AntiviruXSS

How We XSSed 8/9 Top AV Vendors

by
Strukt @strukt93
Brute @brutelogic

Table Of Contents

1. Introduction

1.1 A few words

Antivirus software (abbreviated as AV) has been very frequently used by millions of people worldwide in the past decade. Their main job was to identify and remove viruses from the computer they are running on.

Over the years, the work of AVs has extended to more than just catching and removing viruses from computers. Modern AVs are designed to protect from more types of attacks such as adware, ransomware, spyware, trojan horses, backdoors, worms and rootkits.

While AV softwares are doing quite a good job protecting from the previously mentioned types of attacks, their vendors seem to forget or ignore the protection of their own web applications from the simplest of attacks.

1.2 What is this paper about

This paper is about how the authors attempted and succeeded to find Cross-Site Scripting flaws in the web applications of almost all the top AV software vendors (8 out of 9).

Cross-Site Scripting (XSS) is a type of client-side attacks where the attacker takes control over the victim’s browser via the injection of malicious code (usually javascript).

This type of attacks allows the attackers to do a bunch of things depending on the context of the attack. For example, if the XSS vulnerability lies in the login page of some website, the attacker may be able to steal the user credentials once the user enters their data and presses the “Login” button.

Another example is when the server sends an ill configured session cookie, where it’s left to be manipulated by client-side scripts. The attacker then can steal that cookie via an XSS vulnerability that exists on any page on the web application that also contains the session cookie.

Cross-Site Scripting also allows attackers to redirect users to malicious pages on the internet and allows them to force the browser to download malicious files.

For more information regarding Cross-Site Scripting basics see: https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

For articles regarding XSS attacks see: http://brutelogic.com.br/blog

2. Rating criteria

The authors of this paper have decided to rate each of the findings based on its severity and difficulty to find, each of the ratings will lie on a scale between 1 and 5.

Each of the findings in the AntiviruXSS section will have a rating similar to the following model:

Severity: ⅖
Difficulty: ⅘

Additionally we have decided to rate each of the vendor's responses based on how serious it is, which is determined by how fast their initial response was and how much they seemed to be interested in their follow ups. Again, each of those ratings will lie on a scale between 1 to 5.

Each of the responses in the vendor responses section will have a rating similar to the following model:

Speed: ⅖
Interest: ⅘

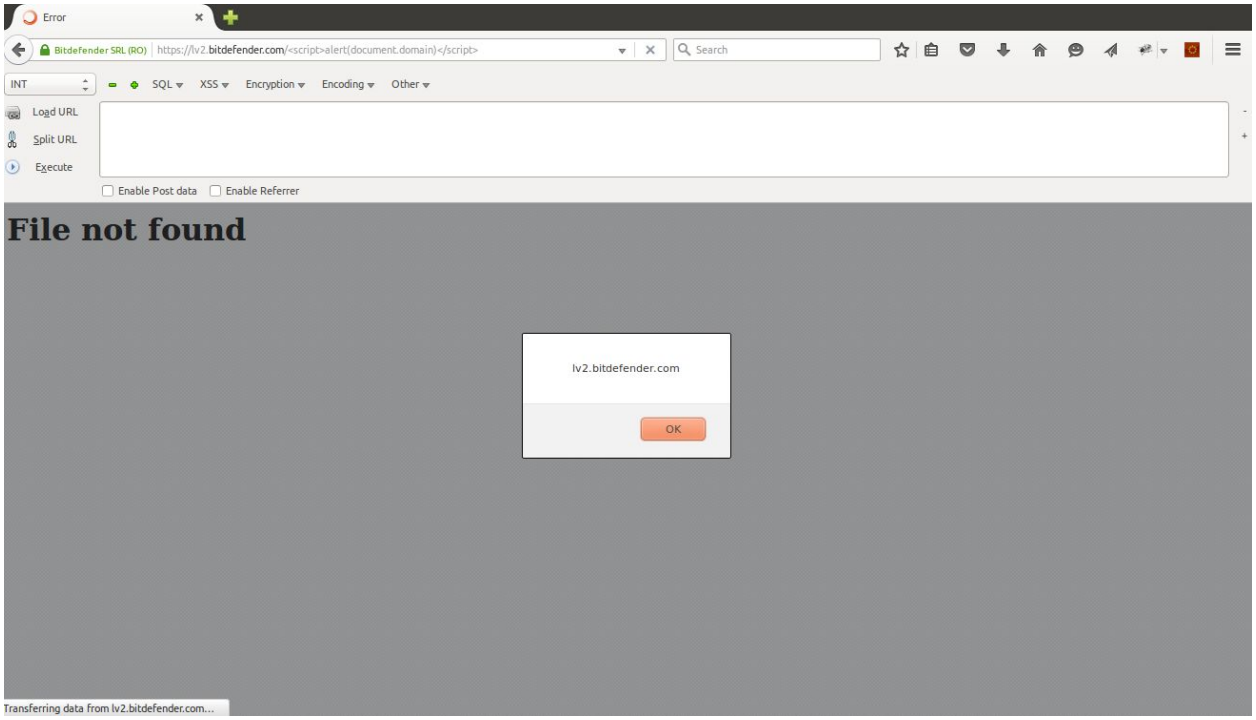
3. AntiviruXSS

This is the main section of the paper, where we explain each of the findings and the payload(s) used to trigger an alert box as a Proof-of-Concept (PoC) for each of them. We will list the findings sorted by difficulty.

3.1 BitDefender

Affected Subdomain: lv2.bitdefender.com

The subdomain mentioned above was susceptible to XSS because of the 404 error page on the server. It did not sanitize the path name from the URL thus adding a simple XSS vector like <script>alert(document.domain)</script> was enough to trigger an alert.



Rating
Severity: ⅖
Difficulty: ⅕

This XSS scored ⅖ in severity as the subdomain has the same cookies the main subdomain www.bitdefender.com had thus the same level of threat if the XSS was on the main subdomain. It scored ⅕ in difficulty as there was no protection at all against XSS.

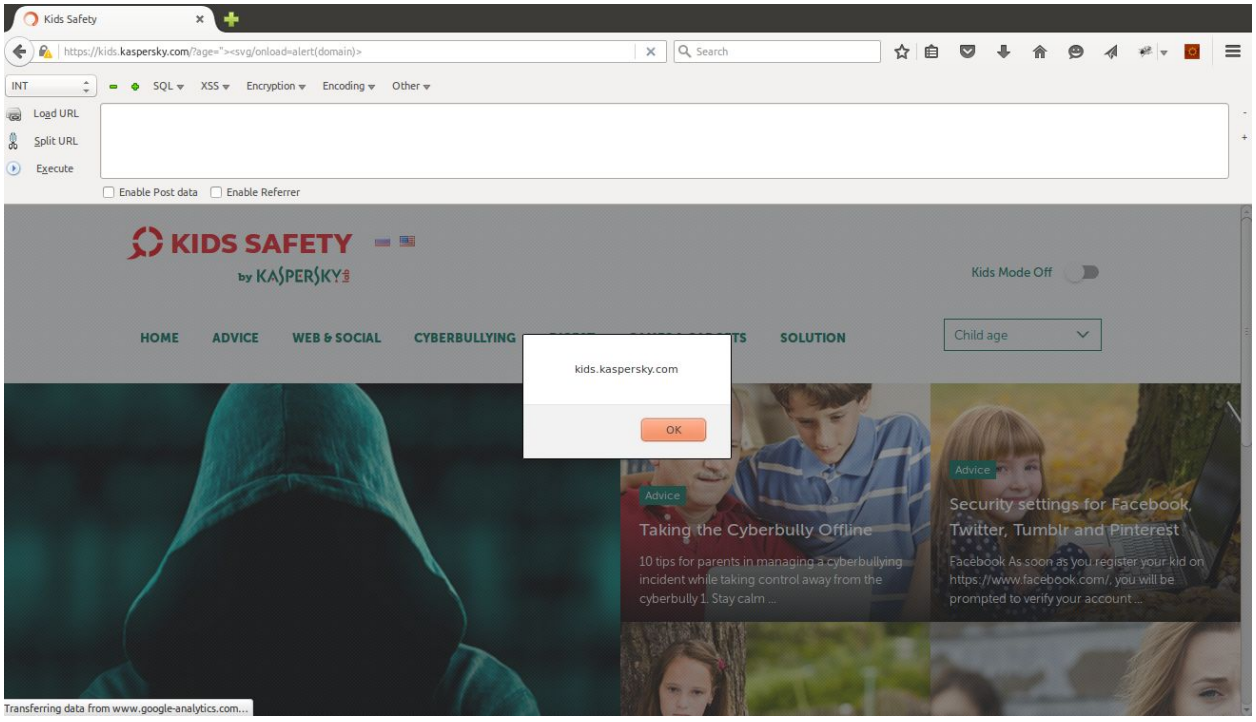
Final Payload:

https://lv2.bitdefender.com/<script>alert(document.domain)</script>

3.2 Kaspersky

Affected Subdomain: kids.kaspersky.com

The above subdomain was vulnerable to XSS through a GET parameter called ‘age’, which is supposed to hold the kid’s age. The parameter had absolutely no protection against XSS and allowed simple vectors like “><svg/onload=alert(domain)> to execute an alert on the subdomain.



Rating
Severity: ⅖
Difficulty: ⅕

Just like BitDefender's this XSS scores ⅘ in severity because the subdomain has the same cookies that www.kaspersky.com does.

It also scores ⅕ in difficulty because there's no protection against XSS attacks.

Final Payload:
https://kids.kaspersky.com/?age="><svg/onload=alert(domain)>

3.3 Panda Security

Affected Subdomain: download.pandasecurity.com

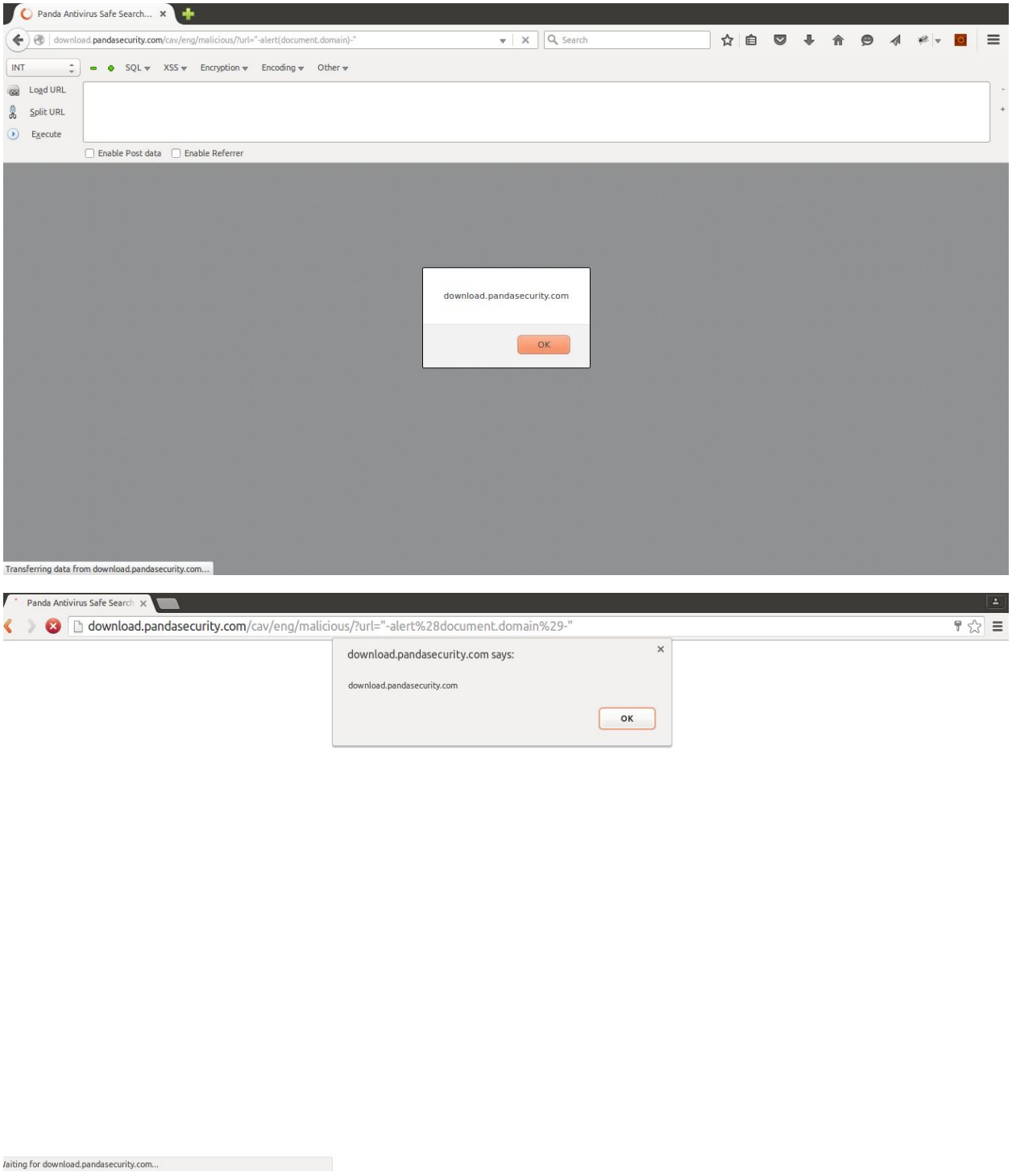
The above subdomain was vulnerable to XSS via the GET parameter "url", which was lying at http://download.pandasecurity.com/cav/eng/malicious/

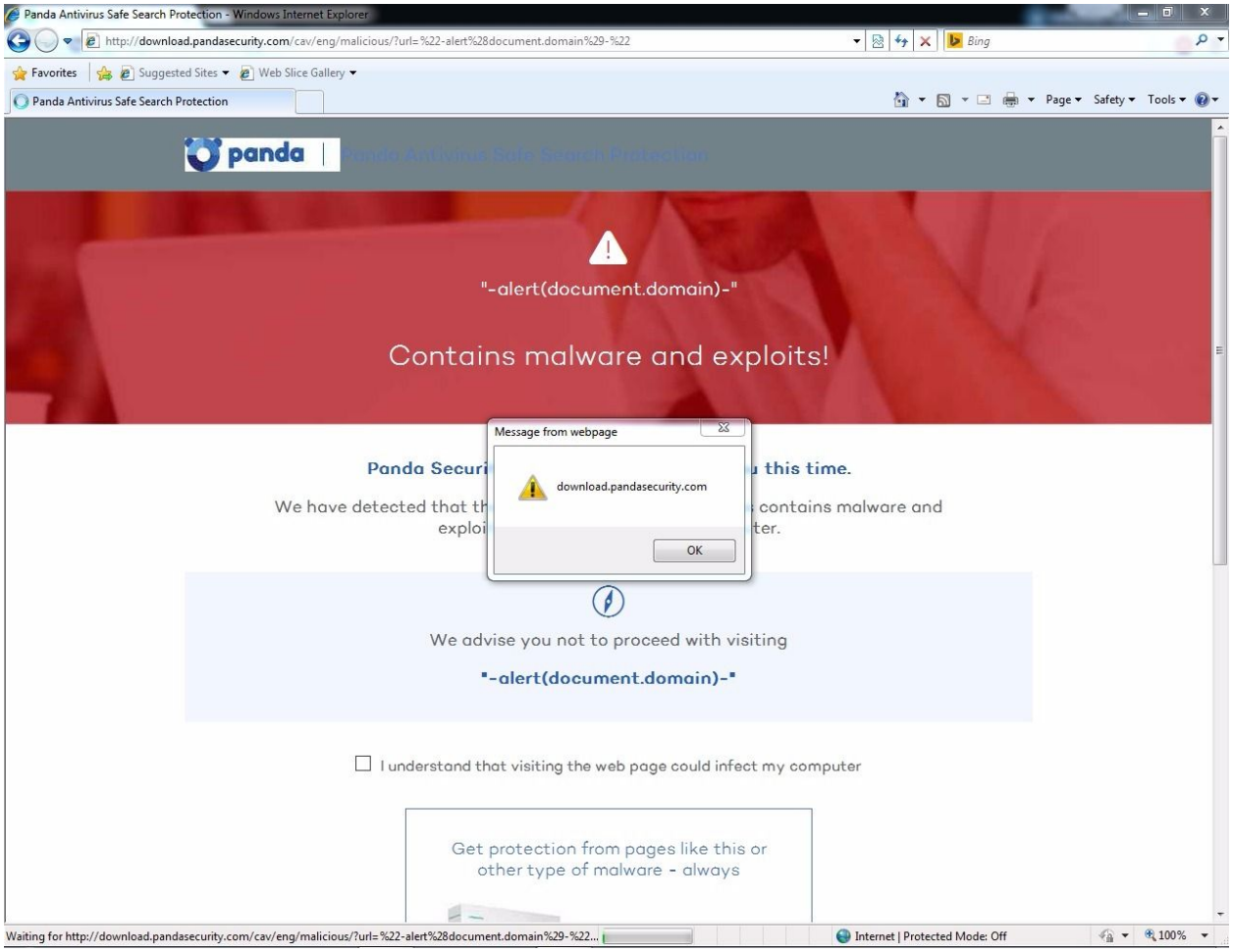
This parameter's value appeared 3 times in the source of the page, once in a script context where it was held between two double quotation marks and twice between <h1> tags.

The parameter's value was not escaped or encoded by any means and so we had two options to exploit this vulnerability.

Our first option was to directly inject a payload with an HTML tag like <svg onload=alert(domain)>. The other option was to break out of the string value of the parameter and "subtract" the alert function from it using "-alert(document.domain)-". We chose the

second option because it is shorter and works on all major browsers including Mozilla FireFox, Google Chrome, Internet Explorer and Safari as seen in the screenshots below.





Panda Security Technology protected you this time.

We have detected that the page you are trying to access contains malware and exploits that could infect your computer.

We advise you not to proceed with visiting

"-alert(document.domain)-"

I understand that visiting the web page could infect my computer

Rating
Severity: ⅗
Difficulty: ⅖

This XSS scored ⅗ in our opinion due to the same circumstances like the two previous examples but in this case none of the cookies were of HttpOnly type. It means that all the cookies can be stolen by client-side scripts thus a bigger risk.

This XSS also scored ⅖ on our difficulty scale because even though there was no protection against XSS, calling the subdomain download.pandasecurity.com without specifying the path to the vulnerable page results in a 403 (Forbidden) error and some more research had to be done first to find the vulnerable one.

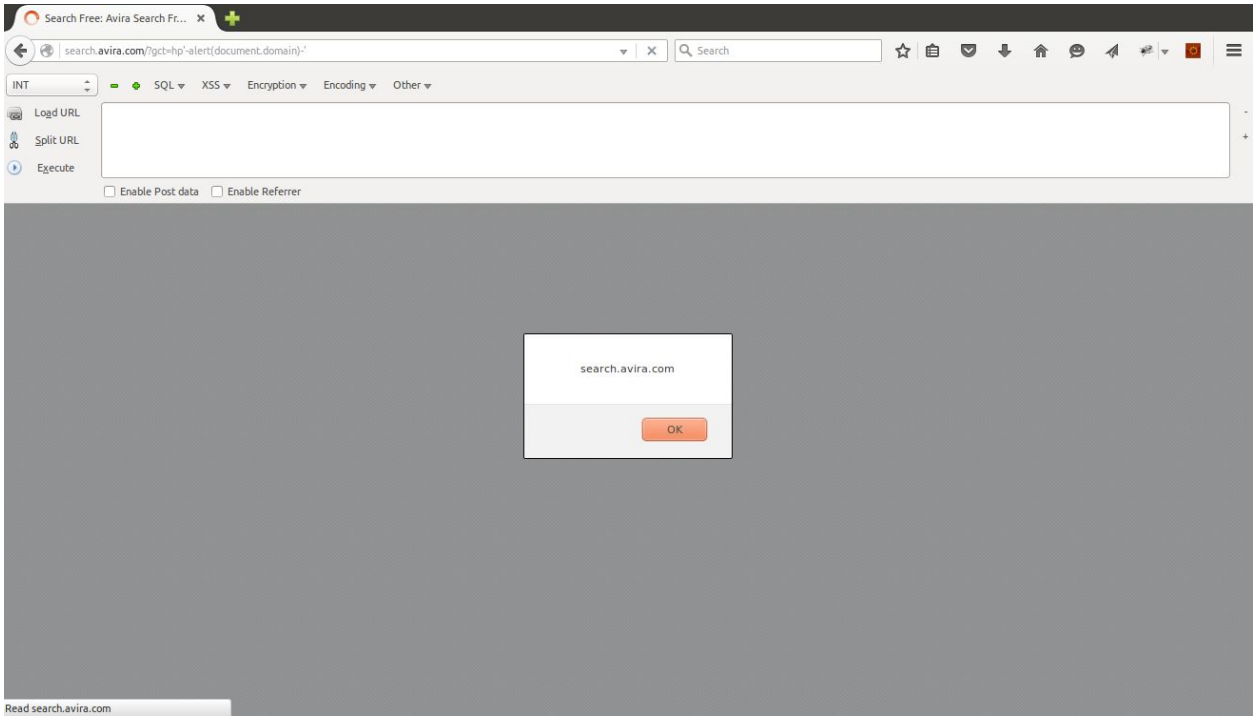
Final Payload:
http://download.pandasecurity.com/cav/eng/malicious/?url="-alert(document.domain)-"

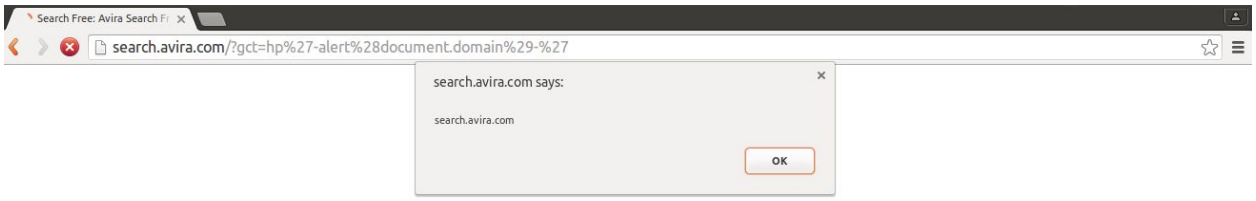
3.4 Avira

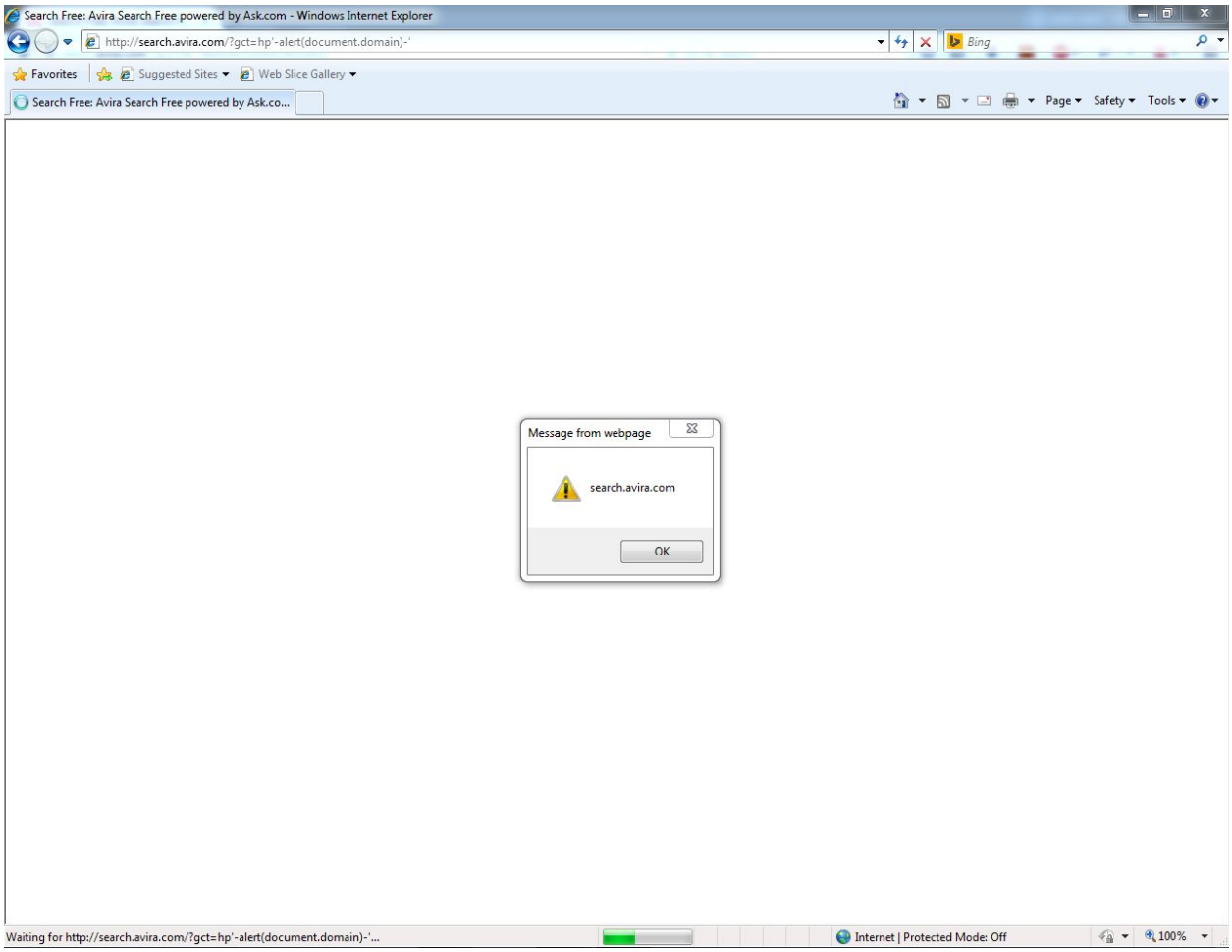
Affected Subdomain: search.avira.com

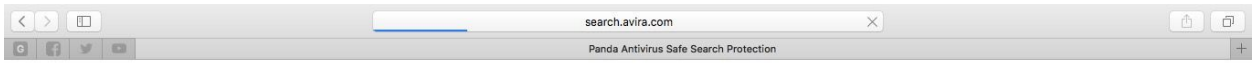
This subdomain was vulnerable to XSS via a GET parameter called “gct” which value appeared 6 times in the source of the page, one of them without any escaping or encoding in a script context while in the five other cases it was very well sanitized.

Again we had to make the same choice between the two approaches to XSS this subdomain and of course again we chose the shorter and universal solution. Screenshots:






Search Free: Avira Search Free powered by Ask.com - Windows Internet Explorer
http://search.avira.com/?gct=hp'-alert(document.domain)-'
Bing
Favorites
Suggested Sites
Web Slice Gallery
Search Free: Avira Search Free powered by Ask.co...
Page
Safety
Tools
Message from webpage
search.avira.com
OK
Waiting for http://search.avira.com/?gct=hp'-alert(document.domain)-'...
Internet | Protected Mode: Off
100%


search.avira.com
Panda Antivirus Safe Search Protection


From "http://search.avira.com":
search.avira.com
OK

Rating
Severity: ⅗
Difficulty: ⅗

This XSS scored ⅗ in severity because it's typically the same case as Panda Security, all the cookies are accessible by client-side scripts including the PHPSESSID which is the PHP session cookie.

It also scored ⅗ in difficulty because it took us some time to find the vulnerable parameter as opposed to Panda Security's case, where the vulnerable parameter was easily caught but the path to the vulnerable page was the one that took time to be found.

Final Payload:
http://search.avira.com/?gct='-alert(document.domain)-'

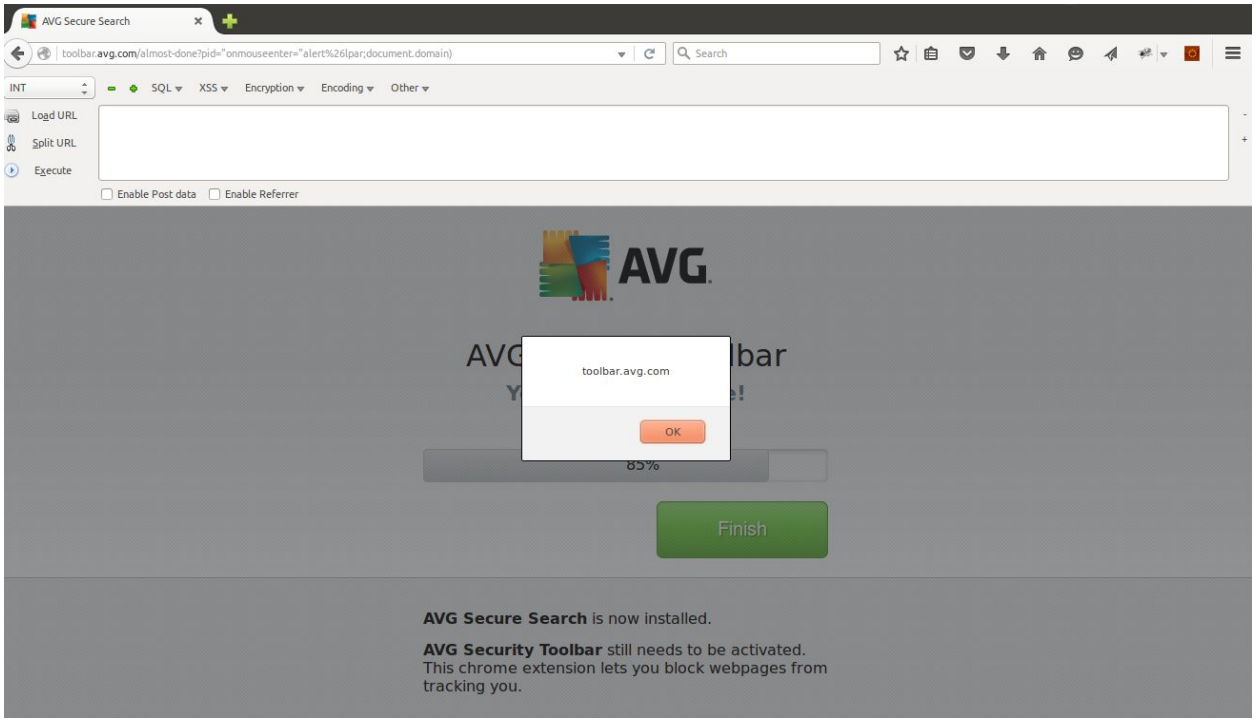
3.5 AVG

Affected Subdomain: toolbar.avg.com

The subdomain above was actually different compared to the 4 previous examples. The web application was running on Microsoft's ASP.NET which default anti-XSS filter detects the XSS attempt if the "<" character is followed by an alphabetical one (A-Z and a-z).

The web application was vulnerable to XSS through two GET parameters named "op" and "pid" whose values were set as the

value of “class” parameters of two tags in the source of the page. We XSSed the page using the “onmouseenter” event handler with details after the screenshot.



Rating
Severity: ⅖
Difficulty: ⅘

This XSS scored ⅖ in severity as the most important cookies were protected by the HttpOnly option which keeps it from being stolen by client-side scripts.

As in difficulty, this one deserves to score a ⅘ on our scale because of the following reasons. First, the path to the affected page was not that obvious and needed some time and search to be found. Second and more important, the use of ASP.NET as a back-end

service made it harder to XSS the page because of its native (although limited) anti-XSS filter.

This being said, we had only one option which is to break out of the “class” attribute but not the tag and then use a JS event handler to execute the XSS on the page. We then tried using the “onmouseover” event handler, but unfortunately we couldn’t because there was some WAF that blocked it.

We tried another mouse event handler, which is “onmouseenter”, and it worked like a charm. All set, only using the alert() function is left to XSS this page. Unfortunately this wasn’t actually the case because an WAF (Web Application Firewall) in place also blocked the use of the left parentheses “(“. This was quite easy to bypass by using it’s HTML encoded counterpart, which is (.

Final Payloads:
http://toolbar.avg.com/almost-done?op="onmouseenter="alert%26lpar;document.domain)

http://toolbar.avg.com/almost-done?pid="onmouseenter="alert%26lpar;document.domain)

3.6 Symantec

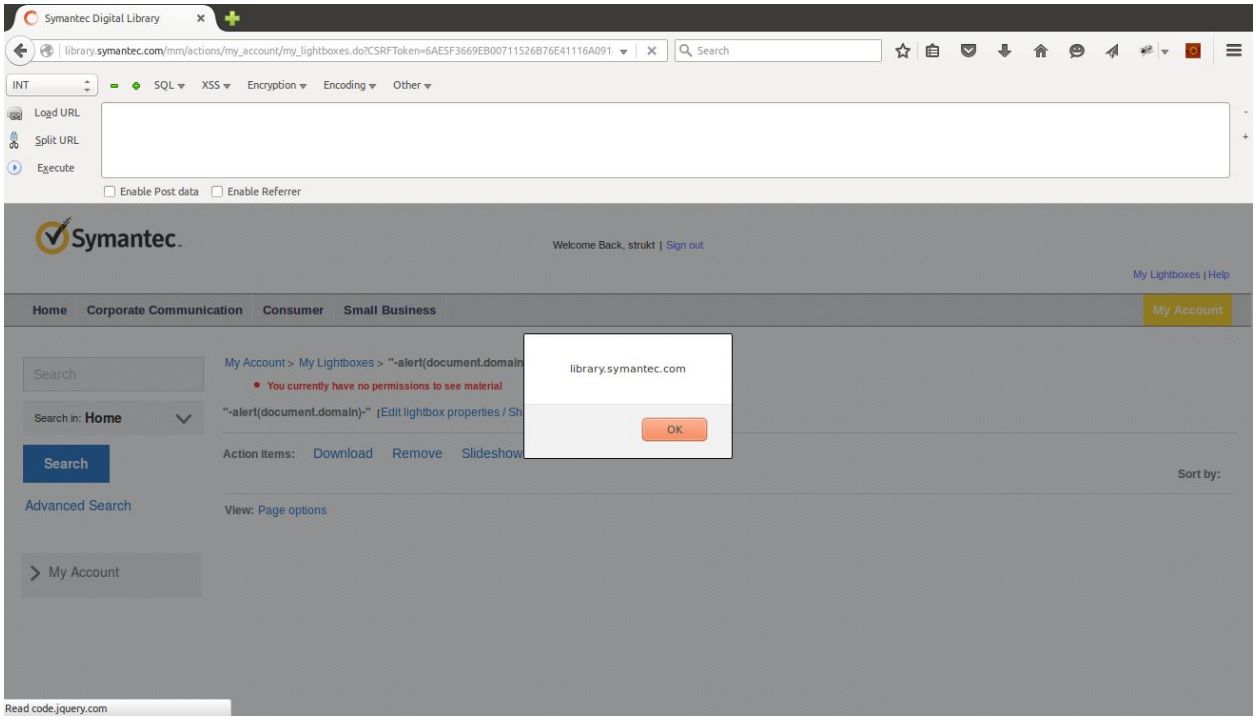
Affected Subdomain: library.symantec.com

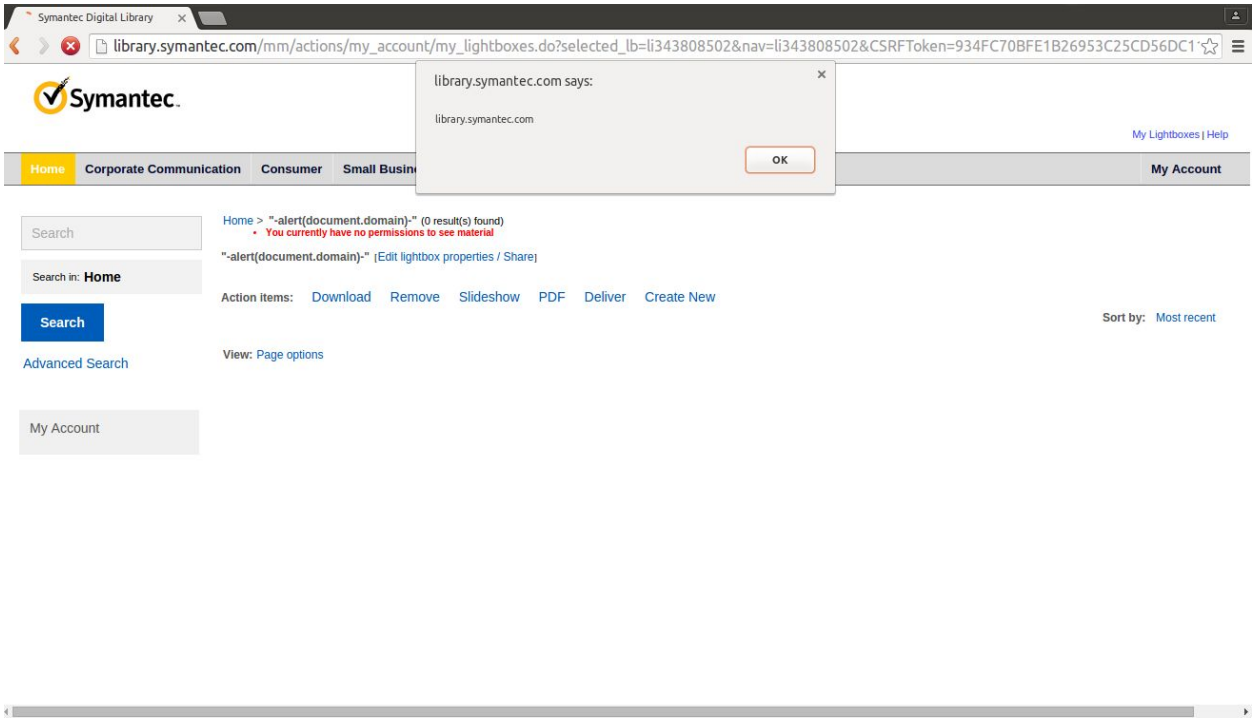
The above subdomain was actually different from all the other cases because this XSS was a stored one. Symantec didn’t build

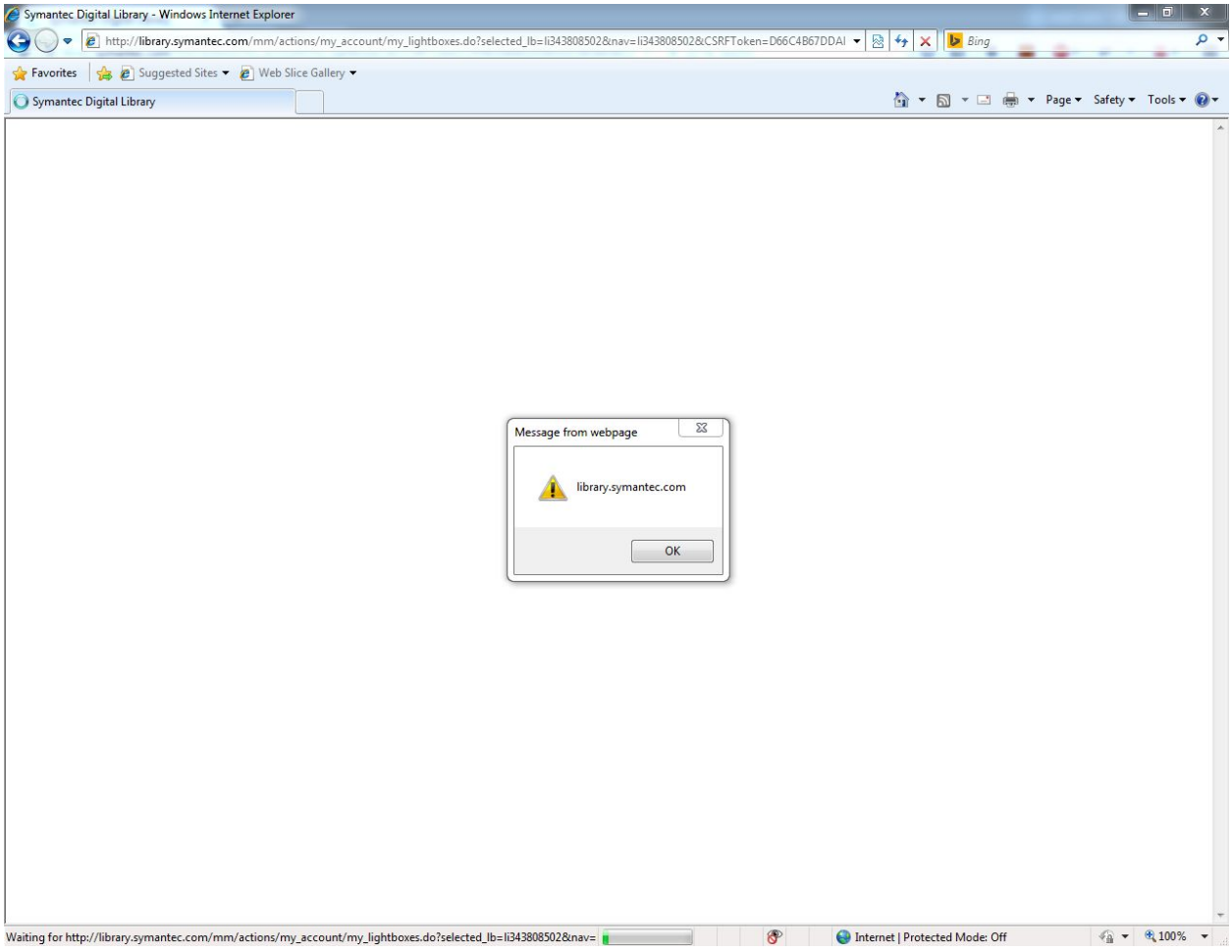
the code running on that subdomain, they were using a platform that was developed by another company called “Getty Images”. The stored XSS was affecting an aspect of the web application called “Lightboxes” where users can create their own lightboxes and share them with other users.

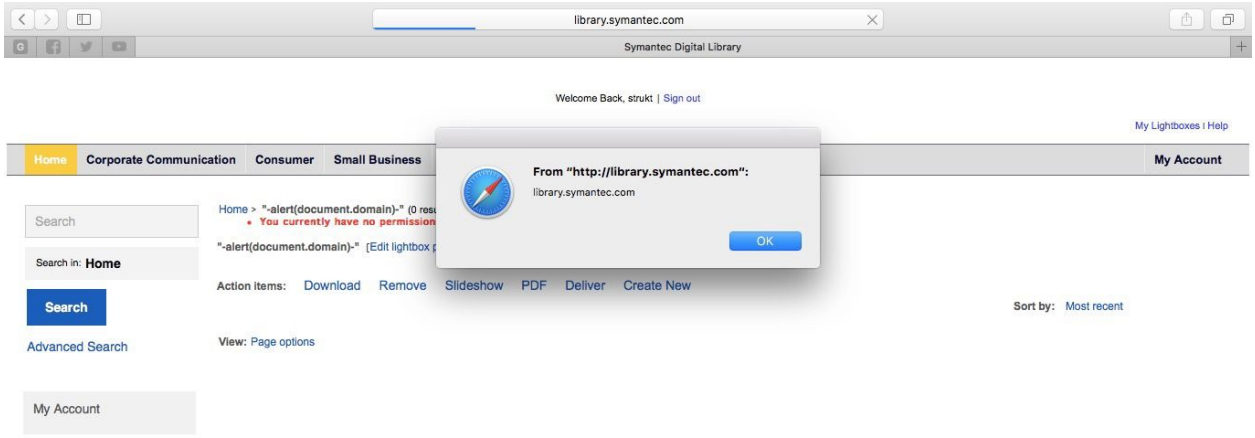
The vulnerability lied in the lightbox name: it appeared in a script context with no sanitization at all, leaving the web application vulnerable to a very devastating type of XSS attacks.

We were again to choose from two exploitation methods, choosing the shorter which works in all major browsers.


Symantec Digital Library
library.symantec.com/mm/actions/my_account/my_lightboxes.do?CSRFToken=6AE5F3669EB00711526B76E41116A091
Search
INT
SQL
XSS
Encryption
Encoding
Other
Load URL
Split URL
Execute
Enable Post data
Enable Referrer
Symantec.
Welcome Back, strukt | Sign out
My Lightboxes | Help
Home
Corporate Communication
Consumer
Small Business
My Account
Search
Search in: Home
Search
Advanced Search
My Account
My Account > My Lightboxes > "-alert(document.domain
You currently have no permissions to see material
"-alert(document.domain)-" (Edit lightbox properties / Sh
Action items: Download Remove Slideshow
Sort by:
View: Page options
library.symantec.com
OK
Read code.jquery.com


Symantec Digital Library
library.symantec.com/mm/actions/my_account/my_lightboxes.do?selected_lb=li343808502&nav=li343808502&CSRFToken=934FC70BFE1B26953C25CD56DC1
library.symantec.com says:
library.symantec.com
OK
Symantec.
My Lightboxes | Help
Home
Corporate Communication
Consumer
Small Busin
My Account
Search
Search in: Home
Search
Advanced Search
My Account
Home > "-alert(document.domain)-" (0 result(s) found)
You currently have no permissions to see material
"-alert(document.domain)-" (Edit lightbox properties / Share)
Action items: Download Remove Slideshow PDF Deliver Create New
Sort by: Most recent
View: Page options


Symantec Digital Library - Windows Internet Explorer
http://library.symantec.com/mm/actions/my_account/my_lightboxes.do?selected_lb=li343808502&nav=li343808502&CSRFToken=D66C4B67DDAI
Bing
Favorites
Suggested Sites
Web Slice Gallery
Symantec Digital Library
Page
Safety
Tools
Message from webpage
library.symantec.com
OK
Waiting for http://library.symantec.com/mm/actions/my_account/my_lightboxes.do?selected_lb=li343808502&nav=
Internet | Protected Mode: Off
100%


library.symantec.com
Symantec Digital Library
Welcome Back, strukt | Sign out
My Lightboxes | Help
Home
Corporate Communication
Consumer
Small Business
My Account
From "http://library.symantec.com":
library.symantec.com
OK
Search
Search in: Home
Search
Advanced Search
My Account
Home > "-alert(document.domain)-" (0 res
You currently have no permission
"-alert(document.domain)-" (Edit lightbox
Action items: Download Remove Slideshow PDF Deliver Create New
Sort by: Most recent
View: Page options

Rating
Severity: 5/5
Difficulty: ⅘

This XSS scored 5/5 on the severity scale and this was decided upon three factors. The first is that the XSS is a stored one which can affect lots of people and may be used to create a worm. The second is that none of the cookies, including the session cookie itself, were protected hence likely to be stolen by client-side scripts.

Finally, the CSRF token was actually stored in a variable inside a <script> tag in the source code. All that being said, the attacker can absolutely take over the account of any user once they share a maliciously named lightbox to them and they accept it.

The XSS also scored ⅘ on the difficulty scale because the subdomain wasn't so easily found. We also had to create accounts to find the vulnerability and make sure it wasn't a self-XSS. Also we had to test every stored user input in order to find that XSS.

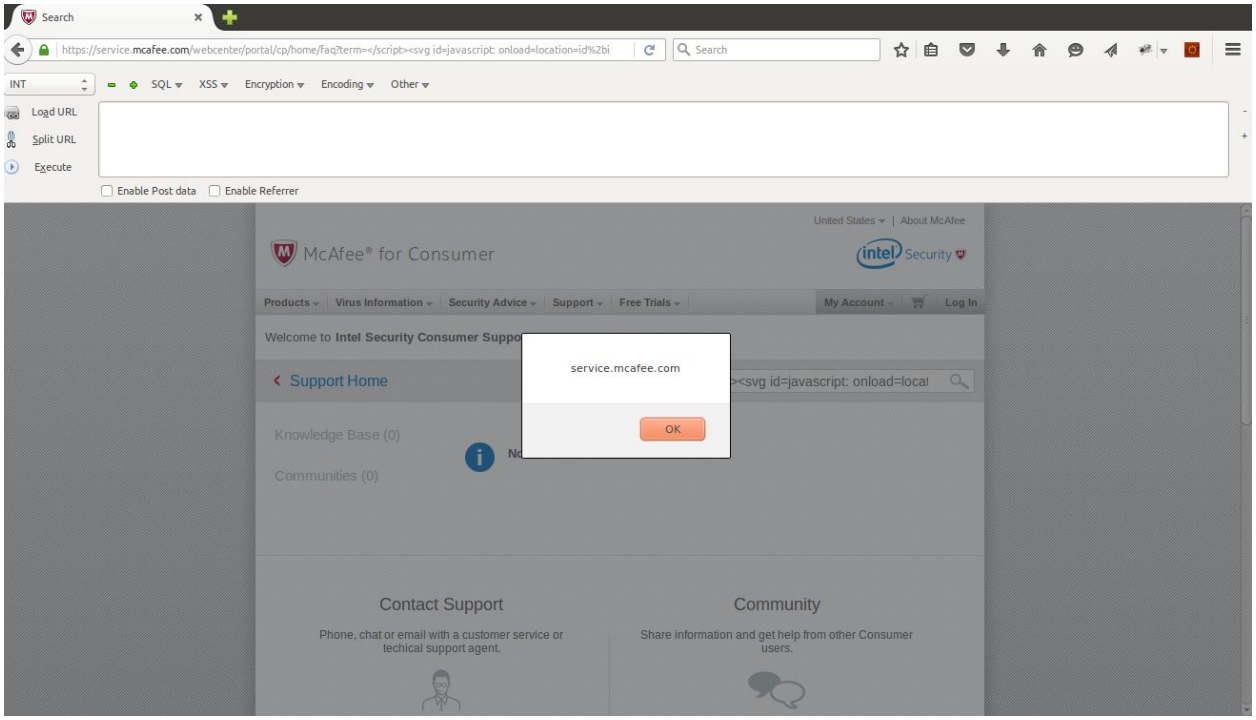
Final Payload:
In this case there's no absolute URL to be followed for the XSS to trigger, the payload for the XSS was "-alert(document.domain)-".

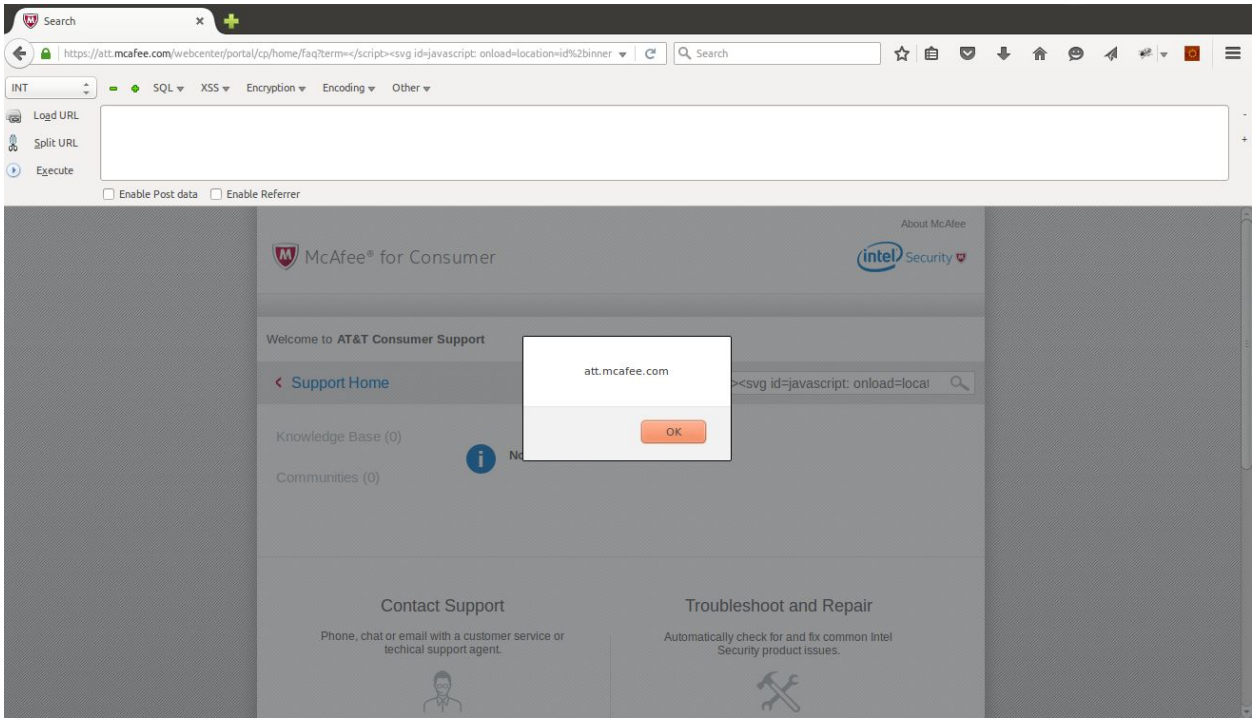
3.7 McAfee

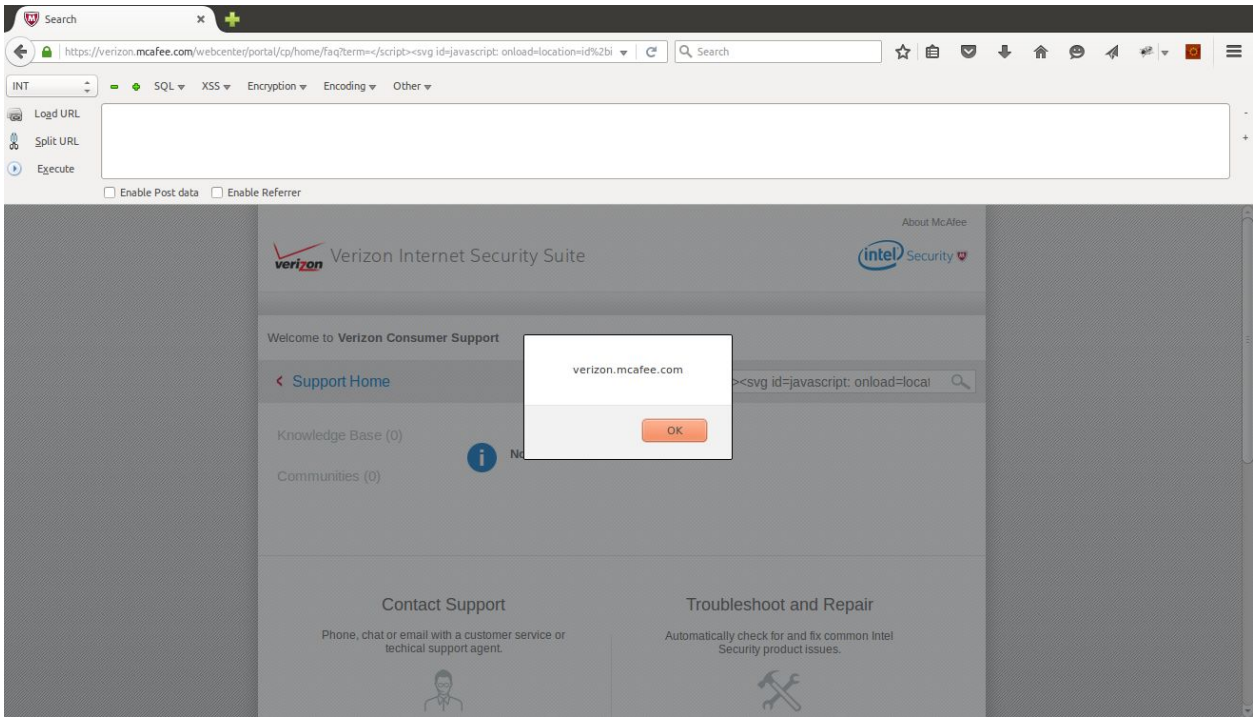
Affected Subdomains: service.mcafee.com att.mcafee.com verizon.mcafee.com truekey.mcafee.com

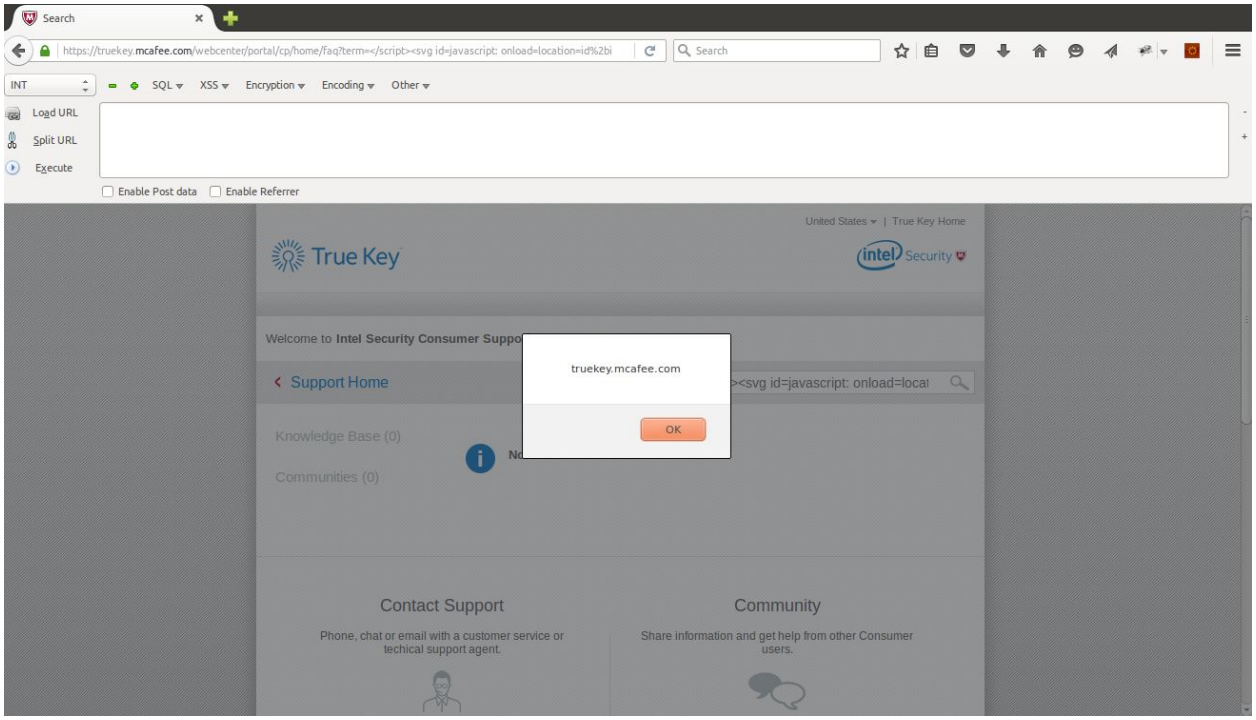
The above 4 subdomains were affected by the same XSS vulnerability as they used the same code. The GET parameter “term” was the one affected, its value appeared 4 times in the source of the page, being 3 times correctly encoded and once with single quote stripped (because the parameter value was held between single quotes).

In this XSS a very unique and smart payload was used that will be discussed right after the screenshots.




Search
https://att.mcafee.com/webcenter/portal/cp/home/faq?term=</script><svg id=javascript: onload=location=id%2binner
Search
INT
SQL
XSS
Encryption
Encoding
Other
Load URL
Split URL
Execute
Enable Post data
Enable Referrer
About McAfee
McAfee® for Consumer
intel Security
Welcome to AT&T Consumer Support
att.mcafee.com
OK
Support Home
><svg id=javascript: onload=local
Knowledge Base (0)
Communities (0)
Contact Support
Phone, chat or email with a customer service or techical support agent.
Troubleshoot and Repair
Automatically check for and fix common Intel Security product issues.


Search
https://verizon.mcafee.com/webcenter/portal/cp/home/faq?term=</script><svg id=javascript: onload=location=id%2bi
Search
INT
SQL
XSS
Encryption
Encoding
Other
Load URL
Split URL
Execute
Enable Post data
Enable Referrer
About McAfee
verizon
Verizon Internet Security Suite
intel Security
Welcome to Verizon Consumer Support
verizon.mcafee.com
OK
Support Home
><svg id=javascript: onload=local
Knowledge Base (0)
Communities (0)
Contact Support
Phone, chat or email with a customer service or techical support agent.
Troubleshoot and Repair
Automatically check for and fix common Intel Security product issues.



Rating
Severity: ⅖
Difficulty: 5/5

This XSS scored a ⅖ on our severity scale because the cookies were very well protected so the attacker doesn't actually have much options to attack the user's session. On the other hand, this XSS was pretty difficult to exploit due to a number of hardships faced while trying to trigger alert(document.domain).

The first problem we faced here was that a single quote was removed from the payload which denied us from using simple, short and universal payloads such as '-alert(document.domain)-'.

This leaves us with the other option which is breaking out of the script context and injecting our own HTML tag with its event handler and this way we have solved our first challenge.

The second problem was that even though we can now inject our own HTML tag, we can't use alert(document.domain) because the left parenthesis "(" was blocked. We could use grave accents (`) to trigger an alert but grave accents print what's written inside them as a literal string so "document.domain" will be shown as "document.domain" and not "subdomain.mcafee.com". *

We tried (in place of left parenthesis but the page still throws 500 (Internal Server Error) thus even HTML encoding couldn't save the day this time. We decided to use the hash sign (#) which allows us to add additional text that is not parsed by the server-side code at all, only by the client-side scripts via the location.hash document property.

Now comes the time of location based payloads, which use the document.location properties to change the URL to something that starts with the "javascript:" pseudo protocol which executes any JS code that is inserted beyond the colon.

Unfortunately we were not able to use use something like </script><svg/onload=location="javascript:alert"+location.hash>#(document.domain) because location.hash will not return only "(document.domain)" but "#(document.domain)". We couldn't use the substr() or slice() functions because they still use parenthesis.

To circumvent this, we used the innerHTML property of HTML elements which returns the text contained between the opening and closing tags of the current element. It made possible to create a payload like the following:

</script><svg/onload=location=”javascript:”+innerHTML+location.hash>”#”-alert(document.domain).

The payload above changes the document.location of the page to javascript:”#”-alert(document.domain) which by concatenating the # sign as a string (between “”) to the alert() function triggers the alert box.

Final Payloads:
https://service.mcafee.com/webcenter/portal/cp/home/faq?term=</script><svg/id=javascript:+onload=location=id%2binnerHTML%2blocation.hash>"&mode=search#"-alert(document.domain)

https://att.mcafee.com/webcenter/portal/cp/home/faq?term=</script><svg/id=javascript:+onload=location=id%2binnerHTML%2blocation.hash>"&mode=search#"-alert(document.domain)

https://verizon.mcafee.com/webcenter/portal/cp/home/faq?term=</script><svg/id=javascript:+onload=location=id%2binnerHTML%2blocation.hash>"&mode=search#"-alert(document.domain)

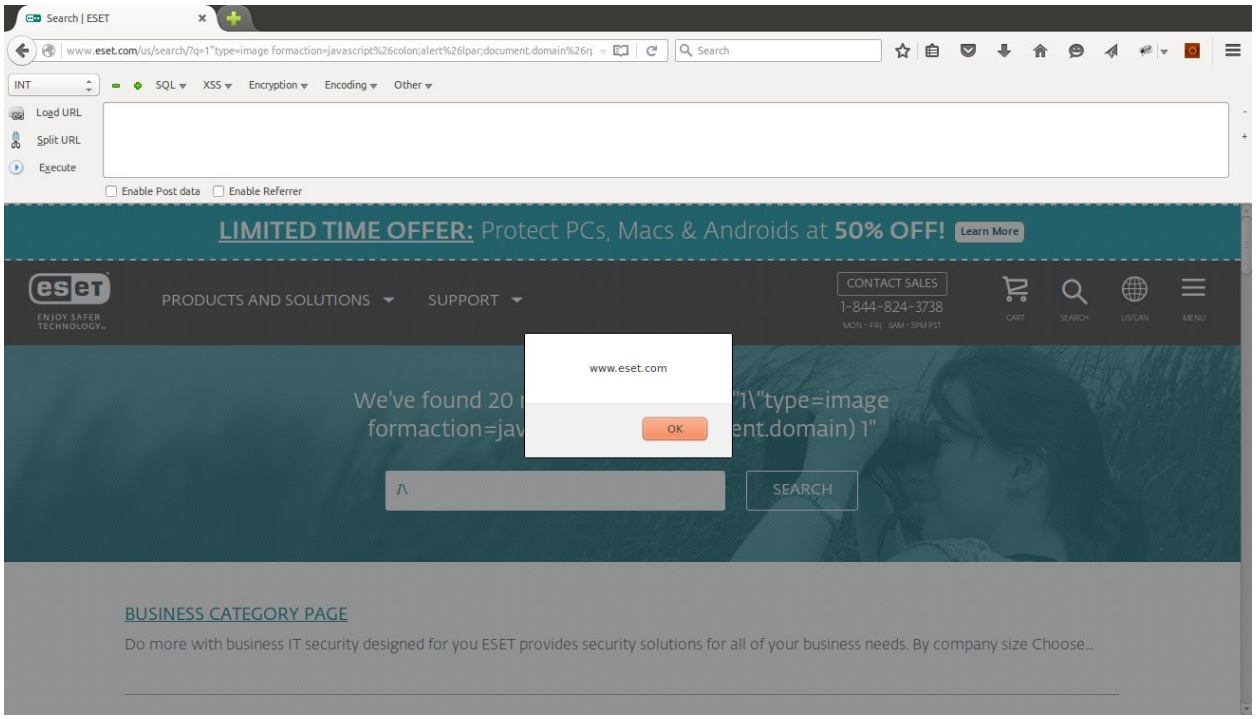
https://truekey.mcafee.com/webcenter/portal/cp/home/faq?term=</script><svg/id=javascript:+onload=location=id%2binnerHTML%2blocation.hash>"&mode=search#"-alert(document.domain)

* We later learned this.

3.8 ESET

Affected Subdomains: www.eset.com

The above subdomain was vulnerable to XSS via GET parameter “q” which is the search engine’s way to introduce search terms to the back-end code. The CMS software used on the website, vulnerable to the XSS, was not developed by ESET but by a company called “TYPO3”.



Rating
Severity: ⅕
Difficulty: 5/5

This XSS scored a ⅕ on the severity scale as the cookies were protected just like in McAfee's case but requires user interaction to trigger.

It scored a 5/5 on the difficulty scale because we spent a long time researching all the other subdomains first for XSS then starting to fingerprint this subdomain for any clues on what software it uses and whether they're vulnerable to XSS or not.

We discovered that the subdomain uses TYPO3's CMS (as stated earlier) so we started looking for any clues on how the CMS's filter works. After some time we discovered a page on GitHub that contains the CMS's unit tests and understood how it works exactly.

The CMS uses a blacklist to make event handlers fail so an event handler like "onclick" is turned to "on<x>click". It also turned the "javascript:" pseudo protocol into "ja<x>vascript\:".Additionaly, ">" and "<" are HTML encoded. We tried using "javascript:" instead of "javascript:" and it worked. We could also bypass the restriction of using "(" and ")" by using (and) instead.

The input reflects only twice in the page source, once between tags and the other in the "value" attribute of an <input> HTML tag. With the HTML encoding of the angle brackets by the application, we were only left with the option of trying to inject event handlers or attributes in the <input> tag.

Luckily, HTML5 introduced a new attribute that can be used with the <input> tags called "formaction". It allows us to set the "action"

attribute’s value of the <form> that the <input> is part of, and it even overrides the value of the “action” parameters if it’s already set.

Using the mentioned attribute in conjunction with the “type” attribute, we could now turn the <input> tag into a button that when clicked, sets the location of the document to “javascript:alert(document.domain)”.

Final Payload:
https://www.eset.com/us/search/?q=1"type=image formaction=javascript%26colon;%26lpar;document.domain%26rpar;+1

4. Vendor Responses

In this section we rate the responses of the AV vendors about the bugs in their web applications. The list is sorted in the same way it was sorted in the last chapter.

All vendors were sent an initial message on the 27th of January 2016 and we gave each of them at least 90 days to acknowledge and fix the bugs before release this paper.

4.1 BitDefender

BitDefender were sent the initial message, a message on the 31st of January and a message on the 1st of March. They only replied after the third message asking for more information on the issue,

when we actually discovered that it was already fixed by them without any reply to our earlier messages.

They then replied on the 21st of April stating that the issue was a duplicate and was fixed after we initially reported it thus not eligible for a bounty.

Rating
Speed: ⅕
Interest: ⅕

4.2 Kaspersky

Kaspersky replied to our initial message 27 minutes after it was sent, being the fastest one to reply.

The bug was acknowledged on the 4th of February, we checked on the 22nd of February and the bug was already fixed (yet we have no idea if it was fixed earlier).

A message was sent to Kaspersky again on that day and they said that they are waiting for a message from their security team. No messages were sent to us again until the release of this paper.

Rating
Speed: 5/5
Interest: ⅘

4.3 Panda Security

Panda Security was sent another message after the initial one: it was on the 31st of January and they replied, acknowledged and fixed the bug on the 1st of February, being the most interested company on our scale.

Rating
Speed: ⅘
Interest: 5/5

4.4 Avira

Avira was sent a message after the initial one then contacted through their Twitter account, when they replied and supplied us with an email to contact and finally seemed to care about their user's security on the internet.

We sent a PoC of the issue on the 2nd of February to the contact they provided then we sent three messages on the 8th, 20th, and 27th of February with no reply to any of them until the publication of this paper.

We found out on the 21st of April that the bug was fixed (possibly earlier than that) without even showing any type of appreciation and not even replying to our messages.

Rating
Speed: ⅕
Interest: ⅕

4.5 AVG

AVG replied to our initial message two days later and the bug was fixed 10 days after their response.

They also awarded us a t-shirt and a certificate of appreciation.

Rating
Speed: ⅘
Interest: ⅘

4.6 Symantec

Symantec replied after our second message, which was sent on the 31st of January and we sent a PoC on the 2nd of February.

They didn’t acknowledge the vulnerability until they replied after three messages were sent on the 8th, 22nd and 27th of February, stating that they have contacted the vendor of the platform and are waiting for a fix.

The bug was not fixed until the release of this paper.

Rating
Speed: ⅕
Interest: ⅕

4.7 McAfee

We tried to contact McAfee through multiple ways: we tried their contact form, their support center and their Twitter account. They finally replied and a PoC was sent on the 4th of February.

The bug was still not fixed until the release of this paper.

Rating
Speed: ⅕
Interest: ⅕

4.8 ESET

ESET replied only 47 minutes after we sent the initial messages, making them the second fastest company to reply.

The bug was already fixed before the PoC was sent on the same day at 9:57pm but we sent a screenshot of the alert box (the same attached to this paper) and they acknowledged our finding.

ESET sent us a formal acknowledgment for our effort.

ESET ENJOY SAFER TECHNOLOGY™

Acknowledgement for reporting security vulnerability

ESET Security team would like to officially thank Mustafa Hasan and Brute Logic for reporting following vulnerabilities:

- XSS in eset.com on January 27, 2015

This information has helped us to improve security of our online services and has prevented malicious exploitation of this vulnerability.

Best regards,

Daniel Chromek
Chief Information Security Officer
ESET s.r.o.
Einsteinova 24
85101 Bratislava
Slovakia

Rating
Speed: 5/5
Interest: 5/5

4.9 Avast

Being the only one out of 9 AV vendors, Avast was not contacted by any means because we were not able to find a XSS. This doesn't mean that it has no such vulnerabilities but only that it wasn't easy for us finding them in the timeframe of this work (about 1 week). Maybe an indicative of the strength of their web security when compared to its competitors, we leave here our congratulations to their team.

#hack2learn