



Inquirix

Bypassing Credential Protection Mechanisms and Measures to Counter these Threats

AUTHOR: Abi Waddell
<http://www.inquirix.org>

VERSION 1.0

Jan 2020

This sets out some methods of how an attack can circumvent credential attack prevention measures and what can be done to prevent and detect such attacks.

This specifically looks at the bypass methods of defeating timing defences, defeating measures that detect multiple logins with the same username, defeating IP address blocking with the use of per-request IP rotation proxies and defeating password change requirements.

Defeating timing-defences

Many login attack defence mechanisms allow a set number of logins from the same IP address and or same username – usually between 10 and 20, before enforcing a wait time of a set number of minutes, after which the credential-guessing requests can resume. This wait time can frustrate attackers attempting to use brute-force login techniques.

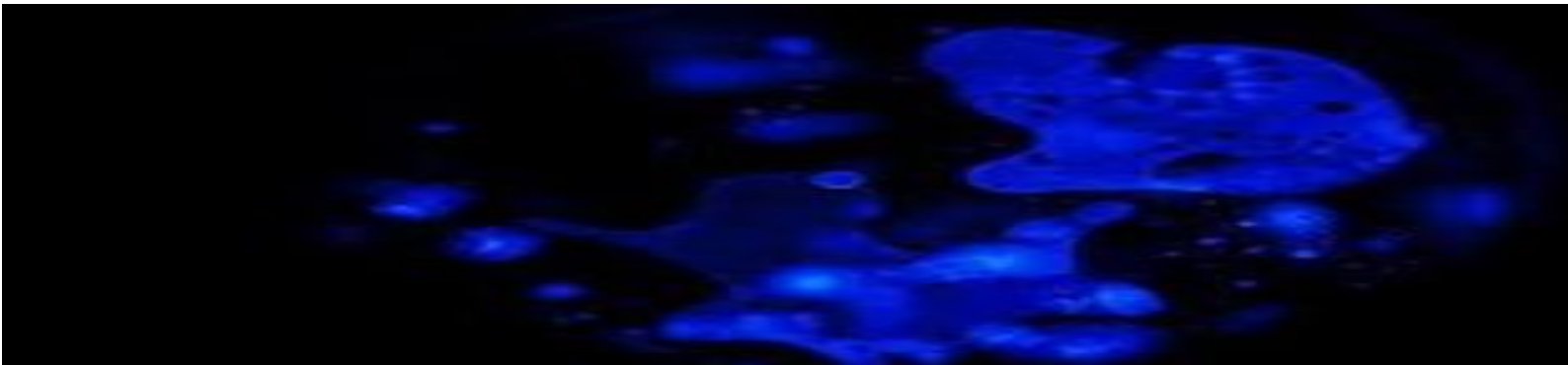
RPR 'Run-Pause-Resume'

The screenshot displays the Turbo Intruder interface. At the top, a table lists various payloads:

Row	Payload	Status	Words	Length	Time	Label
0	APPLSYS	200	10526	26474	0	
1	APPS	200	10526	26474	0	
2	BCAJ	200	10526	26474	0	
3	BIGO	200	10526	26474	0	
4	DCL	200	10526	26474	0	
5	DECMail	200	10526	26474	0	
6	DECNET	200	10526	26474	0	
7	DEFAULT	200	10526	26474	0	

Below the table, the interface shows a detailed view of an HTTP request and response. The request is for `PCST/login/login HTTP/1.1` and includes headers such as `Host`, `User-Agent`, `Accept`, `Accept-Language`, `Accept-Encoding`, `Content-Type`, `Content-Length`, `Origin`, `Connection`, and `Referer`. The response is `HTTP/1.1 200 OK` and includes headers such as `Server`, `Date`, `Content-Type`, `Content-Length`, `Connection`, `X-Powered-By`, `X-Frame-Options`, `X-Content-Type-Options`, `Last-Modified`, `Expires`, `Cache-Control`, `Vary`, `Accept-Ranges`, and `Alt-Svc`. The status bar at the bottom indicates `Reqs: 340 | Queued: 0 | Duration: 164 | RPS: 2 | Connections: 340 | Retries: 0 | Fails: 0 | Next: null`.

RPR is a credential-guessing enhancement to BurpSuite's Turbo-Intruder extension, which can implement pauses during attack runs. This enables the attacker to set a pause time after a specified number of test login requests in the middle of an attack run, in order to avoid having the target site block the attack.



The program allows BurpSuite Intruder to iterate through a list of passwords, but with the ability to set pauses in the middle of the attack after a specified number of requests have been made. After each pause, the attack then resumes automatically.

For example, the script can run the first 20 username/password requests using a specified list of credentials, pause the script for 10 minutes, then carry on with the script from where it left off for the next 20 requests, pause for 10 minutes, run the next 20 requests etc. until the whole list of credentials has been tried.

A free or paid version of BurpSuite with Turbo-Intruder installed is needed for this program. The program and full instructions can be found by contacting us (info@inquirix.org), but this gives a summary:

Download the script1_0.py from this repository. Right click on the login request to be tested (either within the proxy tab in Burp or within the Intruder tab) and select Turbo Intruder which should open another window showing the request to be tested. In the top half of the Turbo-Intruder window (within the 'Raw' tab) replace the password string by entering "%s" (without the quotes) e.g. password=%s. On the bottom half of the same window you will see the default Turbo script. In this section, copy & paste the content of script1_0.py, replacing everything in this part of the window. Within the first lines of the pasted script you will see "#Parameters to configure", where you can edit the following:

```
triedWords=the number of words to try before pausing. Example usage: triedWords=20
timeMins=number of minutes to pause for. Example usage: triedMins=0
timeSecs=number of seconds to pause for. Example usage: triedSecs=5
throttleMillisecs=number of milliseconds to wait before each request. Example usage:
throttleMillisecs=200
```

The script reads the password list from a .txt file which must be named 'words.txt' and copied to the main BurpSuite directory (where the Burp exe file is located). Click on 'Attack' at the bottom of the Turbo window to execute the script.

Note this has only been tested on the 'sniper' attack type within Burp Intruder. Some tweaking of the parameters described above may be needed to ensure this script works against the target to be tested.

Defeating measures that detect multiple logins with the same username

Many websites implement network defences against repeated login requests to prevent misuse and this often comes into play after the user has tried a set number of login attempts (e.g. 10 or 20 etc). The user, after a set period of not making any further requests (usually a few minutes to one hour), is often then permitted by the application to resume making credential-guessing requests.

Pinwheel

The screenshot shows the Turbo Intruder interface. The top section is a table with 7 columns: Row, Payload, Status, Words, Length, Time, and Label. It lists 27 payloads, each with a status of 200, 382 words, and a length of 1289. The bottom section shows the raw HTTP request and response. The request is a POST to /auth.cgi with various headers. The response is an HTTP/1.0 200 OK with headers including Server, Expires, Cache-Control, Content-type, and X-Content-Type-Options. The status bar at the bottom indicates 254 requests, 0 queued, 151 duration, 2 RPS, 254 connections, 0 retries, 0 fails, and 0 next.

Row	Payload	Status	Words	Length	Time	Label
40	admin/777777	200	382	1289	0	
41	guest/password	200	382	1270	0	
42	administrator/instruct	200	382	1270	0	
43	testuser/nicole	200	382	1270	0	
44	root/111111	200	382	1270	0	
45	admin/666666	200	382	1270	0	
46	guest/password1	200	382	1270	0	
47	administrator/functional	200	382	1270	0	
48	testuser/chelsea	200	382	1270	0	
49	root/iloveyou	200	382	1270	0	
50	admin/654321	200	382	1270	0	
51	guest/princess	200	382	1270	0	
52	administrator/dolls	200	382	1270	0	
53	testuser/biteme	200	382	1270	0	
54	root/12345	200	382	1270	0	
55	admin/555555	200	382	1270	0	
56	guest/qwerty	200	382	1270	0	
57	administrator/strap	200	382	1270	0	
58	testuser/matthew	200	382	1270	0	
59	root/12345678	200	382	1270	0	
60	admin/333333	200	382	1270	0	
61	guest/qwerty123	200	382	1270	0	
62	administrator/weather	200	382	1270	0	
63	testuser/access	200	382	1270	0	
64	root/1234567	200	382	1270	0	
65	admin/123456	200	382	1270	0	
66	guest/qwertyuiop	200	382	1270	0	

```
POST /auth.cgi HTTP/1.1
Host: [REDACTED]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0)
Gecko/20100101 Firefox/73.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate

HTTP/1.0 200 OK
Server: [REDACTED]
Expires: -1
Cache-Control: no-cache
Content-type: text/html;charset=UTF-8
X-Content-Type-Options: nosniff

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

Reqs: 254 | Queued: 0 | Duration: 151 | RPS: 2 | Connections: 254 | Retries: 0 | Fails: 0 | Next: null | Completed

Pinwheel is a credential-guessing enhancement to BurpSuite/Turbo-Intruder which allows multiple username/password attempts.

This script allows the tester to try and circumvent any application server login attempt protection which may be more likely to happen with repeated login attempts of the same username. This script allows several username and counterpart passwords lists to be tried in turn and so in this respect is useful if there are login credentials for several users on the same site which need to be tested.

During the running of Burp Intruder, this script takes a username and password from multiple username and (counterpart) password lists and iterates through these lists one after the other until all the words on the lists are tried. In this example there are 4 different usernames contained in one list, and 4 separate lists of passwords - each specific username having its own counterpart password list:

Username 1 in users.txt has counterpart passwords in the file words1.txt
Username 2 in users.txt has counterpart passwords in the file words2.txt
Username 3 in users.txt has counterpart passwords in the file words3.txt



Username 4 in users.txt has counterpart passwords in the file words4.txt

This is what some example requests would look like:

- Request 1: tries the first username in the file users.txt against the first password in the file words1.txt
- Request 2: tries the second username in the file users.txt against the first password in the file words2.txt
- Request 3: tries the third username in the file users.txt against the first password in the file words3.txt
- Request 4: tries the fourth username in the file users.txt against the first password in the file words4.txt
- Request 5: tries the first username in the file users.txt against the second password in the file words1.txt
- Request 6: tries the second username in the file users.txt against the second password in the file words2.txt
- etc
- etc

The requests will continue until all the usernames in the users.txt file have been paired with all the passwords in their associated words(number).txt file.

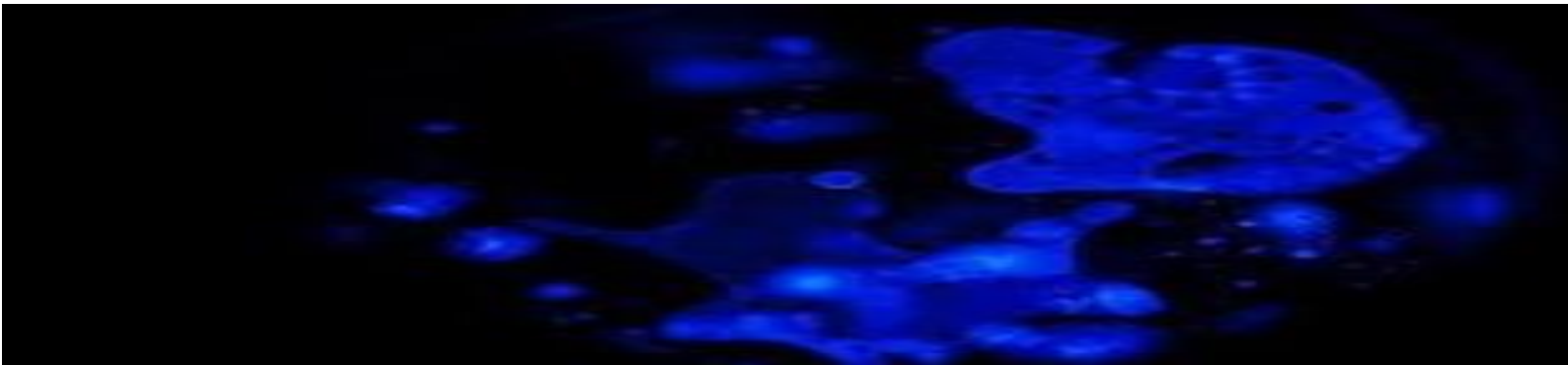
A free or paid version of BurpSuite with Turbo-Intruder installed is needed for this program. The program and full instructions can be found by contacting us (info@inquirix.org), but here is a summary:

Download the script2_0.py from this repo. Right click on the login request to be tested (either within the proxy tab in Burp or within the Intruder tab) and select Turbo Intruder which should open another window showing the request to be tested. In the top half of the Turbo-Intruder window (within the `Raw` tab) replace the password string by entering "%s" (without the quotes) e.g. password=%s. Also place a "%s" in the username string e.g. username=%s

On the bottom half of the same window, you will see the default Turbo script. Copy & paste the content of script2_0.py, replacing everything in this part of the window. Within the first lines of the pasted script you will see "#Parameters to configure", where the following can be edited:

```
concurrentConnections=number of concurrent connections. Example usage:  
concurrentConnections=5  
throttleMillisecs=number of milliseconds to wait before each request. Example usage:  
throttleMillisecs=200
```

Create a users.txt file in the main BurpSuite directory where the Burp.exe is located. In this users.txt file, list in a column all the usernames to be tested. For the number of usernames listed, create the same number of password files in the format words(number).txt.



For example, if there are three usernames to test, there will also be words1.txt, words2.txt and words3.txt files - all saved to the same Burp directory where the burp.exe file is located. Each words.txt file should list the passwords to be tested in a column.

For every username listed there needs to be the same number of words.txt files, so if there are 20 users in the username list, there should be 20 words.txt files containing the passwords. In this respect nothing in the script needs to be altered to reflect the number of users or password lists.

Click on 'Attack' at the bottom of the Turbo window to execute the script. Note this has only been tested on the 'sniper' attack type within Burp Intruder. Some tweaking of the parameters described above may be needed to ensure this script works against the target to be tested.

Defeating IP address blocking with the use of per-request IP rotation proxies

IP address rotation proxies allow the source device to funnel each request through a proxy gateway which in turn obtains an IP address from a large pool of IP addresses. There are paid-for services which offer the use of a different IP address after every few minutes or per request from an address pool of tens of thousands, and this is usually to facilitate web scraping, crawling and bulk account registration.

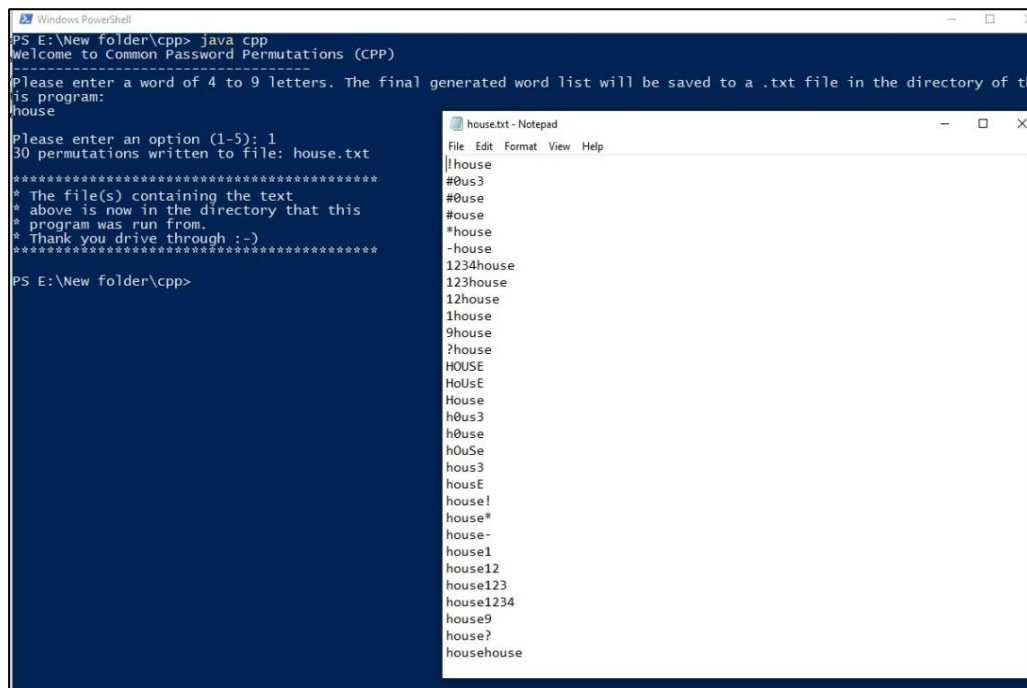
From a credential-attacker perspective, rotation proxies enable the circumvention of any network blocking due to a high number of application requests from the same IP address.

Defeating password change requirements

Users often employ a previously simple password they were once permitted to use when application restrictions were less strict in the past, and so follow some common methods of making such passwords more complex in order to pass stricter application checks when confronted with an enforced password change.

Typical ways that users add complexity include adding numbers, capitalising letters and adding specific special characters.

CPP



```
PS E:\New folder\cpp> java cpp
Welcome to Common Password Permutations (CPP)
-----
Please enter a word of 4 to 9 letters. The final generated word list will be saved to a .txt file in the directory of the
program:
house

Please enter an option (1-5): 1
30 permutations written to file: house.txt
*****
* The file(s) containing the text
* above is now in the directory that this
* program was run from.
* Thank you drive through :-))
*****
PS E:\New folder\cpp>
```

```
house.txt - Notepad
File Edit Format View Help
!house
#0us3
#0use
#ouse
*house
-house
1234house
123house
12house
1house
9house
?house
HOUSE
HoUsE
House
h0us3
h0use
h0uSe
hous3
houseE
house!
house*
house-
house1
house12
house123
house1234
house9
house?
househouse
```

CPP is a small script to produce common variants of a particular password when testing login credentials using tools such as BurpSuite Intruder or similar.

The list of permutations produced of a particular word is not meant to be exhaustive, but merely to show the most common iterations of a word that has been made more complex.

Analysis of huge numbers of existing passwords have shown that when a user is required to set a complex password, they sometimes use their existing (simple) password but change it using predictable methods. This gives them the best balance of meeting the application's complexity requirements whilst still being able to easily remember their password. For instance, if the original password was `cisco`, then it would be common for this to be changed to e.g. Cisco1 or c1sco or cisc0 or cisco99 to make it more complex.

Unlike other password-guessing tools, the lists produced - because they are using permutations which have a higher likelihood of success - are much shorter, which means a password-guessing test will be finished between a few seconds to a few hours depending on the word chosen and application to be tested. The lists can be ported to any tool and the running/configuration of this script is extremely quick and simple.

Once a word is entered in the program, the user can choose one of five options - option 1 being the simplest/shortest and option 5 being the most comprehensive/longest:

Option 1 will produce a very short list of the following permutations of a word:

- doubling the word e.g. testtest
- adding 1,2,3,4,9 separately and together before and after the word e.g. test9, 12test, 1234test
- substituting the letters for symbols e.g. `i' for `1' and `!', `a' for `@' and `h' for `#'.
- substituting `o' for `0', and `e' for `3'
- adding `?', `!', `-' and `*' before and after the word e.g. !test, test*
- capitalising various letters e.g. Test, TEST, TesT

Option 2 will generate a longer list of permutations which do all the variations of option 1), for instance, for the word `house' there would be: #0USE! , !HoUs3, -h0usE etc

Option 3 will generate a longer list which includes everything from options 1) and 2) but also adds the characters `.', `£', `\$' and `%` to the front and then at the end of each word.

Option 4 generates a list which includes everything in option 1 and 2, but with the following additions to the basic permutations as listed in option 1):

- Adds the numbers 1940 through to 2019 to the beginning and end of the basic word e.g. 1955test
- Adds the numbers 00 through to 99 at the beginning and end of the basic word e.g. test67

Option 5 produces the longest list incorporating everything from options 1, 2, 3 and 4.

All the options take only a few seconds to generate a list. The longest list (using a 9-character word) produced by option 5, will consist of around 20,000 - 36000 words. This is reduced to around 3500-14000 words if a 4-character word is used. The shortest list (using a 4-character word) produced by option 1, will only be 20-20 words in length.

The only requirement to running this script is ensuring the local machine can run Java. A password guessing tool such as a free or paid version of BurpSuite is recommended. Contact us for the program files. Save the cpp.class and cpp.java files to a folder on the local machine. Open a command line, go to the directory of these saved files or use a program that can run Java files and type:

java cpp

Then enter the word to be changed. It can consist of any characters, capital letters, symbols, numbers etc, but has to be between 4 and 9 characters (inclusive) in length. Once a word has been entered, you will be asked which option you want from 1 to 5. As mentioned above, option 1 will be the shortest, simplest list, presenting the most common permutations of the entered word. Option 2 will produce longer list as it has more permutations and so on until option 5 which has the longest list of permutations.

The final generated list of permutations will be saved as a .txt file in the same directory as the program files. This list can be copied & pasted into BurpSuite Intruder for password-guessing tests or used in other similar tools.



Improved credential attack mitigation methods

Here are some of the measures that can be taken to counter the above bypass threats:

Do not reuse similar passwords

People should not use previously leaked passwords in their new more complex password. If they used their eldest daughter's name in a previously leaked password, they should not use their second child's name for example. If they had used the word e.g. 'London01', they should not use 'London02'.

Implement better password complexity

The best 'complex' passwords will be those that:

- Use unusual special characters like '~' or '>' rather than '!' or '*'
- Use more than 10 characters in length, but only if word-doubling isn't permitted
- Have spaces between the characters
- Do not use numbers under 100 or dates within the previous 80 years
- Will not simply reverse or jumble the characters from an existing password. Complexity checking mechanisms in many applications allow the same password to be chosen but reversed or jumbled, for instance changing 'house1942' to '1942house'.

Keep login response timings the same

Take care to keep the response timings between the initial login request and captcha verification, and then captcha request to the final authentication response the same, as in some cases the two sets of timings, either separately or combined, may point to obvious difference in times depending on whether the credentials are valid. Attackers can exploit this difference to enumerate valid usernames.

Flag standard password list compilations

Often brute-force attacks which employ pre-compiled lists of passwords, for instance the 'top 1000 default passwords' or '500 most common passwords', will be detectable after the first few words are entered in the target systems. Rather than waiting until 100 or 1000 repeated rapid requests are made before flagging it as suspicious, the first 3 passwords should immediately raise alarm bells and the connection blocked and logged. Attackers will not usually mix up the order of the words in these password lists and instead merely copy them into their attack tools as they are.

A quick search on Google will find quite a few commonly used lists and the network or system administrator merely has to take the first 3 words in each of these lists and in order, and ensure that the application or device ruleset flags when these words are used in quick succession in their original set order.

Enable username obfuscation and encrypting during login requests

If the user ID cannot easily be tampered with because the website encrypts and obfuscates it, this will go a long way to preventing login compromise.

Make login forms non-proxy aware

Some website login forms cannot easily be filtered through an interception proxy. Research is currently taking place to understand why this is the case and when more information is available on these reasons, then website administrators will be in a better position to help prevent request and response interception and tampering.

Blacklist or flag proxy service IP addresses

IP addresses belonging to well-known proxy services should be blacklisted. Proxy services are offered as either a free or paid service to facilitate web scraping amongst other activities. Many rotate addresses from a large pool of addresses and if these are known they can be blacklisted or flagged.

Flag or block sequential numbers being used in either the password or username fields

Any letter or number sequences being submitted in each request in either login or other user-interaction points in the application should be flagged as suspicious. A series of passwords to the same username following a predictable sequence could look like this for example: password1, password2, password3; or 1001, 1002, 1003; or apassword, bpassword, cpassword, and so on.

Monitor rapid requests to web pages containing a readable user ID

Any web page containing an HTTP response that contains a readable User ID – if it cannot be changed to something else, or encrypted, should be monitored for any rapid number of requests and responses made in a short space of time.

Using publicly leaked passwords for an account

Lists of publicly leaked accounts for an organisation can be fed into any network and application screening process. If two or more of such accounts are attempted in a short space of time, then this should be flagged as suspicious.

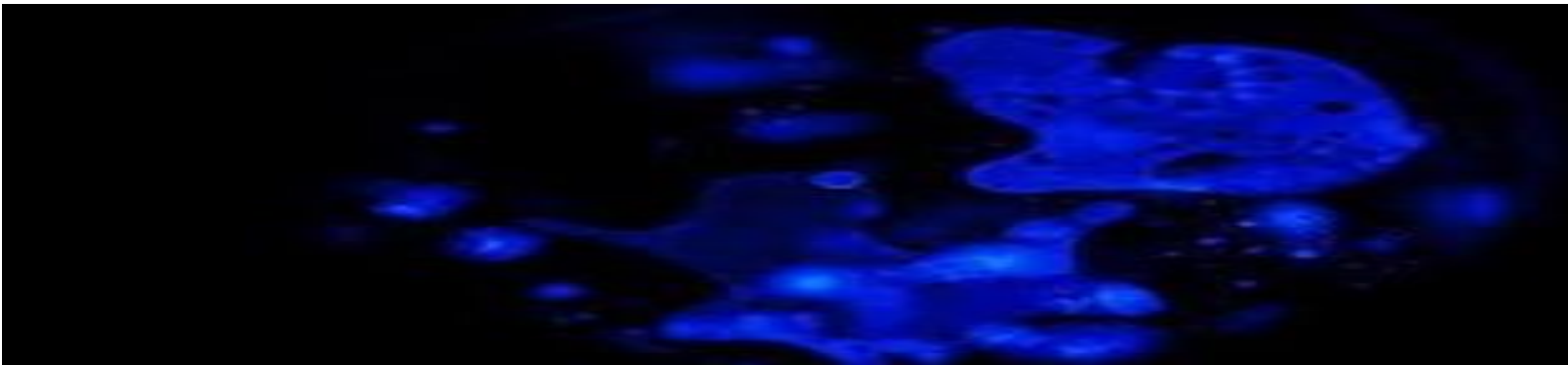
High frequency of logins using different usernames

A high number of login attempts in a short space of time and using different usernames should be flagged.

Flag the use of deliberately nonsense words in forms

Create rulesets to detect the use of words which are deliberately nonsense as this may be indicative of the user testing the site responses, both in login forms and any other part of the website which allows user interaction. In credential enumeration particularly, the attacker will compare responses or try to elicit error messages with deliberately wrong names - i.e. with the use of a gibberish password, with a potentially valid username.

Rulesets should look for words which contain two or more of letters which are not usually paired e.g. hh, aa, yy ; three or more of any letter which are together e.g. aaa, bbb, mmm; the presence of all numbers and or special characters in the username, home address and other similar fields, the presence of letters in the phone number field; and identical strings of characters used in more than one field.



Likewise, in login forms, non-existent domains after the '@' in the username, e.g. @yggu6843wflho6.com could point to the use of a deliberately wrong email address which would be used for enumeration purposes.

High frequency of forgot password requests using different usernames

A high number of forgot password requests using different usernames within a short time period could point to attack reconnaissance activity.

Timing of login attempts

Screen for sets of similar login timings where the user agent is the same. Automated brute forcing tools can change their speed for each login attempt, but they can still be identified as suspicious from the interval timing of the login requests – not whether there are hundreds made in a short space of time, but whether each click of the submit/enter button has almost the same interval of time between them. Here are some examples:

Traditional alarm set on the number of requests in a specified time period e.g.:

- 100 requests or more per minute

Improved method of reporting on the similarity of interval between each login request:

Example of timings showing that an automated tool is being used:

- Time between login request 1 and login request 2 = 5 seconds
- Time between login request 2 and login request 3 = 5 seconds
- Time between login request 3 and login request 4 = 5 seconds
- Time between login request 4 and login request 5 = 5 seconds
- Time between login request 5 and login request 6 = 5 seconds

Example of timings showing that manual attempts are being used:

- Time between login request 1 and login request 2 = 4 seconds
- Time between login request 2 and login request 3 = 7 seconds
- Time between login request 3 and login request 4 = 5 seconds
- Time between login request 4 and login request 5 = 8 seconds
- Time between login request 5 and login request 6 = 6 seconds

Example of timings showing that genuine users are logging in:

- Time between login request 1 and login request 2 = 4 minutes
- Time between login request 2 and login request 3 = 20 minutes
- Time between login request 3 and login request 4 = 6 minutes
- Time between login request 4 and login request 5 = 30 minutes
- Time between login request 5 and login request 6 = 12 minutes

Increase the length of time after when repeated login requests are blocked

Traditionally, repeated credential attempts are blocking after several wrong tries within a short space of time. Some credential guessing attacks have large time intervals between each attempt, or there may be a lot of attempts in a short space of time followed by a long time with nothing tried, then a resumption of repeated attempts. Such attempts should instead be monitored at least over a few hours. Here is a possible attack pattern which might circumvent the traditional method of blocking:

- Login attempt 1 with username `joebloggs`
- Login attempt 2 with username `joebloggs`
- Login attempt 3 with username `joebloggs`
- Login attempt 4 with username `joebloggs`
- Login attempt 5 with username `joebloggs`

Interval of 75 minutes

- Login attempt 6 with username `joebloggs`
- Login attempt 7 with username `joebloggs`
- Login attempt 8 with username `joebloggs`
- Login attempt 9 with username `joebloggs`
- Login attempt 10 with username `joebloggs`

Likewise, invalid login attempts which are not made concurrently should be screened. If for example the system is set to lockout an account after six wrong concurrent attempts, then this could be circumvented by attempting five login guesses, waiting for an hour, trying five more guesses, waiting for an hour, and so on.

Flag same valid with different invalid usernames

The use of a valid username (with invalid password), followed by a login with a deliberately gibberish or invalid username, points to suspicious activity. Attackers may test the application's response to a valid versus invalid username to see if there are any error messages which may indicate whether a username is likely to be valid, for example:

Username: attacker@validemail.com password: deliberatelywrongpassword

System response: `incorrect password`

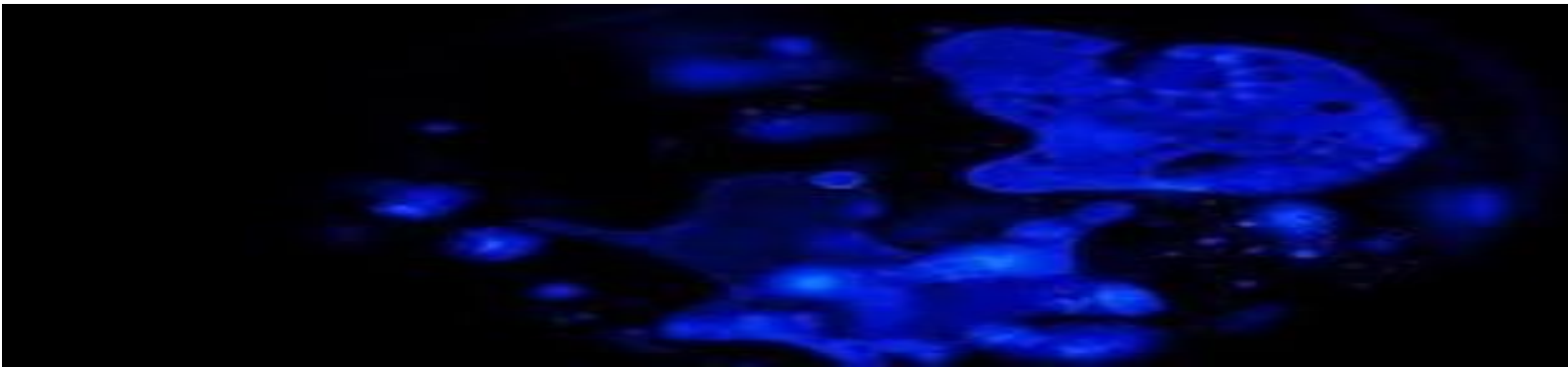
Username: attacker@deliberatelywrongemail.com password: deliberatelywrongpassword

System response: `there is no record of this user`

Username: victim@victimemail.com password: deliberatelywrongpassword

System response: `incorrect password`

The attacker now knows that the victim has an account on this application based on the error messages, and so, such request patterns should be flagged.



Implement a captcha and two-factor verification.

These are two very effective methods of frustrating a credential attack. Two-factor authentication codes should be at least six figures in length and enforce a time in minutes during when the verification should take place. This is to reduce the risk of authentication code guessing which may take place over several hours. A six-figure code is not unlikely to be cracked within a short timeframe of a few minutes.