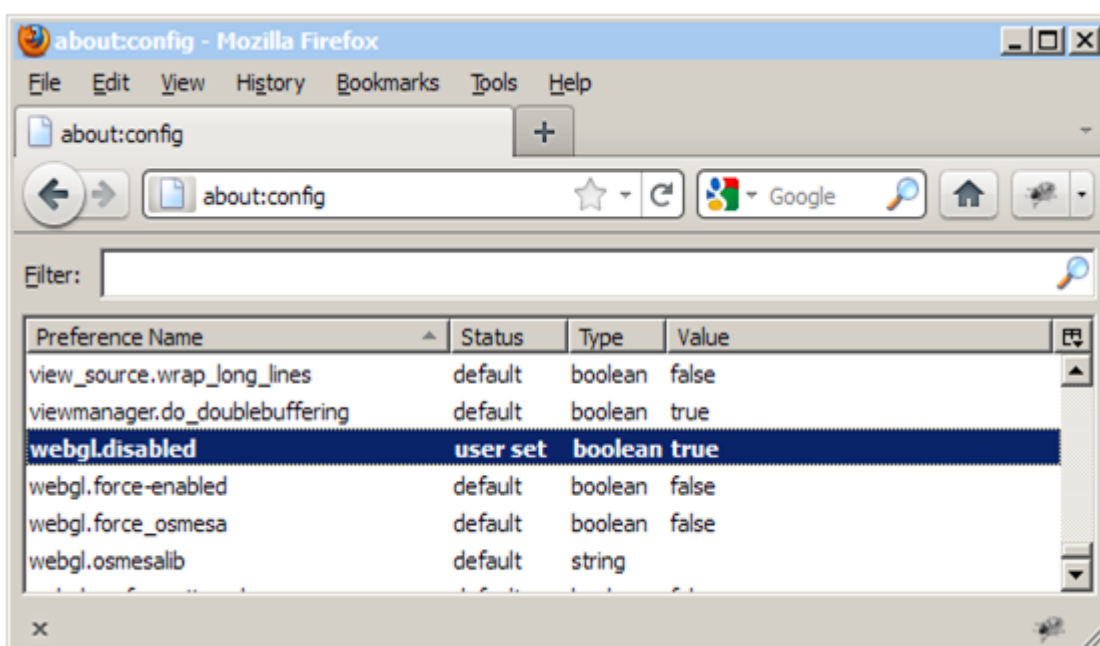
Context recommend that users disable WebGL support from within their web browsers in the short term (see below).

However in the longer term, Context believes that browser vendors should, by default, disable WebGL from within their web browsers. We would like to see functionality included that would allow users to opt-in for WebGL applications that they trust on a case by case basis.

“You said we should consider disabling WebGL, how exactly would you go about doing that?”

Firefox 4

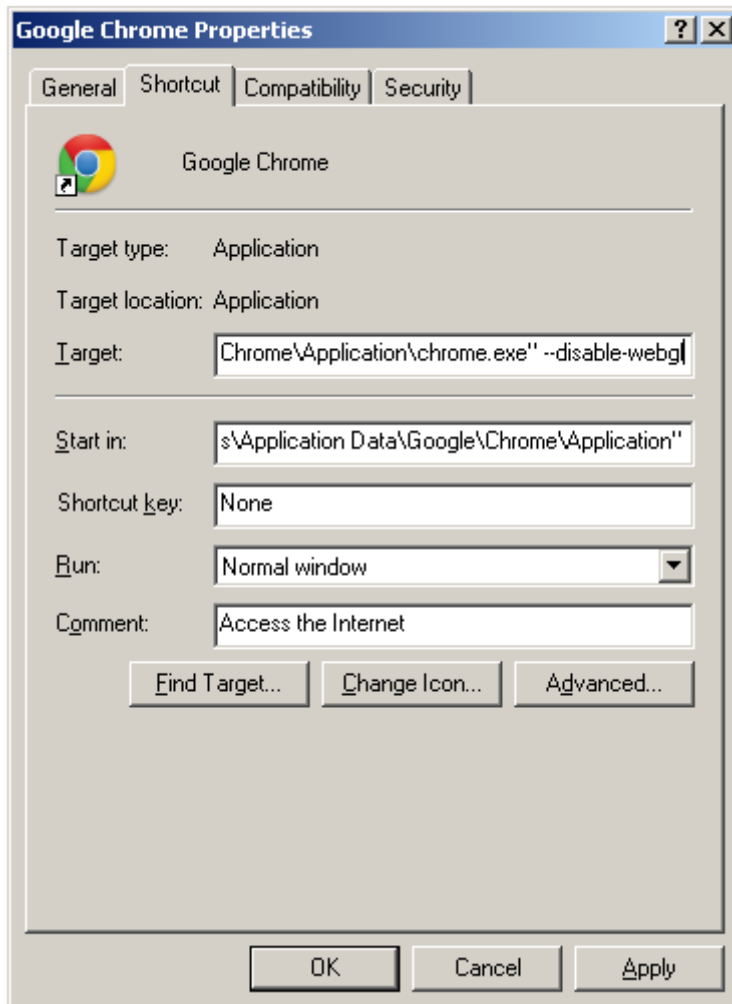
- Type into the URL bar “about:config” and click the “I’ll be careful” button.
- Find the setting “webgl.disabled” and set it to true as show in the following picture:





Chrome

- For Chrome on Windows pass the flag "--disable-webgl" when running the executable by changing the shortcut in the start menu. A user can right click on the chrome shortcut, select properties and add the flag as per the following screenshot.



- For Chrome on OSX the parameter "--disable-webgl" needs to be added on startup. See the following article for details on adding parameters for chrome.
<http://superuser.com/questions/271678/how-do-i-pass-command-line-arguments-to-dock-items>

Safari

- Unless you are using a nightly builds of WebKit WebGL is not easily available and requires a user preference setting to enable. Therefore you do not need to actively disable it at the moment.

"No Script Provides WebGL Security for Firefox"

Context has been working with the developers of the Firefox plug-in NoScript (<http://noscript.net/>) to include support to allow users to selectively disable WebGL. Context recommends this plug-in to protect users from malicious content from the Internet.



“Would the use of the GL_ARB_robustness extension eliminate the Denial of Service issues with WebGL?”

Khronos (the authors of the WebGL specification) released a blog posting on 9th May 2010 detailing certain security mechanisms that enhance the security of WebGL <http://www.khronos.org/news/permalink/webgl-security>. One of the theoretical solutions suggested is the use of GL_ARB_robustness which is an extension that graphic card manufacturers can provide to detect the Denial of Service issue that we detailed in our original Blog. Based on the current information available it seems that it would at least mitigate the direct DoS condition where the whole machine becomes unresponsive or crashes. It puts the onus on the Graphics Hardware manufacturer to develop a mechanism to detect a lock up condition and reset the device, the extension then really provides a way for clients to recognise compatible configurations.

However we do not believe it addresses the wider issue. The resetting of the graphics card and driver should be seen as a crutch to OS stability when exceptional conditions occur and not as a mechanism to protect users from malicious code. Resetting the graphics card isn't guaranteed to be a trouble free operation; all other users of the graphics subsystem will need to correctly handle the event. The graphics stack would have to ensure that any hardware resources are recreated before use to guard against another application misusing it. This operation, while not causing a DoS directly, could still indirectly effect the entire system and the applications running on it.

Perhaps the closest analogy would be to consider a hard drive which could be locked up when someone issued a malicious Write command. Rather than stopping malicious code from issuing these “bad” writes, instead the manufacturer just allows the hard disk to be reset, making everyone's state inconsistent from the file system layer to user mode processes. That does not seem a great way of fixing it.

“The proof of concept does not seem to work correctly; is WebGL not supported on Nvidia/ATI/Intel graphics cards?”

In theory any graphics card capable of supporting OpenGL (or DirectX on Windows) should work with WebGL. However the browsers have a blacklist of graphic drivers where significant issues have been encountered and will block WebGL use. Therefore normally the way to get WebGL support is to update to the latest drivers for your graphics card however not all hardware has compatible drivers. Where a driver is blacklisted by the browser it is still possible for a user to override this check, however this is not recommended.

“Is there a way of blocking the cross-domain image issue without removing support for cross-domain textures or imposing CORS?”

There would almost certainly be ways of doing it; however it would likely cause significant issues to the usability of any platform which tried. For example the shaders could be prevented from accessing texture data completely; this would seem somewhat overkill and remove most of the benefits of the programming model. You also cannot just eliminate loop constructs in the shaders; the render time is directly related to how many pixels you need to draw, therefore one trick would be to scale a polygon relative to the brightness of a pixel you want to read, that would increase the number of times the fragment shader is executed which would produce a measurable



time difference. Therefore Context would recommend the use of a mechanism to manage cross-domain images for example the requirement of CORS within WebGL.

“What responses have you had from the Browser vendors?”

Context has reported these issues and other vulnerabilities to the Mozilla Security group who has raised a number of internal bug reports regarding the issues that we have found, including issues that we have not publicly disclosed. They have also passed the information onto Google for Chrome. The Mozilla Security Group has been very receptive to the issues that we have raised and have been very responsive to our concerns. They have also passed the information onto Google for Chrome.



Blog 3 - WebGL – More WebGL Security Flaws

16th June 2011

Summary

In this blog post Context demonstrates how to steal user data through web browsers using a vulnerability in Firefox's implementation of WebGL. This is a continuation of our research into serious design flaws that could affect any browser which implements WebGL, currently Chrome and Firefox.

Context has been researching the new 3D graphics technology, WebGL, which allows web pages to draw fast 3D graphics in a similar manner to computer games. This exciting technology has the capability to deliver a much richer experience to web users.

However, to enable this impressive breakthrough in online technology, web browsers (currently Chrome and Firefox) have had to expose low level parts of their operating systems which previously could not be directly accessed by potentially malicious web pages, thus creating a number of potential security vulnerabilities.

Context identified this (and other) issues with WebGL by evaluating Chrome and Firefox WebGL implementations against the conformance test suite devised by Khronos, the consortium which draws up the WebGL specification. We have established that none of the current implementations comply with this standard.

Furthermore, Context's research found that Khronos' recommended defence against the DoS issue (WebGL_ARB_robustness) is not fit for purpose. First, only certain chipsets and operating systems (NVIDIA on Windows and Linux) support this feature. Moreover, this extension only offers mitigation, not a comprehensive solution to WebGL DoS issues.

In the video below, we show how anyone running Firefox 4 with WebGL support is vulnerable to having malicious web pages capture screenshots of any window on their system. These screenshots could be of other web pages, the user's desktop and other applications that run on their system.

Background

In our first [blog](#) Context outlines serious security concerns related to the use of WebGL. We were able to steal images from other web pages and crash people's machines (Denial of Service, or DoS) from a malicious website. These examples showed the danger of allowing malicious code to run on graphics cards which were never designed to defend against this threat. After reviewing our previous work Firefox has now removed support for cross-domain images (<https://hacks.mozilla.org/2011/06/cross-domain-webgl-textures-disabled-in-firefox-5>); while Khronos is updating the WebGL specification to include protection from DoS (using a new OpenGL extension GL_ARB_robustness) and Cross-Origin Resource Shading (CORS) attacks (<http://www.khronos.org/news/permalink/webgl-security>). The fact that it is doing so begs the question as to whether this technology was specified, designed and implemented with security in mind.

The problems we identified in our first blog were examples of the types of issues that can be created as a result of WebGL use. It is not totally unexpected or unusual for there to



be security issues associated with a new technology, but it is crucial that the standard and the correct mitigation processes are quickly adjusted once such problems are identified to ensure that end user security is not compromised. To this end Context reviewed the conformance tests that Khronos has provided for WebGL vendors to use in assessing compliance to the standard. Through this work Context has discovered that neither Chrome nor Firefox passed the Khronos tests, including a number that are directly related to security. Context then explored the consequences of one of the failed conformance tests: the issue it identified allowed us to extract images containing data from the user's desktop and from other web browser sessions such as authenticated pages. This issue was specific to Firefox and will be fixed in the next version of the browser.

If you are concerned by WebGL based attacks see our [FAQ](#) for details on whether you are vulnerable and if so how to protect yourself.

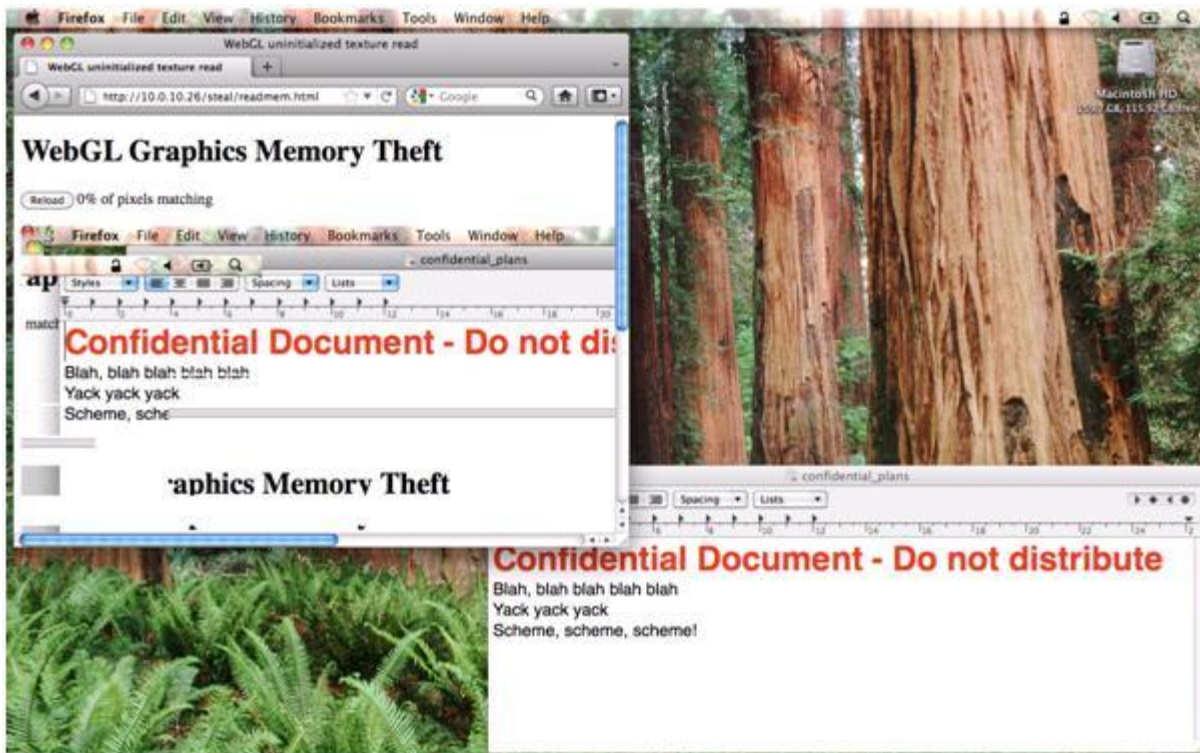


Figure 1 - Stealing Graphics Memory from the Desktop

Cross-platform testing against the Khronos solution

One important aspect of our research has been to examine the testing standards that exist to help ensure the effective operation of WebGL, to determine the extent to which they help or hinder efforts to ensure end user security.

To this end, Context set up machines to test responses from WebGL and Khronos to the issues we have already raised about security and WebGL. We tested Google Chrome and Mozilla Firefox using different operating systems and graphic cards.

We performed three sets of tests:



1. Each browser-operating system-graphics combination was inspected for the GL_ARB_robustness extension. This is Khronos' proposed solution to fix the denial of service issues we had previously reported. It is interesting to see how widely the extension is deployed today.
2. We ran the proof of concept (as mentioned in the original [blog](#)) to see if users are vulnerable even with the robustness extension.
3. Finally we ran Khronos' own WebGL conformance suite (<http://www.khronos.org/webgl/wiki/Testing/Conformance>), which has been developed to try and test adherence to the WebGL specification. It should be expected that the browser developers would be running this suite as part of their standard release testing, so it ought to test a suitable range of features to determine whether or not there is significant platform dependence.

We used one of each of the main graphics cards (Intel, NVidia, and ATI/AMD), on Windows XP SP3, Windows 7 and Ubuntu Linux 11.04. We also tested one Mac which had an NVidia 9400M graphics card. For the Windows platforms the latest driver available at the time was installed; for Ubuntu and OSX we used the latest proprietary driver available with the distribution. The tests were split into two groups. On Windows platforms Firefox and Chrome use ANGLE (which is an OpenGL ES implementation on top of Direct3D); while on Linux and OSX OpenGL is used directly. This is why only NVidia was tested on Linux (as we did not have anything but an NVidia based Mac) while Chrome had WebGL force-enabled to produce a comparison between Windows platforms. The following table contains the driver versions on each platform which was tested.

| Platform | ATI Radeon HD3400 Driver | NVidia Geforce 310 Driver | Intel G41 Driver |
|----------------|--------------------------|---------------------------|------------------|
| Windows XP SP3 | 11.5 | 6.14.12.7061 | 6.14.10.5355 |
| Windows 7 | 11.5 | 8.17.12.7061 | 8.15.10.2302 |
| Mac OSX 10.6.7 | - | (9400M) 6.2.6 | - |
| UBuntu 11.04 | - | 270.41.06 | - |



The following table shows on which platforms and drivers Chrome and Firefox WebGL implementations can expect to find the GL_ARB_robustness extension implemented.

| Platform | ATI | NVidia | Intel |
|----------------|---------------|---------------|---------------|
| Windows XP SP3 | Not supported | Supported | Not supported |
| Windows 7 | Not supported | Supported | Not supported |
| Mac OSX 10.6.7 | - | Supported | - |
| UBuntu 11.04 | - | Not supported | - |

The following table is the results of running Chrome and Firefox against the denial of service proof of concept on each platform.

| Platform | ATI | NVidia | Intel |
|----------------|---|--|--|
| Windows XP SP3 | Screen blacks out after approximately 3 seconds before recovering with a window saying that the ATI VPU Recover has reset the graphics accelerator. | Desktop freezes. OS eventually crashes. | Desktop freezes. OS eventually crashes. |
| Windows 7 | Screen goes black after a few seconds and then recovers with a warning about display driver not responding and has been recovered. | Screen goes black after a few seconds and then recovers with a warning about display driver not responding and has been recovered. | Screen goes black after a few seconds and then recovers with a warning about display driver not responding and has been recovered. |
| Mac OSX 10.6.7 | - | Desktop freezes, requires a reboot to restore functionality | - |



| Platform | ATI | NVidia | Intel |
|--------------|-----|--|-------|
| UBuntu 11.04 | - | Screen goes black after a few seconds and then recovers. | - |

The conformance suite consists of 144 main tests, each of which can contain one or more sub tests. The table below only records main test failure, because generally if one subtest fails most fail with a similar error so it makes sense to record at this level. Each browser, platform and graphics card is reported separately.

| Platform | Browser | Conformance Results Failures (out of 144 tests) | | |
|----------------|--------------------|---|--------|-------|
| | | ATI | NVidia | Intel |
| Windows XP SP3 | Chrome 11.0.696.71 | 15 | 15 | 8 |
| | Firefox 4.0.1 | 23 | 22 | 22 |
| Windows 7 | Chrome 11.0.696.71 | 15 | 15 | 8 |
| | Firefox 4.0.1 | 23 | 22 | 22 |
| Mac OSX 10.6.7 | Chrome 11.0.696.71 | - | 6 | - |
| | Firefox 4.0.1 | - | 27 | - |
| UBuntu 11.04 | Chrome 11.0.696.71 | - | 4 | - |
| | Firefox 4.0.1 | - | 21 | - |

A few conclusions can be drawn from these results. At present it seems only NVidia supports the GL_ARB_robustness extension, which on XP doesn't stop the denial of service attack (this is likely to be because the extension is not enabled by any web browser). But only Windows XP and OSX configurations suffered from the denial of service condition. The driver for the ATI card on XP seems to have implemented its own



reliability mechanism, managed to reset the device and was then able to resume operation. This would seem to indicate a better result than perhaps could be expected. Furthermore Mozilla have stated in a recent post that “forthcoming GL_ARB_robustness_2 extension will help even more” (<http://blog.jprosevear.org/2011/05/13/webgl-security/>) which rather undermines any claims made about the benefits offered by the earlier version.

As for the conformance test results, it would seem that no browser can claim to offer full support for WebGL, based on the Khronos specification, as all fail some part of the conformance suite. Comparison of the failures between similar platforms reveals a degree of commonality. For example, Chrome on XP exhibits the same bugs as Chrome on 7. Between disparate platforms the browser must be exposing some aspect of the underlying graphics implementation to the web page for problems to arise, even if it is something trivial such as incorrect return values. The results on OSX are especially anomalous: Chrome has the second best result out of all the platforms yet Firefox has the worst.

The Case of the ATI Failure

The results of the conformance tests are one thing, but where does security come into the picture? Well as Khronos indicates at <http://www.khronos.org/webgl/wiki/Testing/Conformance> the primary purpose of the conformance test is to find incompatibilities between platforms. But some of the tests do also have a security assessing function.

The structuring of the WebGL conformance test suite ensures that no one test is marked out as being any more significant than any of the others. They are all seemingly arbitrary test cases with no specific purpose in mind from a standard adherence point of view. That makes this suite quite different from, say, the CSS conformance test suite (<http://test.csswg.org/suites/css2.1/20110323/xhtml1/toc.xht>) which has tests for each important parts of the standard, all referred directly back to the original description in the standard.

So are any security issues addressed directly in this testing regime? From the table of the Windows platforms all the tests seem to match for a given browser on a particular card, except for one extra failure, for ATI on XP in Firefox. From a statistical point of view it doesn't significantly change the results for Firefox on XP, up from around 15% failure to 16%.

If the conformance suite did relate directly to the standard it might have been possible to see that the failing test was actually testing for a critical security requirement, in this case testing to ensure that that certain operations do not expose uninitialised data to an untrusted web page. But it doesn't, so the test becomes lost in the noise of 22 other issues, most of which relate to the return of trivial incorrect error codes.

In fact, this wasn't just a failure on ATI cards on XP, it was an issue that just happened to become visible during this test. Further inspection confirmed it also affected NVidia cards on XP in the same way. And it wasn't limited to the Microsoft platform either: OSX also had the issue and the way in which the graphics card is used under OSX actually made the problem even more significant.



It turned out that this “failure” was known to Mozilla developers, who seem to have chosen to ignore the results of these tests on XP and OSX. Perhaps if the conformance suite had made it clear that failing this test could indicate a serious security issue the developers might have flagged this up earlier?

The bug itself was the result of a relatively simple mistake in clearing graphics memory which did not take into account some of the pre-existing state of the graphics layer. It isn't clear why Linux or Windows 7 were not affected by this; it seems likely that it was down to some specific implementation feature which eliminated the issue. Context contacted Mozilla's security team who have committed themselves to resolving this issue in time for the next release (expected 21st June 2011).

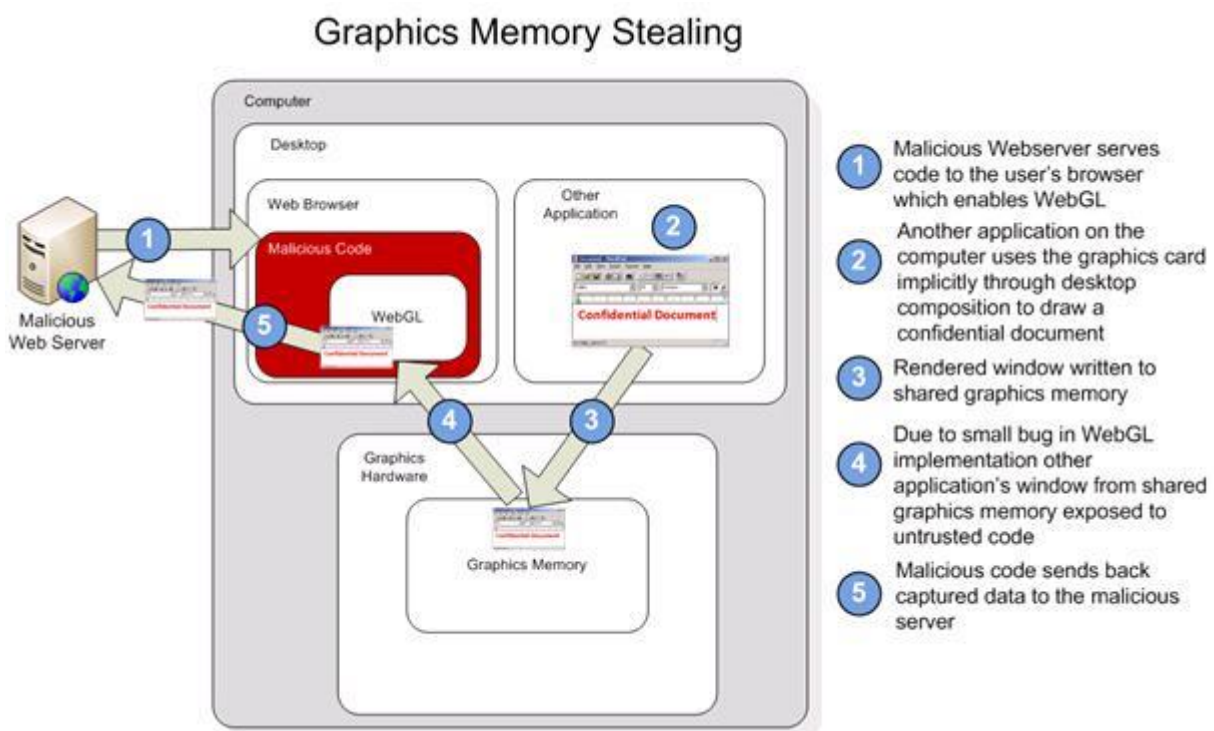


Figure 2 Graphics Memory Stealing

Exploiting the Vulnerability

The vulnerability we discovered enables any graphics image that has been displayed on the system to be stolen by an attacker by reading uninitialised data from graphics memory. This is not limited to WebGL content but includes other web pages, a user's desktop and other applications.

To prove that it was possible to exploit this bug, we developed a proof of concept exploit that would be able to steal web page data. As we feel this solution was quite novel it is presented here for interest. From the original vulnerability, an attacker would have no control over which specific region was read. The memory accessible was allocated based on a number of different properties, but it wasn't necessarily predictable.



An attacker would face two main challenges in trying to exploit this vulnerability: getting the data an attacker wants into graphics memory, and then finding it again afterwards.

As an example target, let's use an authenticated LinkedIn session. To be certain of getting the data which would be of interest to an attacker into the graphics memory we used Firefox's 2D compositing features. This means Firefox will render the element to a texture and upload it to the graphics card to speed up compositing, so populating the graphics memory with the data we wish to steal.

The second issue seems a bit more difficult to solve. Certainly it would be possible to just capture any data we can and use a human to decipher whether it is the data we required, but that isn't especially efficient. Instead we want to somehow key the data in memory so that when we read it out we can be reasonably certain it is the data we requested. Obviously we cannot directly manipulate the contents of the IFRAME (otherwise this would be a pointless exercise), so another approach is necessary.

This is where SVG filters come into play. It is possible to create a filter which will manipulate the colour values of the texture we are trying to capture. As this is pre-composited the texture in video memory is coloured appropriately. The following SVG filter when applied to the IFRAME will convert each pixel to greyscale and store the value in the blue channel. Then it sets the red channel to a known value between 0 and 1. For each run the red channel value is randomly generated and video memory searched for that colour value. So when we find a significant proportion of pixels with that red channel value we know we have struck a seam of interesting data.

```
<svg>
<filter>
  <!-- Convert to greyscale in the green channel -->
  <feColorMatrix style="color-interpolation-filters:sRGB"
    values="0 0 0 0 0
           0 0 0 0 0
           0.333 0.333 0.333 0 0
           0 0 0 1 0"/>
  <feComponentTransfer>
    <!-- Set intercept to the red channel value we want -->
    <feFuncR type="linear" slope="0" intercept="1"/>
  </feComponentTransfer>
</filter>
</svg>
```

Of course, there are no guarantees that the data is in any sort of order. For our proof of concept this wasn't that much of an issue, because most of the time the data was at least readable. But the technique could be taken further to encode positional information into the texture, to allow reconstruction of the original arrangement. One improvement we did make was to compress the colour information into the unused blue channel. This made it possible to extract a reasonable looking colour image from the captured data.

The following is a video showing the process in action. In the video we show the loading of LinkedIn into an IFRAME and then the application of the filter to tag the memory. The image is then rotated to spray the graphics card's memory. By exploiting the graphics



memory leakage bug we can then trawl the memory for the colour we have tagged the image with and extract an image of the web page. This image is then in our control and can be sent to a web server under the attacker's control. To exploit this issue in the real world a malicious link would be sent to a user and this process of stealing images of authenticated web pages would occur without the user's knowledge.

Conclusions

It would be unreasonable to expect full conformance to the complete specification of a new standard: there are always likely to be edge cases. But, as we have stressed before, some areas of WebGL need to be carefully implemented to prevent security issues arising and unfortunately in this case, because security-related conformance tests are not clearly identified, it is not possible to determine if an implementation is secure. This has been a contributory factor in security issues being missed by developers of the current browser implementations of WebGL, which has in turn created serious security flaws. Browser developers should start banning non-conformant configurations as they are identified until the security issues that have been highlighted are resolved.

Context therefore recommends that users and system administrators disable WebGL.



Contributions

The project was a collaboration between the following members of the Context team:

- James Forshaw
- Michael Jordan
- Paul Stone



About Context

Context Information Security is an independent security consultancy specialising in both technical security and information assurance services.

The company was founded in 1998. Its client base has grown steadily over the years, thanks in large part to personal recommendations from existing clients who value us as business partners. We believe our success is based on the value our clients place on our product-agnostic, holistic approach; the way we work closely with them to develop a tailored service; and to the independence, integrity and technical skills of our consultants.

The company's client base now includes some of the most prestigious blue chip companies in the world, as well as government organisations.

The best security experts need to bring a broad portfolio of skills to the job, so Context has always sought to recruit staff with extensive business experience as well as technical expertise. Our aim is to provide effective and practical solutions, advice and support: when we report back to clients we always communicate our findings and recommendations in plain terms at a business level as well as in the form of an in-depth technical report.





Context Information Security Ltd

London (HQ)

4th Floor
30 Marsh Wall
London E14 9TP
United Kingdom

Cheltenham

Corinth House
117 Bath Road
Cheltenham GL53 7LS
United Kingdom

Düsseldorf

Adersstrasse 28
1. Obergeschoss
D-40215 Düsseldorf
Germany

Melbourne

Level 9, 440 Collins St
Melbourne
Victoria 3000
Australia