# Global Information Assurance Certification Paper

# A Fuzzing Approach to Credentials Discovery using Burp Intruder

*GIAC (GPEN) Gold Certification*

Author: Karl Dawson, karldawson1@gmail.com
Advisor: Joey Niem

Accepted: 11th February 2009

## Abstract

*Password guessing against web-based applications typically relies on a pattern match of what a 'successful' login response looks like. It may also consider HTML status codes such as looking for a '200 OK' server response. Armed with this information, the tester is able to begin processing hundreds or thousands of server responses, looking for alerts when the anticipated pattern is detected. But what if you don't know what a 'successful' login response looks like for the application you're testing? Or the server always returns a '200 OK' response? Or similarly, what if the server responds with a failure for the access you were after, but provides information about another form of access you hadn't known existed? This may not be a 'successful' login, but may still provide you with a wealth of information; some of the failure information itself may also be of particular interest.*

*Instead of basing success on the assumption of what a successful login keyword may be, the approach described here is to treat the application credentials as just another input to be tested. As such, it relies on 'anomaly detection' to discover not just 'successful' logins, but potentially a host of other information that may otherwise go undiscovered. To help this process further, this approach also takes into consideration the relatively slow rate at which on-line credentials testing occurs, and shows how some knowledge of human behavior is an integral part of speeding things up.*

*The tool used to conduct this testing is Burp Intruder, part of the Burp Suite 'web application vulnerability scanner' package from PortSwigger (www.portswigger.net, 2009).*

Karl Dawson, karldawson1@gmail.com

# Table of Contents



Karl Dawson, karldawson1@gmail.com

# 1. Introduction

Usernames and passwords have been an Achilles heel of computing systems for almost as long as we've had computers (Kessler, 1996). They're used in almost all aspects of daily life, including banking, shopping, recreation and work. Entire industries have arisen with a range of products and services designed to exploit them, manage them, support them, enhance them, and develop replacements to them.

Their simplicity is a strength and a weakness. They are easy to implement, universally understood, and are 'low tech', in so much as no special hardware or software is required for their use. But they are also subject to the limitations of human memory and behavior, and while passwords are capable of considerable complexity, they rarely reach such heights.

A variety of solutions have been developed to compensate for these weaknesses, including hardware and software tokens and biometric recognition systems. These solutions aim to provide an environment with multi factor authentication capabilities, but their uptake has been relatively slow (Braue, 2005). This particularly affects the consumer sector where the distribution, support and use of multiple authentication mechanisms can be problematic (Felker, 2007).

Despite this, cracking passwords on web-based authentication systems may still be unsuccessful if there isn't a good understanding of the target system. The structure of usernames and passwords, the presence of account lockout features, the type of backend authentication system, and the speed of the target system can all impact on the end result. Also, a username may not have privileges on the target system, but could still give access to other systems that are as yet unknown to the tester.

To increase the chance of finding suitable credentials, a broader approach than just brute force and dictionary attacks needs to be adopted (OWASP, 2009). Such an approach should aim to discover as much as possible about the structure and types of usernames in use, the sorts of access they provide, and the possible authentication systems that support them. It should assess the presence of account lockout functionality to ensure that all possible accounts are in fact tested. Just as important, it should discover

as much as possible about the structure of the passwords. Not just whether complexity is used, but other factors relating to the people who created them.

The following sections start with an overview of the methodology used to guide the testing process, as well as a general overview of the components of Burp that are used to conduct the tests themselves. This is followed by an analysis of usernames; a step that is often overlooked in the rush to crack a password. It is however a vital step in dealing with the comparatively slow authentication mechanisms of on-line systems. Passwords are then covered and, as with usernames, this includes an assessment of their structure, and how knowledge of human behavior can be used to maximize the chances of guessing the right ones. A brief real-world example will then be presented, showing the process end to end.

Burp Intruder, part of the broader Burp Suite of tools (Portswigger, 2009), will be used to conduct all testing. Burp Intruder is an HTML fuzzer (OWASP, 2009) well suited to conducting tests such as this, as well as the more traditional brute force and dictionary attacks. Discussion and examples will be provided throughout each section showing which features of Burp Intruder may be of most benefit.

Since Burp is primarily a web-based assessment tool, the scope of this document is limited to the testing of credentials against on-line systems. This typically includes web applications and portals, but may also encompass web-enabled infrastructure devices such as routers and switches, as well as Remote Terminal Units (RTUs) used in the process control and SCADA arenas. The username and password discovery processes though, should be applicable to other systems where credentials are required, such as telnet, SSH, or FTP.

## 2. Methodology

The testing methodology described here fits within the 'Testing For User Enumeration (OWASP-AT-002)' section of the OWASP Testing Guide (OWASP, 2009), and is designed to be used on production systems as part of a formal penetration testing exercise. To this end, a core aim of the processes described is to have a minimal impact on the system being tested. This doesn't mean that the process will be stealthy, but rather

that it's designed to minimize the likelihood of there being any impact on those systems. Credentials attacks do however present the possibility of causing account lockouts which result in a denial of service for the organization under test. This should be avoided unless it's been defined in an agreed Rules of Engagement.

The very nature of this methodology involves the continual analysis and tuning of the responses being returned. It's therefore ideal for the early detection of performance related problems or similar conditions that may impact on the target system.

The testing methodology is also designed for use against on-line systems, where processing and resource limitations reduce the total number of attempts per second to relatively low numbers when compared to off-line testing. As such, there's a heavy focus on research and analysis to ensure that the selected usernames and passwords have the highest likelihood of success.

In the spirit of fuzzing though, this methodology is also looking for the strange and unusual. It's trying to look outside the box for the systems and access that weren't expected, and for the programming errors that allow them to be exploited.

## 3. Getting Burp Up and Running

Burp supports a wide range of parameter fuzzing across all aspects of an HTML request, and provides an equally wide array of canned or user-supplied input to be sent to those parameters. This section provides a quick overview of the groundwork that's needed to get the application to a point where attacks can be conducted, but which aren't often changed from that point onwards during the course of an attack.

This groundwork starts with the initial capture of a request with Burp Proxy, progresses through the passing of the request over to Burp Intruder, and on to the configuration of its various attack types. It finishes with the selection of the parameters within the request that need fuzzing. Once this point is reached a range of different attacks can then be carried out and repeated as required.

An overall process flow is also provided by way of Figure 1, which demonstrates a typical end-to-end usage of Burp. This starts with the interception by Burp Proxy of an

HTML POST containing the credentials to be manipulated. The 'Send to Intruder' action has then been selected so that the request can be forwarded to Burp Intruder. Once a request has been forwarded to another part of the Burp tool suite, Burp Proxy is free to continue intercepting, forwarding or dropping subsequent requests, as well as sending them to other tools such as Burp Repeater or Burp Sequencer. A single request such as the one shown can also be sent to multiple applications within the Burp tool suite so that it can continue to be manipulated in other ways.

Once the request is in Burp Intruder, a variety of options and payload settings can be configured, before launching the attack and reviewing the results.

Each portion of this end-to-end process flow will be described in more detail in the following sections.

The specifics of how to use Burp Proxy's features is out of scope and won't be covered. However, its use should be familiar to anyone who's used similar proxy applications such as Paros, Odysseus, Charles or Webscarab. The OWASP site provides links to all of these tools and more.

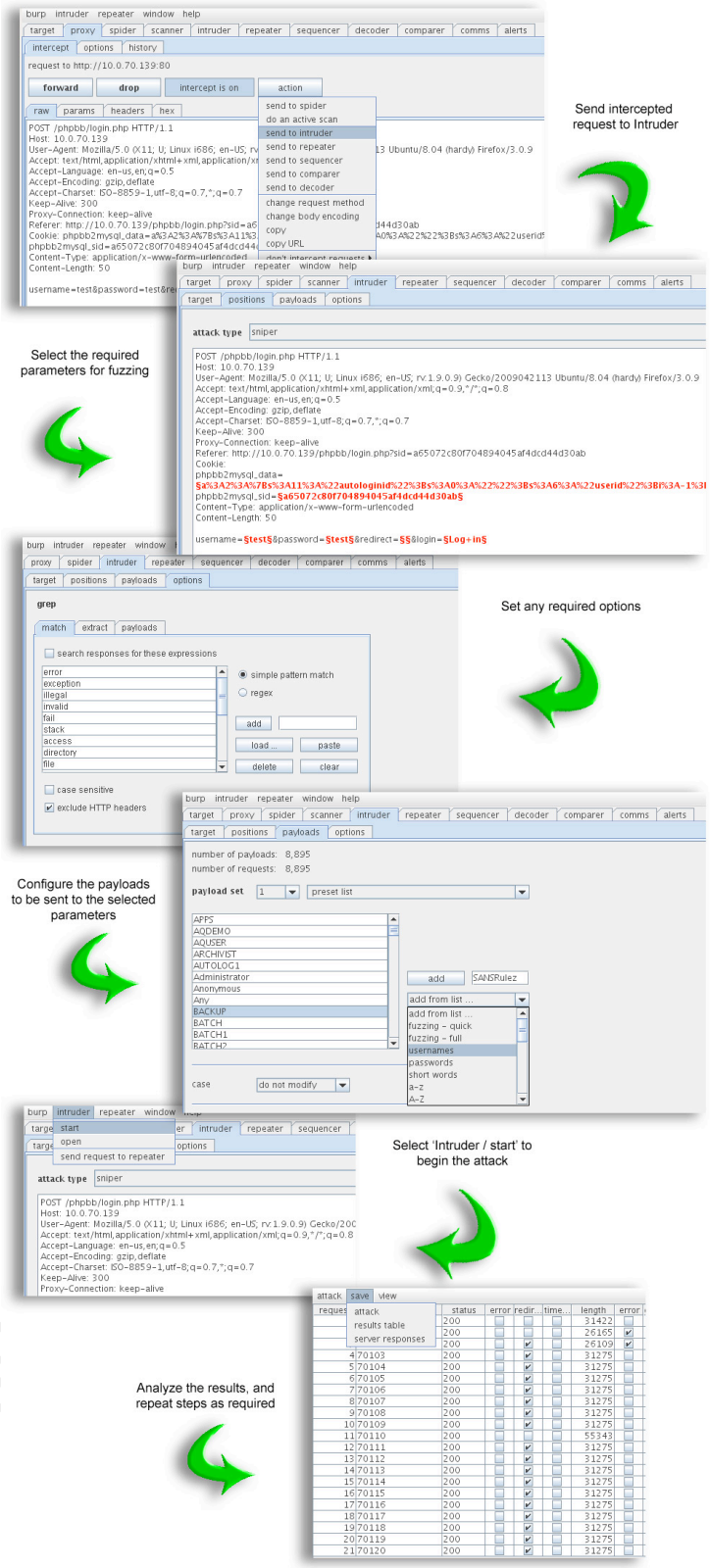


*Figure 1 - Burp Intruder process overview*

Karl Dawson, karldawson1@gmail.com

As mentioned above, Burp Intruder uses attack types to define how payloads are delivered to the parameters needing to be fuzzed, and are selectable from the Positions tab, as shown in Figure 2. For the purpose of discovering credentials, the sniper attack type will be used in the majority of examples.

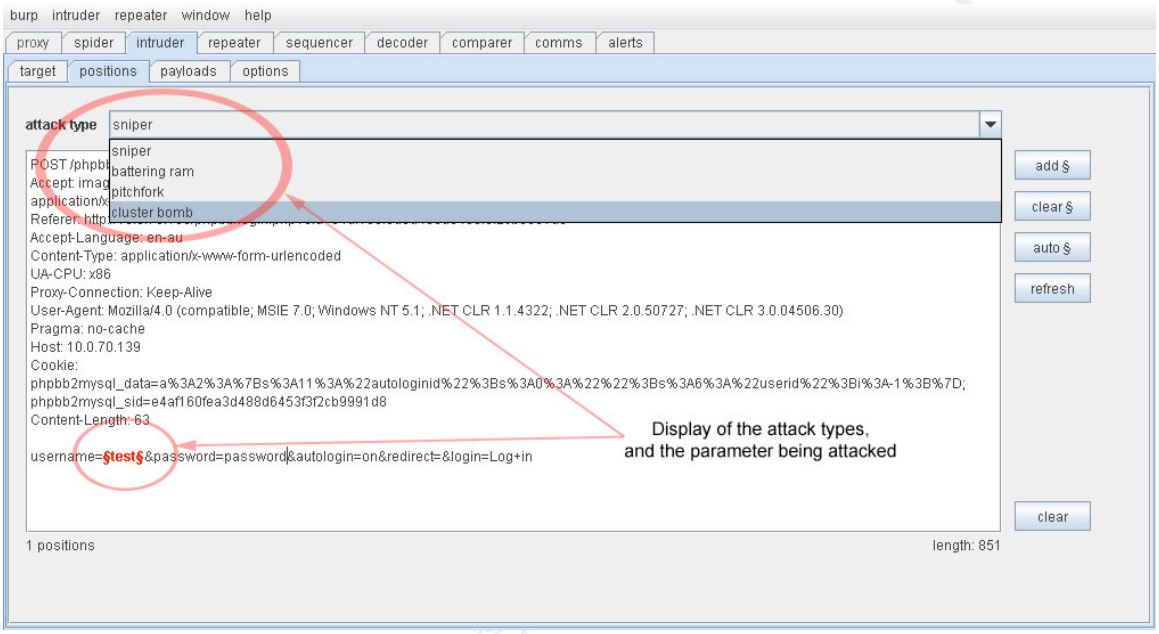


*Figure 2 - Burp Intruder attack types*

It's worth understanding how each of them work though, and how they could be applied to specific testing scenarios. The attack types are as follows (Portswigger, 2009):

- **Sniper** – sends a single payload to each of the selected parameters; i.e. each parameter is sequentially tested with the same set of variables
- **Battering ram** – sends a single payload to all of the selected parameters at once; i.e. all parameters will be passed the first variable, followed by all parameters being passed the second variable, and so on until the payload is completed
- **Pitchfork** – sends a specific payload to each of the selected parameters; i.e. all parameters need to be passed its own payload, and the variables of each payload are passed to its designated parameter in sequence
- **Cluster bomb** – starts with a specific payload to each parameter, and when all variables have been tested, will start testing with the payload from the next variable, such that all parameters get tested with all variables

By default, Burp Intruder will attempt to discover all parameters suitable for fuzzing within the request and mark them with the '§' symbol. At this stage the process is only concerned with the discovery and manipulation of usernames and passwords, so all of the automatically highlighted parameters can be cleared by selecting them, and clicking the 'clear §' button, as shown in Figure 3. The required parameters can then be selected by using the 'add §' button (or by not deselecting it in the first step).

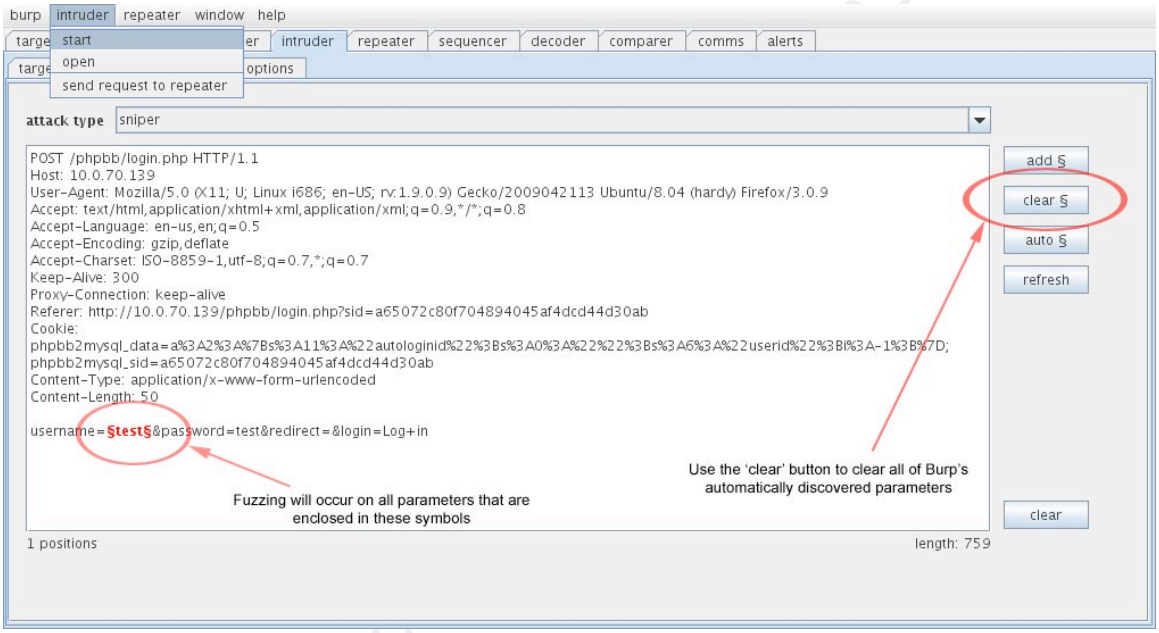


*Figure 3 - Request in Intruder ready for manipulation*

At this point a request has been loaded and portions of it have been selected for fuzzing. Burp Intruder is now up and running, and ready to start sending manipulated data.

The following sections will deal with the processes associated with selecting usernames and passwords to send to the manipulated parameters, as well as how those usernames and passwords can most effectively be delivered to those parameters.

## 4. The Human Element

Before moving into the technical aspects, it's worth touching on the significant role that human aspects can contribute to successful credential exploitation. While society continues to use computer systems, and continue to need humans to access those

computer systems, there will continue to be weak passwords. The methodology discussed here relies on it and, based on the results of recently publicized attacks, will continue to be applicable for some time to come.

As an example, a phishing attack conducted against MySpace users in 2006 demonstrates this, providing a useful insight into the structure of passwords associated with more than 34,000 accounts. A study of this attack (Schneier, 2006) shows that the top twenty passwords included password, password1, abc123, 123abc, qwerty1, and 123456. It also showed that numbers were used in well over half the passwords (most often at the end of the password), and that 65% of the passwords contained eight characters or less.

This example shows that despite the trillions of combinations available with an eight character alphanumeric password, only a small proportion of them is actually used. Of that small proportion, predictable patterns such as dictionary words still play a notable part.

This human element is being stressed at this point because it will be a recurring theme, and an essential tool, throughout the remainder of the sections.

## 5. Username Discovery

Password cracking against an online system can be slow. Where an offline crack using tools such as Hydra or John the Ripper may provide hundreds of thousands of tries per second or more, a similar online crack may be in the order of tens of tries per second or less. This can be due to a number of factors including available network bandwidth, the speed or utilization of the target server, and the need to accommodate password lockouts (TechRepublic, 2008).

With so few attempts per second occurring, knowing the correct format of usernames, and preferably the actual usernames themselves, can significantly reduce the time taken to gain a successful result.

A reverse brute force attack (webappsec.org, 2005) can be used as part of this discovery process. Where a typical brute force attack tries many passwords against each

username, a reverse attack tries a single password against a large username list. This list can be compiled from a range of sources including corporate phone directories, e-mail distribution lists, Google searches, social engineering, dumpster diving or usernames found on other compromised systems.

Using a reverse brute force attack will also ensure that accounts aren't locked out, while still allowing the tester to check for any unusual responses that may be returned. The reverse brute force approach is covered in more detail in Section 6, Assessing Account Lockout.

While this section is concerned with the discovery of usernames, those usernames still need to be paired with a password. That password should be chosen for maximum effect, and so should be one that is suspected of being associated with the site or organization under test. A common password like 'password' could be used, but only if it's first been determined that the passwords for that site don't require complexity such as the inclusion of upper case or numbers. Using the MySpace example, only 0.22% of the 34,000 compromised accounts were 'password1', but that still gives access to almost 75 accounts. Finding usernames is good, but gaining some form of access or unusual response with a valid username and password combination on the first attempt is even better. Putting together a suitable password for use during this stage of the process is discussed further in Section 9, Password Discovery.

The timing of the username discovery phase, as well as any subsequent password testing, can also assist in gaining access to the target system by exploiting human behavior and common corporate practices (Hitachi ID Systems, 2009). For example, the likelihood of passwords being changed on a Monday, combined with a process that resets passwords to a standard default value, may increase the likelihood of a successful compromise.

## 5.1. Username Structure

Knowing the required structure of a username allows an existing list of names to be added to by including similar or related lists of names that have been formatted in the same way. This might be the first name followed by the last name, the first initial

followed by the last name, or either of those but in e-mail address format. A quick way of finding the required structure can be to have a look at the web page source code. Some sites, such as the one shown in Figure 4, specifically test to make sure that the username is only submitted as an e-mail address.

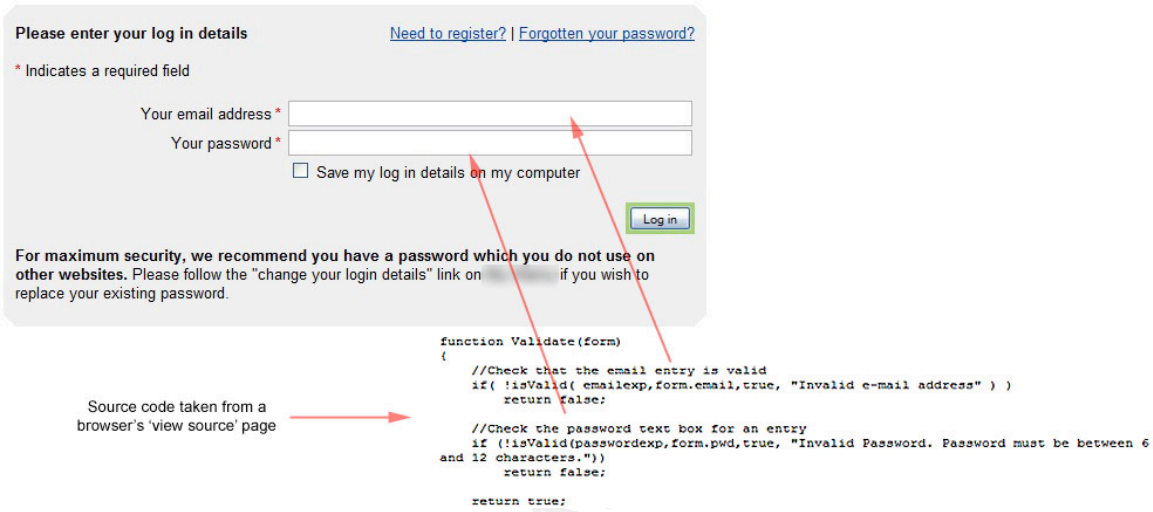


*Figure 4 – Credentials source code*

Based on this, the system can be tested to see if it returns different results for a correctly formatted valid, as opposed to an invalid, username (OWASP, 2009). The screen captures in Figure 5 show the results of some quick tests. The response on the left shows that the e-mail address (username) isn't on record, while the response on the right shows that the address is valid, but the password is incorrect.

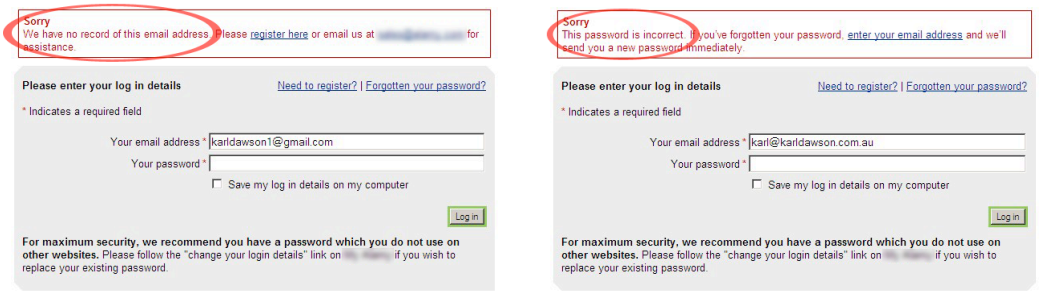


*Figure 5 - Valid and invalid credentials: very similar responses*

The difference between the responses in this example is subtle and will be affected by the length of the e-mail address, since it's returned as part of the server response. This will however still be detectable when Burp Intruder's results are viewed. Some systems will return noticeably different web pages as shown in Figure 6, making the identification

of valid and invalid usernames that much easier.

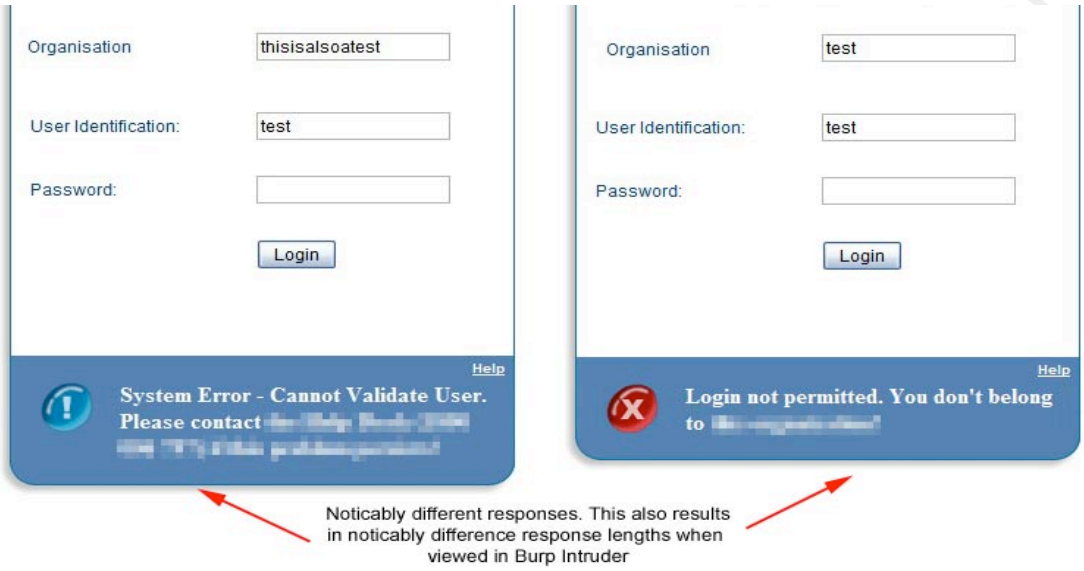


*Figure 6 – Valid and invalid credentials: noticeably different responses*

Irrespective of the differences between valid or invalid credentials, Burp Intruder can be used to distinguish between then as shown in Figure 7. This applies to username formats besides just e-mail addresses, since it's looking for how the server responds to the credentials, no just the credentials themselves.

To aid in spotting the differences between the similar logins shown in Figure 5, Burp Intruder's grep functionality can be used to look for possible keywords, such as 'error', 'invalid', 'access', and 'incorrect'. The use of the grep function is described later in this section.

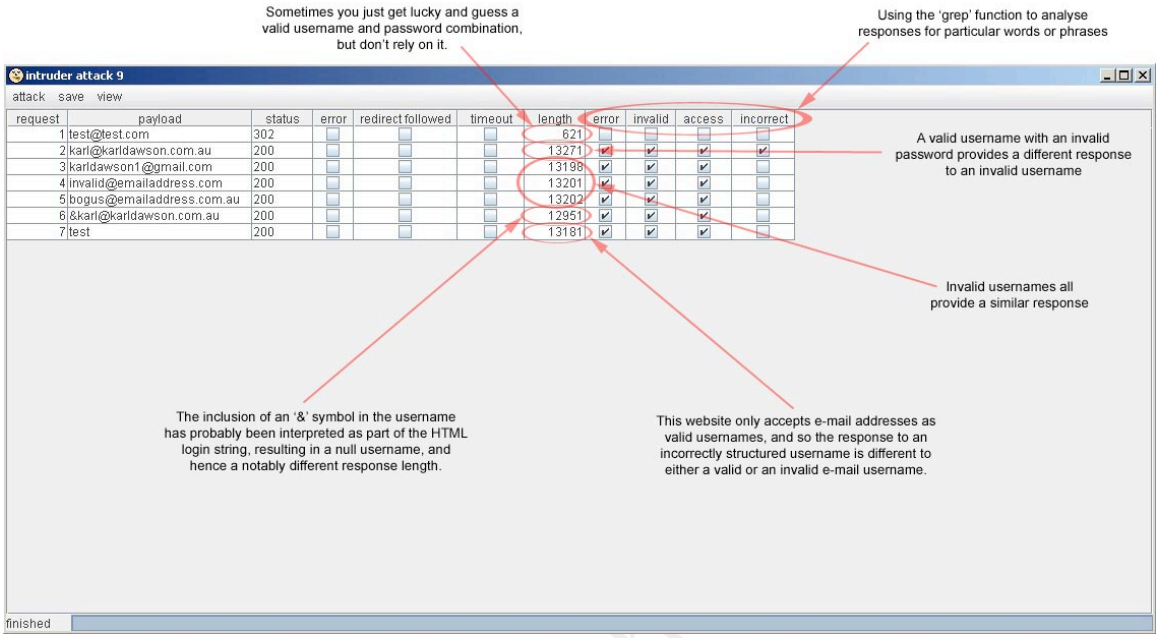


*Figure 7 - Response differences*

From the results of the seven e-mail addresses (payloads) shown in Figure 7, we're able to identify the following:

- a valid login which includes a '302 redirect' (IETF. Fielding, R et al, 1999) and an obviously different response length
- a valid username with an invalid password, with a response length approximately 70 bytes more than that of an invalid username. The grep function has also detected the keyword 'incorrect' in the server response
- 3 incorrect usernames, all with a response length of approximately 13,200 bytes
- an incorrectly formatted username and password combination through the submission of an '&' symbol, which has resulted in a response length approximately 310 bytes less than a valid username
- a slightly different response for an incorrectly formatted username (i.e. not in e-mail address format)

### 5.1.1. Using grep

Burp Intruder's grep function allows keyword search strings to be added manually or loaded from a file, and allows those search strings to be manipulated and leveraged in various ways. For the credentials discovery requirements shown here though, simple pattern matching is sufficient.

Figure 8 shows the customized list of search terms used in the previous example. The default list provided is comprehensive, however the specific responses generated by the target system should be analyzed and keywords added or removed as required.

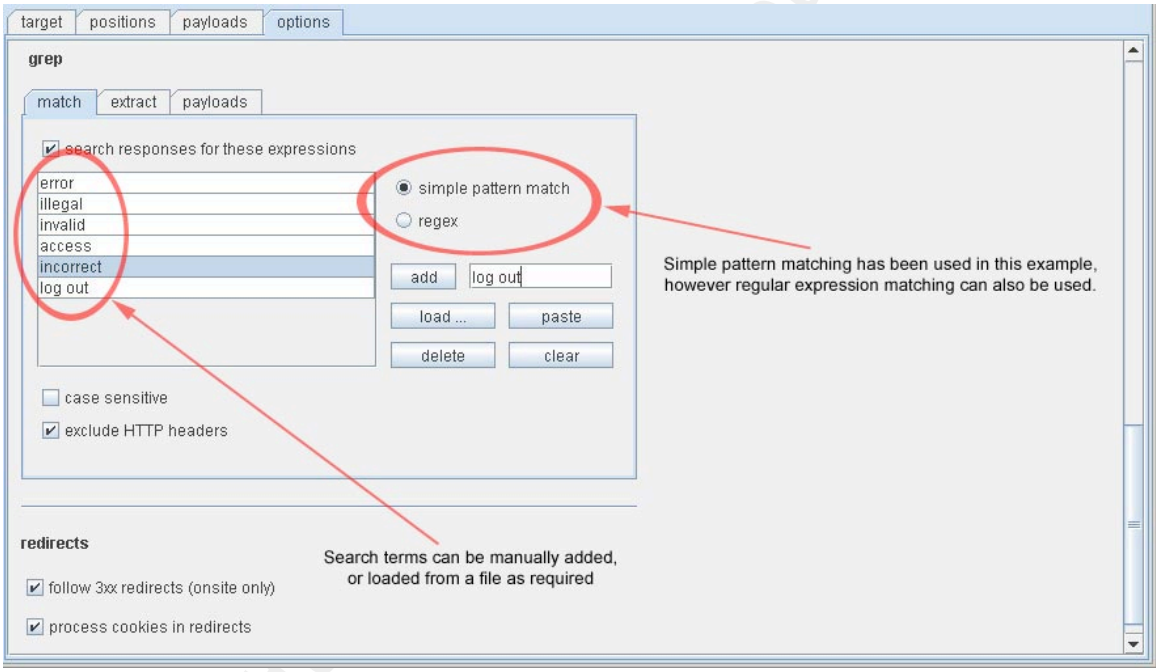


*Figure 8 – Burp Intruder's customizable grep function*

### 5.1.2. Default Usernames

A number of default usernames are included with a range of hardware and software vendor applications (CIRT, 2008), and time and effort can be saved by including these in a username list. With some of these applications the default usernames can't be changed and it's up to the administrators to either use strong passwords, or disable the access that the accounts provide. This of course doesn't always occur, and since these accounts are often linked with privileged access, their use should not be overlooked. There are a number of websites dedicated to the collection of details associated with these

accounts, including the manufacturer, product, version, username, and in some instances the password (anameless.com, 2003). From a username perspective, some of the most common include: admin, Administrator, adm, root, guest, operator, system, security, manager, debug and user (OWASP, 2009)

## 5.2. Username Payload Creation

Burp Intruder comes with its own list of default usernames as shown in Figure 9 which is accessible via the Payloads tab, and then by selecting 'usernames' from the 'add from list' drop-down list. If a small list of custom usernames has been compiled, these can be manually added to the default list or used independently each time they're required.

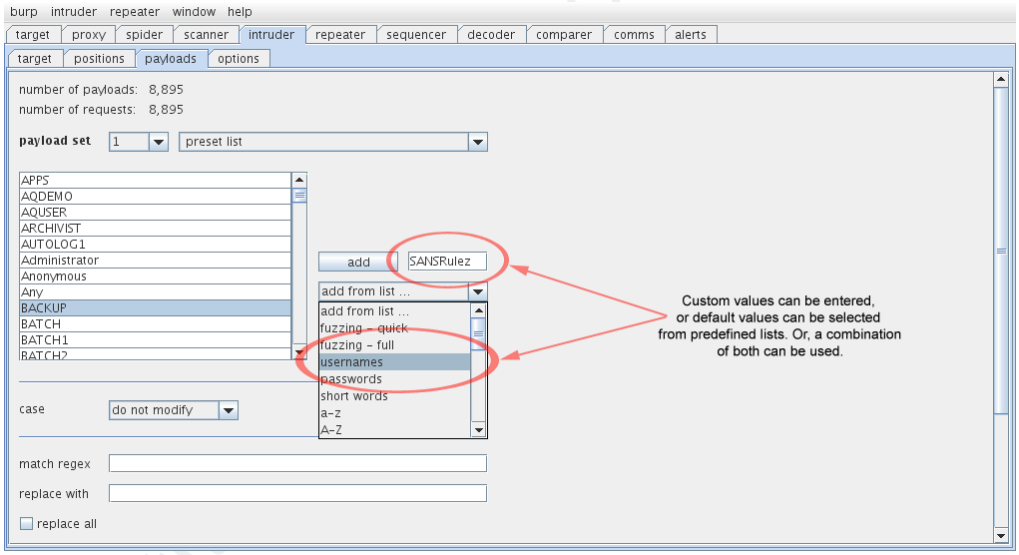


*Figure 9 - Creating a manual list of usernames*

The manual addition of usernames isn't practical for larger lists, so a suite of payload importation and generation capabilities are provided. Figure 10 shows an example of how this is performed, and Section 9.4 discusses these capabilities in more detail.

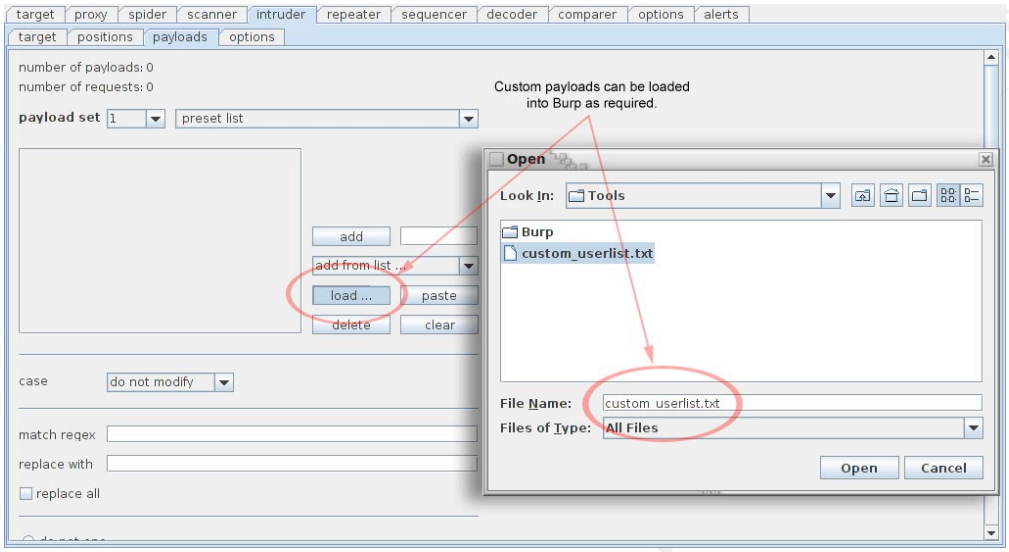


*Figure 10 - Customized payload import*

## 5.3. Anomaly Detection

So far Burp Intruder's been used to conduct a reverse brute force test against a planned and customized list of usernames. During the test, anomalies may have been detected such as usernames that respond differently to what was expected for a valid or an invalid account, as shown in Figure 11.

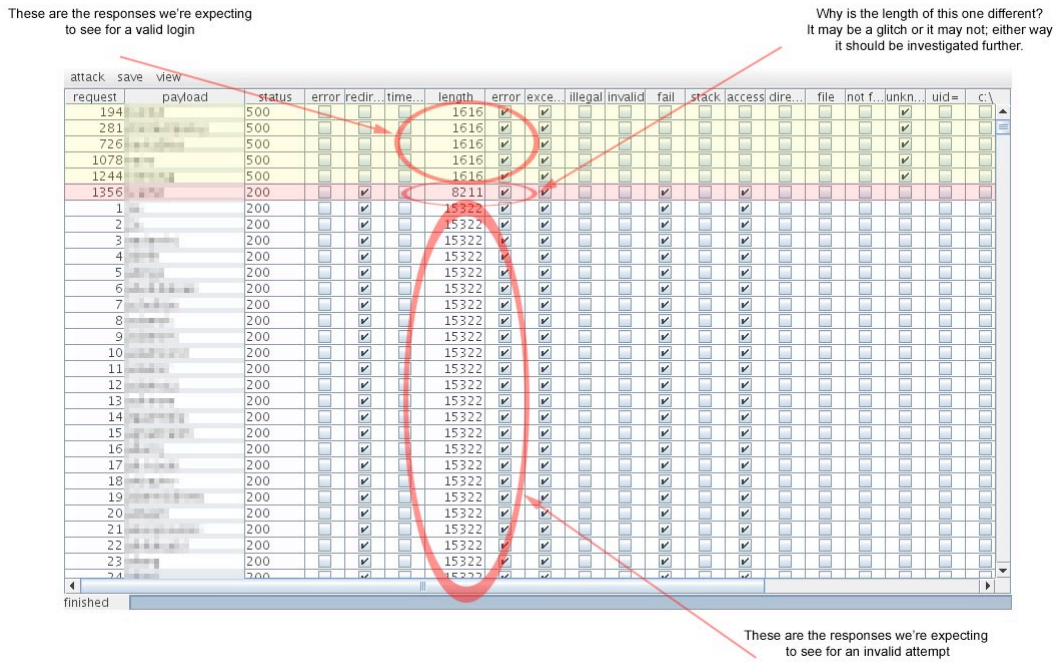


*Figure 11 – Username anomalies*

Karl Dawson, karldawson1@gmail.com

These anomalies may be uncovered during testing for usernames or for passwords. They may be glitches, or they may show the presence of accounts that provide access to different systems, or that require another round of authentication before proceeding further. Whatever the reason, these anomalies need to be targeted for further investigation. If the username and password testing methodology was to rely solely on the presence of keywords, it's likely that these anomalies would go undiscovered.

## 6. Assessing Account Lockout

At this point a refined list of usernames has been created, and the process of testing them against a password payload can begin. This could be done by just loading in the list of usernames as one payload, a list of passwords as a second payload, and selecting the 'cluster bomb' attack type to test all passwords against all usernames.

There are, however, a number of drawbacks to this, one of the most notable being the high likelihood that account retry limits will be exceeded, locking them out permanently or at least for some time. Locking out the accounts is of no benefit from a credentials discovery perspective since there'll soon be nothing left to test, but more importantly it'll effectively conduct a Denial of Service (DoS) attack against the target environment. This can be compounded further if, for example, the accounts being targeted are Internet-facing and are managed within a central directory service, and that directory service is also used to authenticate the businesses' internal users. Unless the conduct of DoS attacks has been documented in a Rules of Engagement, then testing methods that prevent it from occurring are required.

As discussed previously, Burp Intruder can be used to conduct a reverse brute force attack as part of the username discovery process. A similar testing method can also be used to test for the presence of account lockout timers, and to test for how long those timers may be.

Once these timeout values are determined, the reverse brute force attack methodology can continue to be used against all accounts, while preventing those accounts from being locked out.

This process is diagrammatically represented in Figure 12.

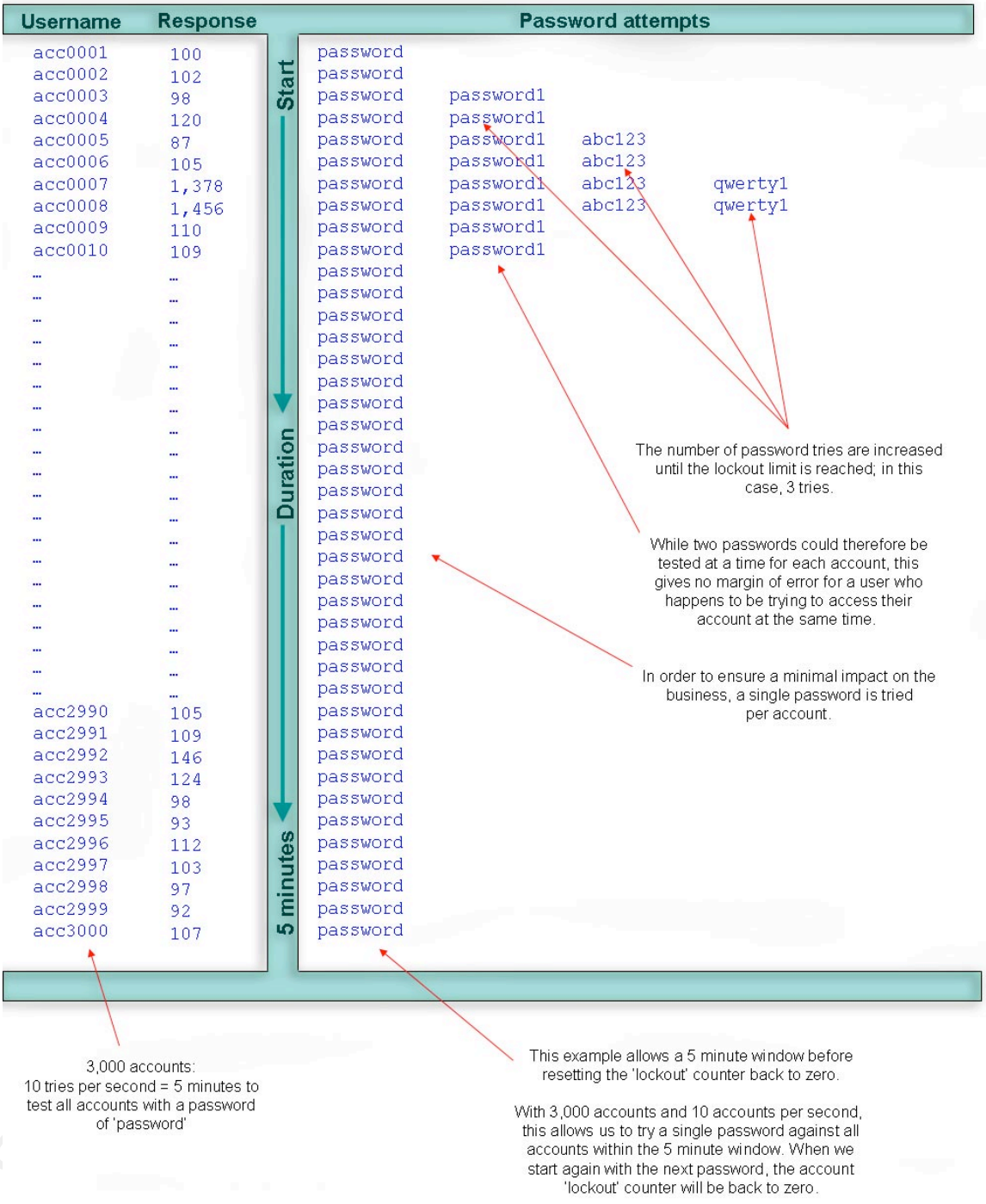


*Figure 12 – Assessing account lockout*

The example in Figure 12 is based on a custom list of 3,000 usernames, and makes the assumption that account lockout details haven't been provided to the tester. Further, it assumes that a 'test' account to test the account lockouts also hasn't been provided.

Testing begins by progressively increasing the number of password attempts on different accounts until the account lockout threshold is reached. For this example that threshold is reached after three tries, which then results in a different response being generated by the target system. This can be seen by the increased response time for a locked out account transaction.

Rather than locking out an account completely, often systems disable the account for a few minutes at a time. For this example the account lockout duration was found to be five minutes.

With 3,000 accounts and a target server that's processing login requests at approximately 10 per second, this means that by processing a single login request per account, the end of the username list will be reached just as the account retry timer is being reset on the first accounts tested. In this way Burp Intruder can be used to test entire username and password lists without locking out the account, and without creating a DoS on the target system.

Results will vary depending on the size of the username list, the account lockout duration and the speed at which the target system processes requests. A consequence of this may be a delay between the completion of testing one password against all usernames, and the start of testing with the next one. With an attack using large username and password lists, that have been researched and tailored for the system under test, there's a good chance that the first pass will provide at least some sort of result. The down-time can then be spent on further investigation rather than sitting and waiting for timers to expire.

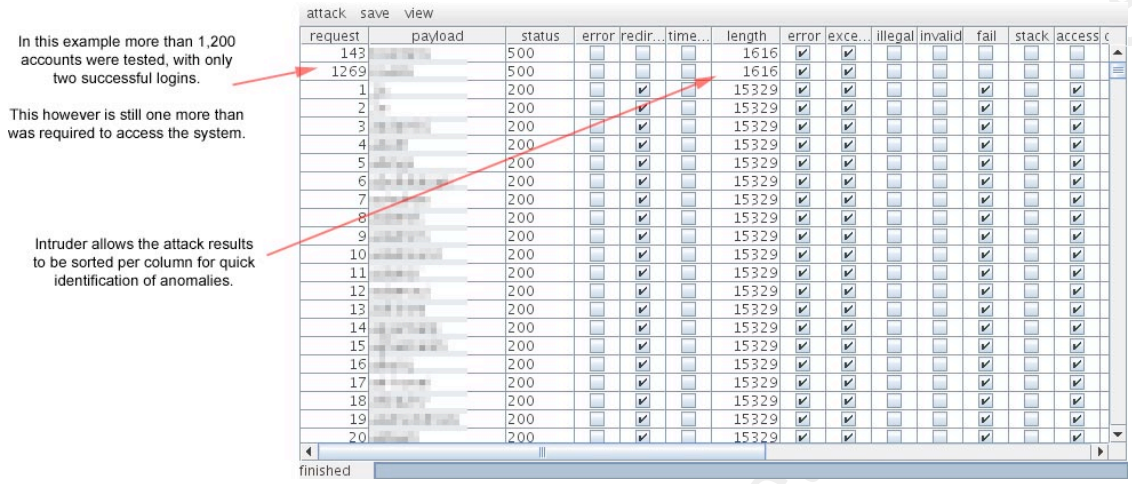


*Figure 13 - Sorted results*

As shown in Figure 13, even only two compromised accounts out of a username list of more than 1,200 still provides two points of deeper access into the system. Even if those accounts don't provide the desired access, it's possible that further investigation into the information and systems they do contain may yield previously unknown information on other accounts, or other areas of the application.

## 7. Adjusting Timing

Rather than waiting by the clock before kicking off the next round of password attempts, Burp Intruder's timing settings can be configured to tune the way in which requests are sent to the server. The timing settings are available under the Options tab as shown in Figure 14.

The main setting to note here is the 'throttle' which can be set as either a fixed or variable delay, and is measured in milliseconds. In the example shown, a fixed delay of 500ms between requests has been defined, and has been used in conjunction with a 'concurrent request' setting of one. The concurrent request setting defines how many parallel requests are sent to the server before waiting for a response.

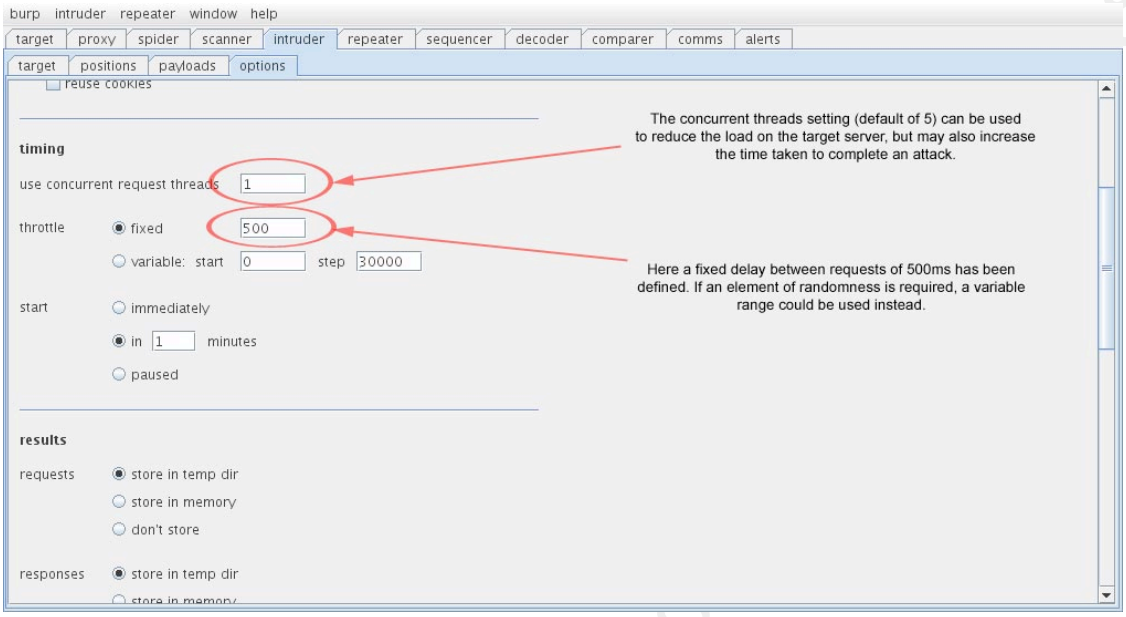


*Figure 14 – Burp Intruder timing settings*

Testing for account lockouts may have shown that no such restrictions are in place. If this is the case then Burp Intruder's timing settings, in conjunction with an eye on the results display, can be adjusted to maximize the rate at which passwords are tried without exceeding bandwidth or server processing capabilities. A variable throttle setting could then be used to reduce the likelihood of the requests looking exactly like they are; part of an automated attack.

## 8. Dealing with HTTP Basic Authentication

All of the attack methods so far have assumed that HTTP form-based authentication is used on the target system. With form-based authentication, Burp Intruder is able to intercept the username and password fields via Burp Proxy, making the manipulation of them straightforward. With basic authentication however, the credentials – in the form 'username:password' – are Base-64 encoded prior to being received by the proxy, meaning that the credentials appear as an encoded stream of text rather than distinct usernames and passwords. As such, the direct manipulation of the usernames and passwords at first appears problematic.

Figure 15 shows a way around this, through the use of the 'Base-64 encode'

functionality available via the Options tab. Using this it's possible to mark the encoded text under the Positions tab for fuzzing, and supply it a payload as per the usual method. A caveat however is that both the username and password need to be passed as part of the payload, and so a standard password file isn't enough. Burp Intruder provides a workaround for this, by allowing a prefix or suffix to be added to the password as shown in Figure 15.

An alternative is to use the array of payload generation techniques to create payloads that include the 'username:' portion of the credentials as part of the password. Burp Intruder's payload generation capabilities are described in section 9.4, Payload Generation.

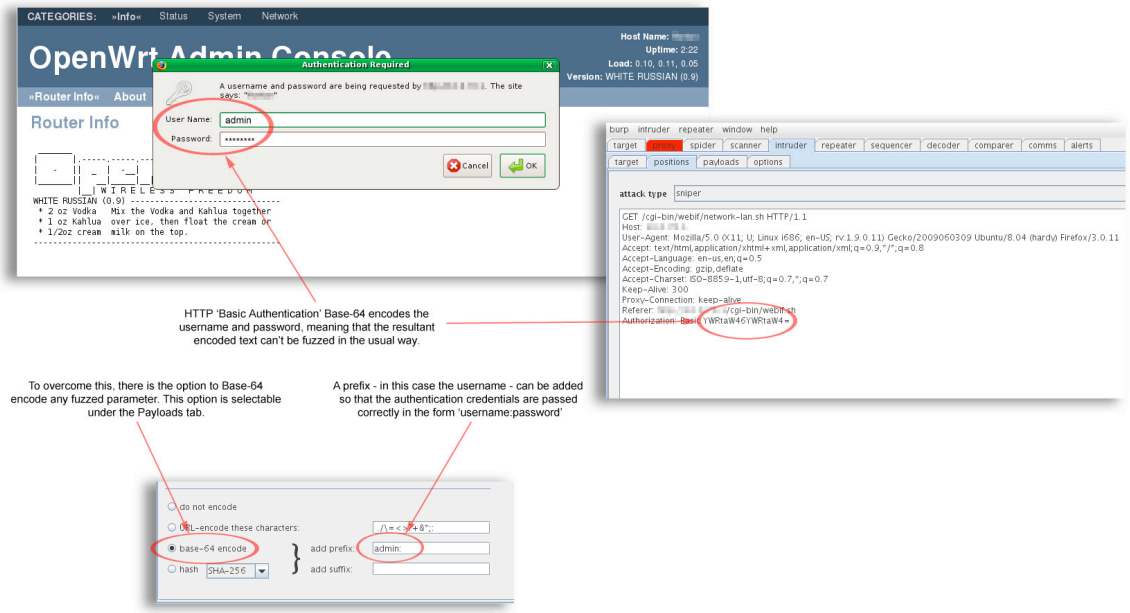


*Figure 15 - Dealing with HTTP basic authentication*

## 9. Password Discovery

The tailoring of passwords to a specific system is critical for most on-line applications due to speed and resource constraints. During the username discovery phase, research was carried out to try and determine if usernames had to conform to a particular

structure, such as being based on an e-mail address. Tests were then conducted to see if the responses from the target system disclosed which of the correctly structured usernames might be valid and which might be invalid.

## 9.1. Password Structure

These same processes apply to the discovery of passwords. The structure of a valid password needs to be determined as thoroughly as possible so that the resultant password list can be tested against all usernames in a realistic timeframe. An example of the need for this is demonstrated in section 9.4.3, Payload – Brute Force. This shows that it would take more than 70 days to cycle through all of the ~62,000,000 permutations and combinations for an alpha-numeric password only five characters long. Clearly, being able to narrow down the number of combinations that are needed is a vital consideration.

The other aspect, being able to determine a valid as opposed to an invalid password, is of course the 'end game' of this whole exercise, and not an intermediary step like it was with the usernames.

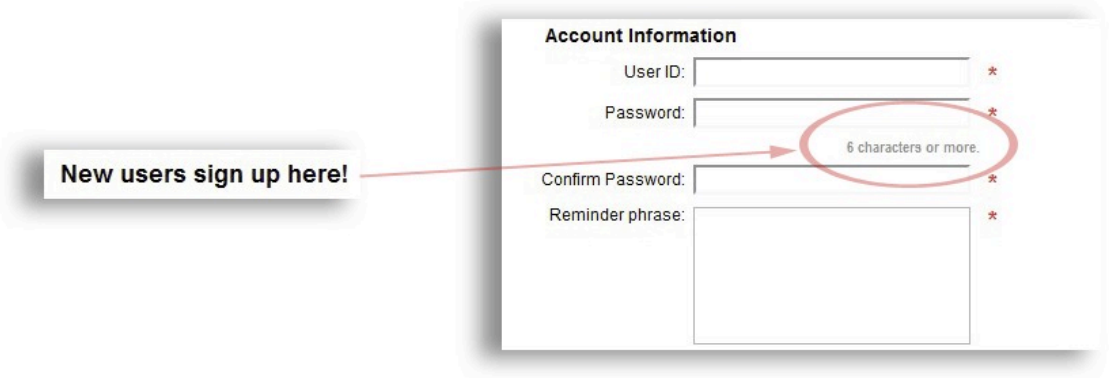


*Figure 16 - New user password information*

Determining password structure may be as easy as having the site tell you what the requirements are, as shown in Figure 16. This of course may only be applicable to sites that allow users to create their own accounts, and even then it may only show what the minimum requirements are. While this doesn't let us know what the upper limits are it may still be helpful. Knowing that a password only needs to be six characters long means that there will no doubt be people who follow the path of least resistance and create a

password to meet that minimum requirement.

Another possibility is to gain access to a password policy document that explains details such as minimum length, expiry periods and complexity. This however is usually only available during 'white-box' testing, and so isn't considered a viable option here.

Researching the specific habits of an organization can also be of use. Many organizations set user's passwords to a common default setting for new accounts, or reset them to a default setting if a password's forgotten. Also, not all organizations enforce the changing of a password at next login, so there's always a chance that there will be active accounts with the default password still enabled. If not, 'password' or '123456' are always good starts, particularly with a large user-base. Geographic, organizational, cultural, sporting and demographic influences, as well as political and socioeconomic factors should also be taken into consideration (Brown, 2006).

In addition, techniques such as crawling the company's website and the websites of targeted employees, as well as searching social networking sites such as FaceBook, LinkedIn and Twitter may provide keywords that can be used as the basis of a targeted attack.

## 9.2. Logging Server Responses

Burp supports the ability to collect this sort of information, as well as required password lengths and potential complexity requirements, by logging all server responses as shown in Figure 17. There's the ability to log the requests and/or responses from each of its tools individually, or as an aggregate as shown in the example, and is selectable under the 'Options' tab. If the option of logging all responses to a single file is selected, the size of the file should be monitored. It can quickly grow to be many megabytes, and can become unwieldy to work with.

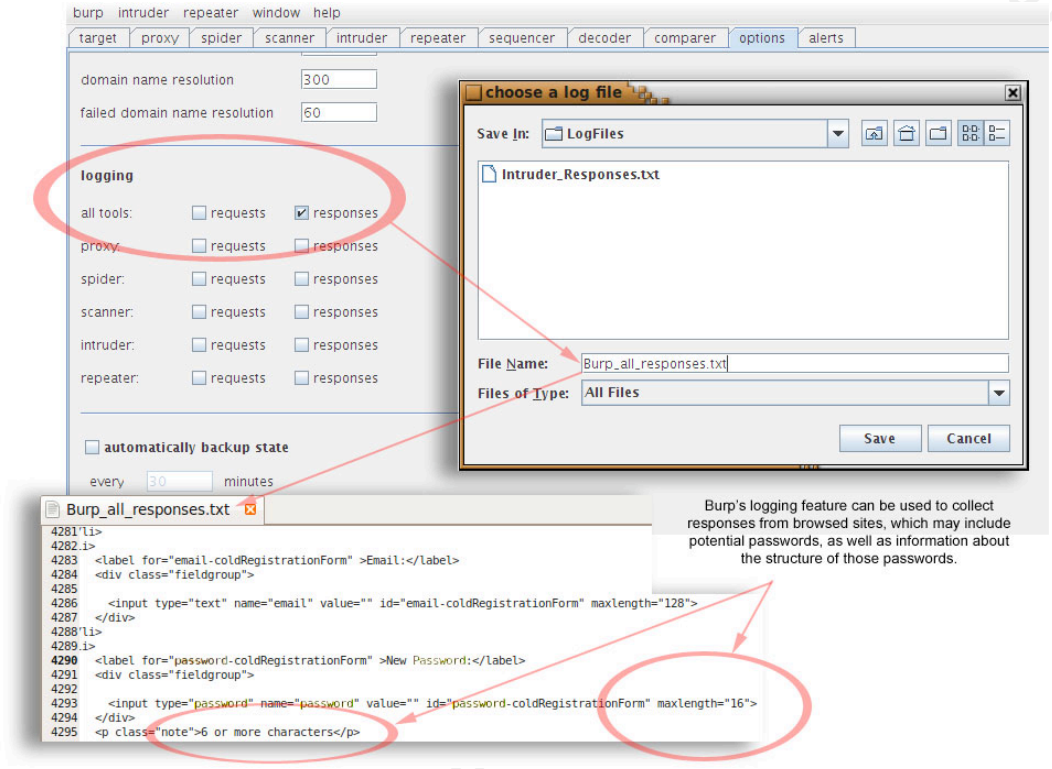


*Figure 17 - Burp response logging*

In Figure 17, the option to log all server responses was selected. The social networking site profiles of targeted employees were then browsed, resulting in the pages being logged to a flat file. Potential password candidates were jotted down manually as the profiles were examined, but some information might have been missed, and so the Burp logs act as a useful archive.

The logs contain the full HTML source code of each visited page, making them difficult to read when trying to spot specific information such as people's hobbies, pet's names, or favorite holiday destinations. There are however a variety of tools and command-line options for stripping away the HTML noise, leaving just the site's text as shown in Figure 18.

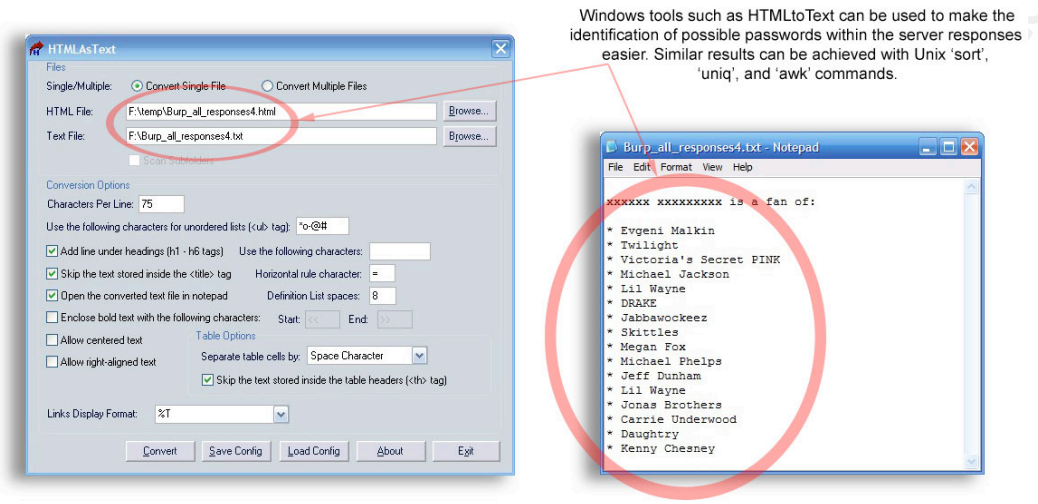


*Figure 18 - Extracting possible passwords from Burp logs*

## 9.3. The Human Element

Discovering online credentials is as much about the research as it is about the technology used to expose them. Knowing the target and knowing the way in which people structure passwords in general goes a long way towards a successful exploit.

The Human Element was touched on in section 4, and used the MySpace hack as an example of password usage in 2006. A more recent example is the output from the 2009 'Zero For 0wned' exploits, as published in their ZF05.txt e-zine. These exploits provide a useful example of the password structures currently in use by a broad range of the IT community (Rootsecurity 2009). Accounts were compromised from the websites of Perlmonks.org and Elitehackers.info; the former being a more mainstream site used by a range of IT professionals, and the latter (as described by ZF0) as being more frequented by script kiddies.

An examination of the compromised accounts shows that none of the Perlmonks.org users had a password of 'password', whereas at least a dozen of the Elitehackers.info accounts did. Users of both sites still made use of dictionary words such as 'william', 'chicken', 'insecure', 'kenny', and 'easter', as well as numbers such as '123456' and '112233'. There was also extensive use of character and case substitution. Genre and site-specific passwords also make a notable appearance, with the likes of 'whiterabbit', 'MonkPerl', 'eminem' and 'limpbizkit'.

Users from both web sites made use of passwords from the "Top 500 Worst Passwords" list (Burnett, 2005), such as 'qwerty', 'letmein', and 'trustno1', although some went to the trouble of adding case and/or character substitution.

This knowledge of the current state of general password usage, in conjunction with specific knowledge collected on the target environment, can then be carried forward into the password generation process.

## 9.4. Payload Generation

Once the research for the target system has been completed, the username and password payloads for that system can be generated. These payloads can be generated using an external application and then loaded into Burp Intruder, or its payload generation capability can be used; or a mixture of both.

Burp Intruder offers a suite of payload generation options, all accessible under the Payloads tab. Since its functionality has a broader scope than just attacking credentials, not all of the available payload options are useful for generating usernames and passwords. Also, the generation techniques listed may be used to assist in the credentials discover process in ways that aren't covered here, and which will depend on the specifics of the environment being tested.

As an example, Figure 19 uses the grep function with a numerical payload to search for possible hidden portal access within a website. In this example, the numerical payload was used to cycle through some combinations of a website's URL structure, while the grep function was used to look for the keywords 'Username' and 'Password'. In this way, the payload generation capabilities are able to be used to not just gain access to accounts, but to find where they may be in the first place.

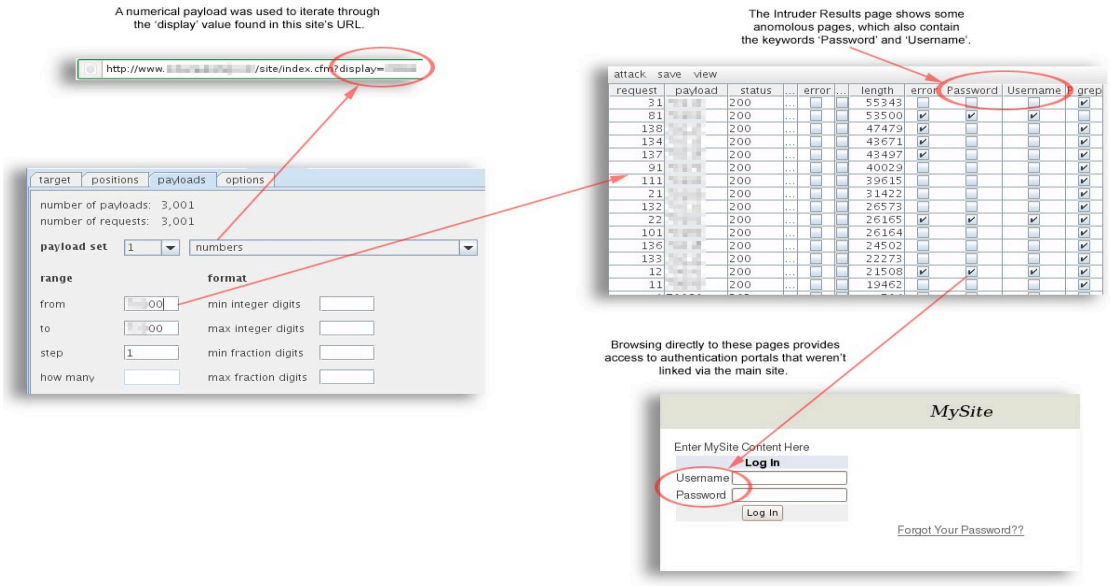


*Figure 19 - Finding authentication portals*

The following sections then, cover those generation techniques that can most commonly be used to aid in the username and password discovery processes. This includes dictionary attacks, character and case substitution, as well as more complex tasks such as the creation of email addresses from lists of first and last names.

### 9.4.1. Payload – Username or Password Files

Burp Intruder has a range of default payloads, including usernames, numbers, and of course passwords as shown in Figure 20.

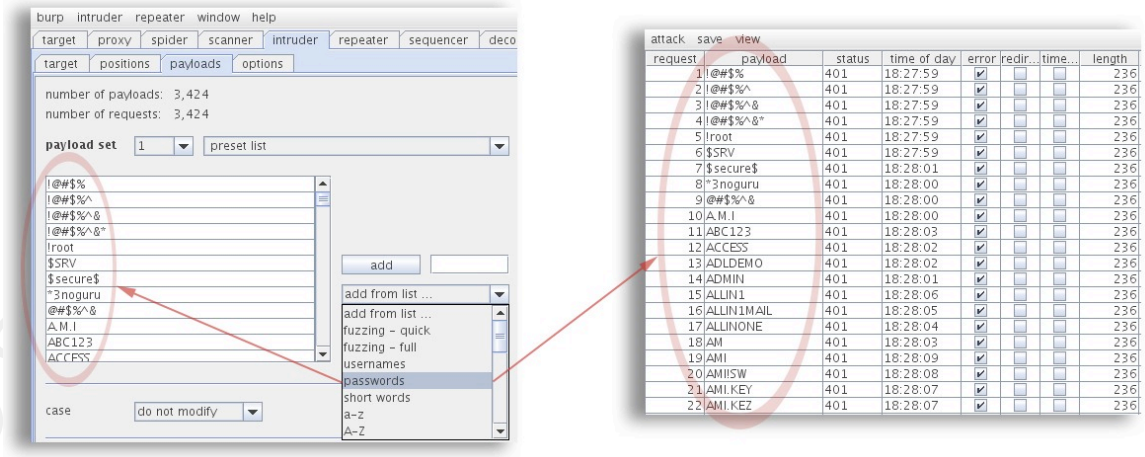


*Figure 20 - Payload – Default Password File*

Karl Dawson, karldawson1@gmail.com

The example shows the default password file. For larger engagements where the list may be used repeatedly over an extended period of time, there's the ability to customize the default payload strings. This is achieved by editing and/or adding text files to the '/burp/Payload Strings/' directory. This allows the customized list to be quickly selected from the payloads drop-down list rather than needing to import a file each time.

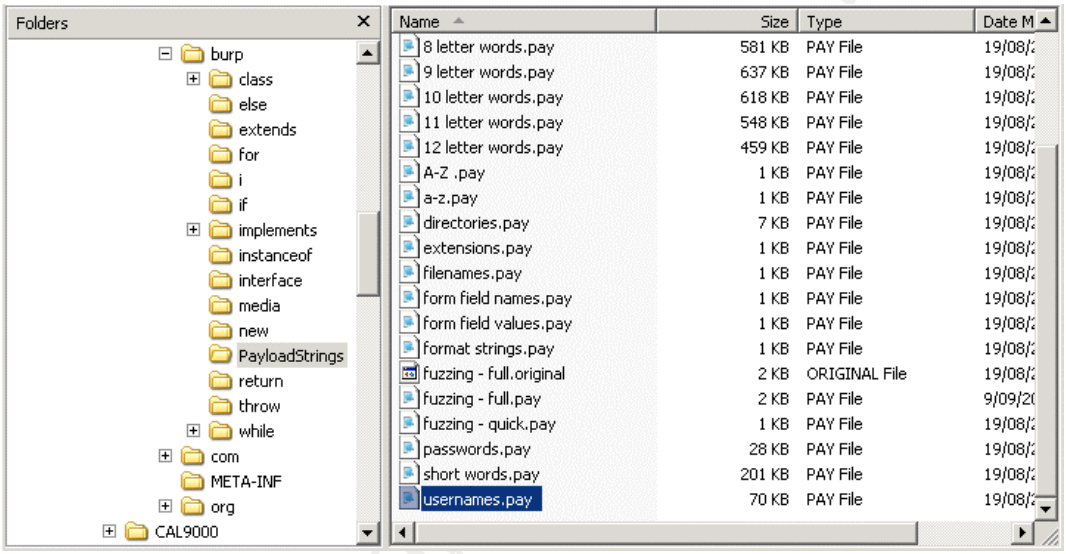


*Figure 21 - Payload directory*

Alternatively, custom files can be loaded as shown in Figure 22, and can be of benefit where the username or password file is being constantly updated.

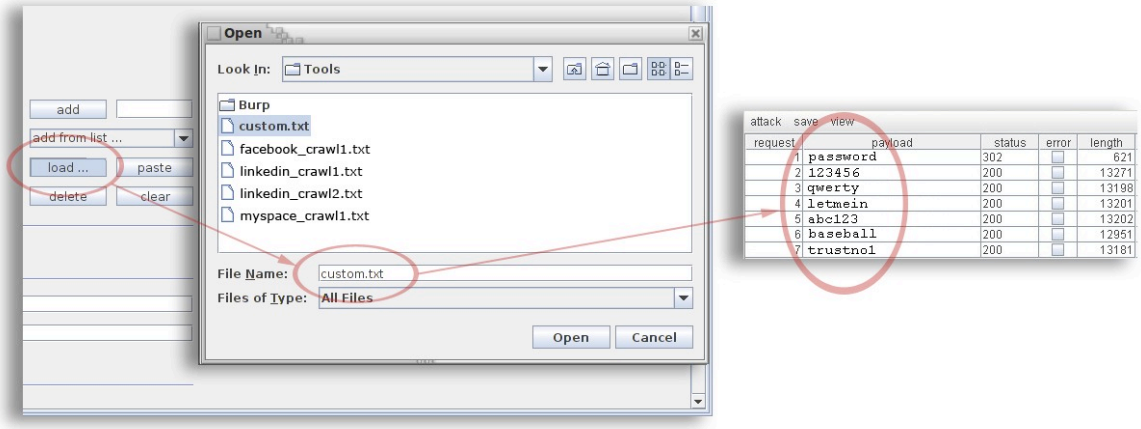


Figure 22 - Payload - Custom Password File

Given that relatively small password files are typically going to be used as part of the on-line as opposed to off-line testing process, this method of password generation and

importation is likely to be one of the most beneficial.

### 9.4.2. Payload – Runtime File

A drawback with the importation of password files using the method shown in Section 9.4.1 is that since the files are loaded into Burp Intruder before use, they consume system resources. An alternative is to use the Runtime payload, which reads a single record in at a time from an external file, as shown in Figure 23. Such an approach may be useful where system resources are limited, or where a very large list of passwords is to be used.

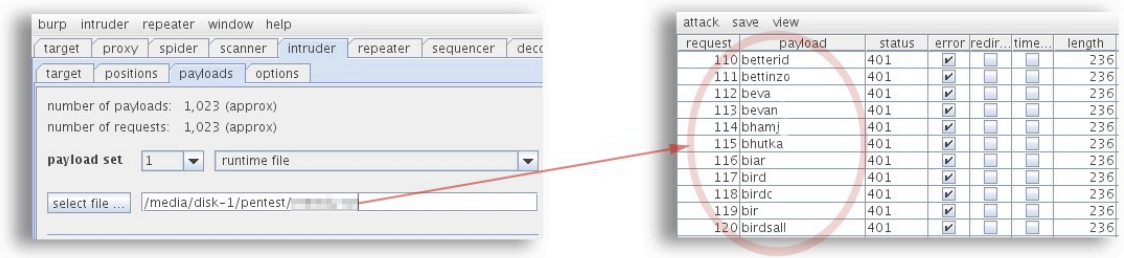


*Figure 23 - Payload - Runtime File*

### 9.4.3. Payload – Brute Force

As its name suggests, the Brute Force payload allows for the configuration of the minimum and maximum length of a password, as well as the characters that make up that password. As shown in Figure 24 this payload works sequentially and methodically up through the character set selected, so in the example shown, each time the attack is run, it will start at 'aaaa', then move on to 'baaa' and 'caaa', and so on until all iterations are complete. As such, this payload method may not be suitable for use with long passwords, or where research hasn't been able to narrow down the character set.

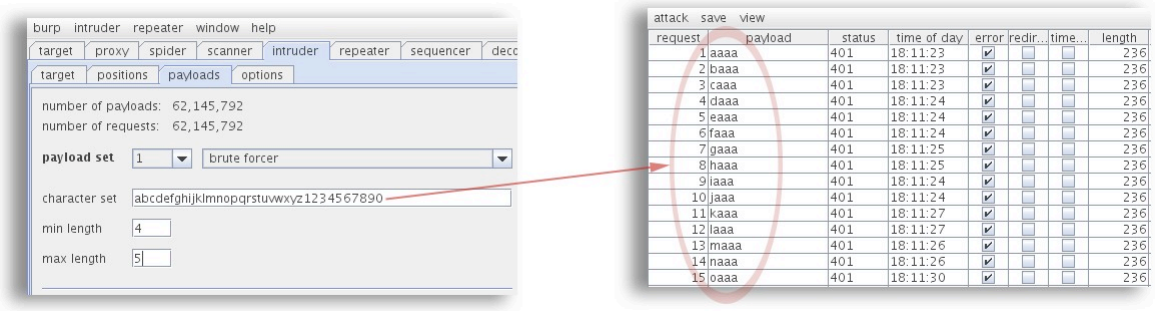


*Figure 24 - Payload - Brute Force*

Based on the previous example of 10 passwords per second, it would take more than 70 days to iterate through all of the ~62,000,000 permutations and combinations available with an alpha-numeric password between four and five characters long. For passwords longer than six characters this is unlikely to be a realistic avenue of attack.

A possible option to reduce this total number of permutations is to remove characters that have a low occurrence rate in the English language, such as ‘k’, ‘v’, ‘x’, ‘z’, ‘j’, and ‘q’ (askoxford.com, 2009) This would reduce the permutations and combinations by more than half to ~25,000,000. Including only the numbers ‘0’ and ‘1’ will significantly reduce this even further.

### 9.4.4. Payload – Case Substitution

As shown in the MySpace, Perlmonks and Elitehackers examples, passwords based on dictionary words are still in widespread use. Case substitution can be performed on an old password rather than needing to generate a new password, and so Burp Intruder provides two methods of performing this as shown in Figure 25.

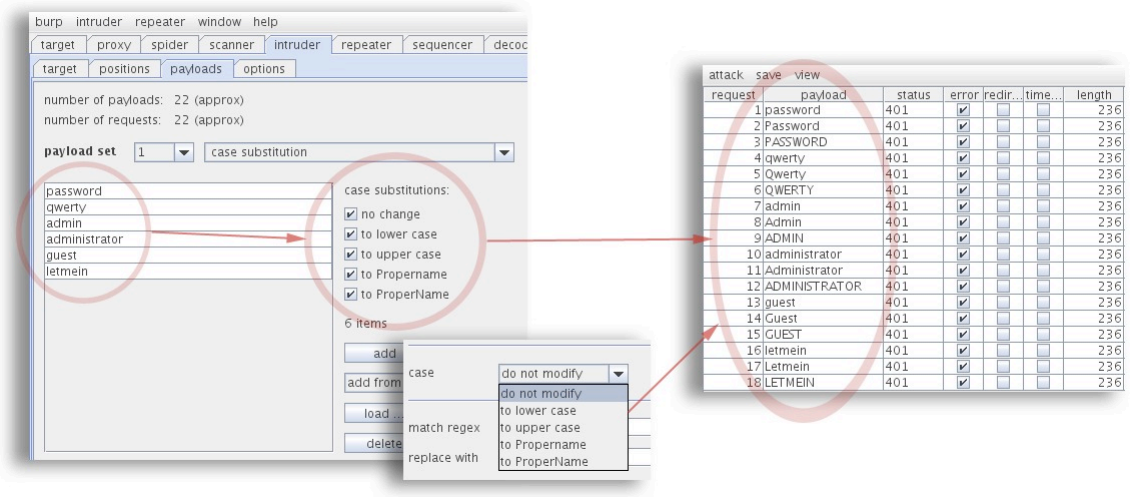


*Figure 25 - Payload - Case Substitution*

This may also be of use in instances where the imported password file has come from the likes of a website crawl or other automated data collection technique where words with a mixture of case may be present. The case substitution payload can then be used, for example, to make sure that all words in the file are submitted as lowercase.

### 9.4.5. Payload – Character Substitution

Character substitution is one of the more useful generation techniques from a password perspective. As shown in Figure 26, this payload allows a list of characters to be defined, as well as the corresponding values that they should be substituted with.

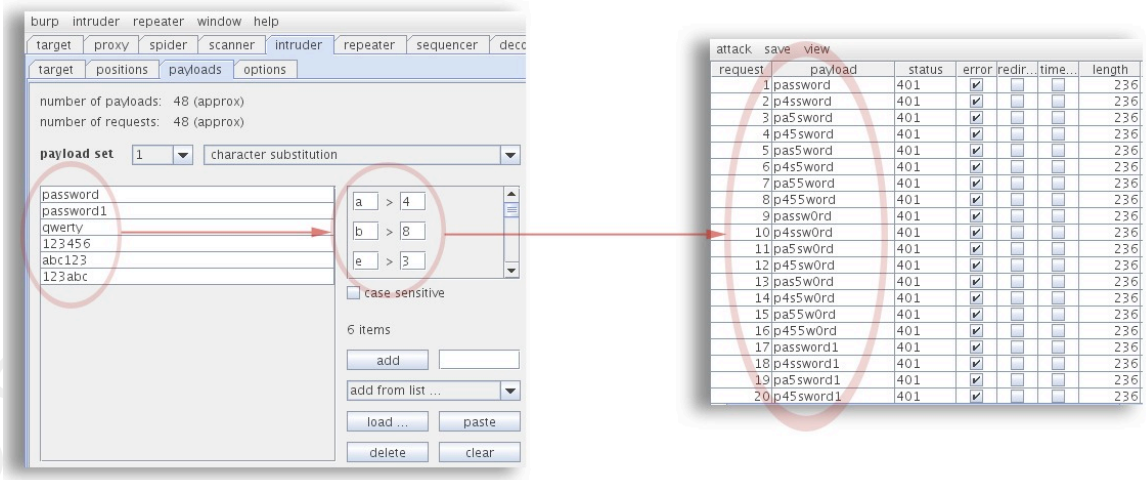


*Figure 26 - Payload - Character Substitution*

With ‘leetspeak’ being used as a common method of extending the life of an existing password or adding complexity to a new one, this payload allows for up to sixteen different character substitutions to occur. Only simple substitutions are allowed, such as changing ‘a’ to ‘4’ and ‘b’ to ‘6’; changing to multiple characters such as ‘a’ to ‘/\’ or ‘b’ to ‘13’ isn’t currently supported.

Using this payload, substitution changes can be made quickly without the need to re-edit a password file. It also allows changes to be applied across multiple files, such as those collected from a web-crawl or a names database, without impacting the integrity of those files.

### 9.4.6. Payload – Numbers

Applicable to sites that make use of PINs, the Numbers payload allows for the generation of a range of numbers ‘from’ a value ‘to’ another value, in whatever number of ‘steps’ are required.

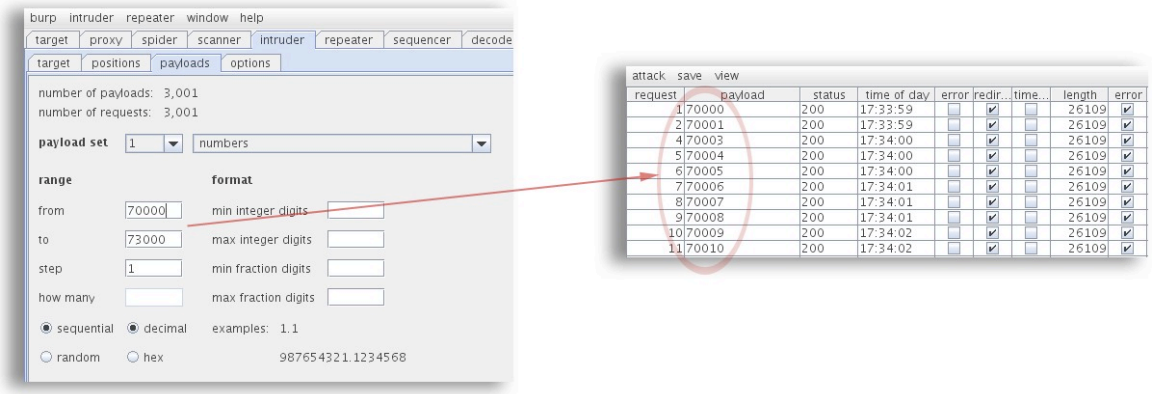


*Figure 27 - Payload - Numbers*

These numbers can then be generated sequentially as shown in Figure 27, or randomly by selecting the ‘random’ button and specifying how many numbers are to be generated in the ‘how many’ field.

The numbers payload is a good example of the vital role that research into the structure of usernames and passwords can play in the success of an attack; a dictionary attack is very likely to fail against a site expecting six digit PINs.

### 9.4.7. Payload – Custom Iterations

Burp Intruder's Custom Iterator provides the ability to quickly generate extensive password lists that may be based on a couple of key words or values that are related to a particular target. Each word or value is entered into one of eight 'positions', which are then joined together (with a separator if required) in the final list.

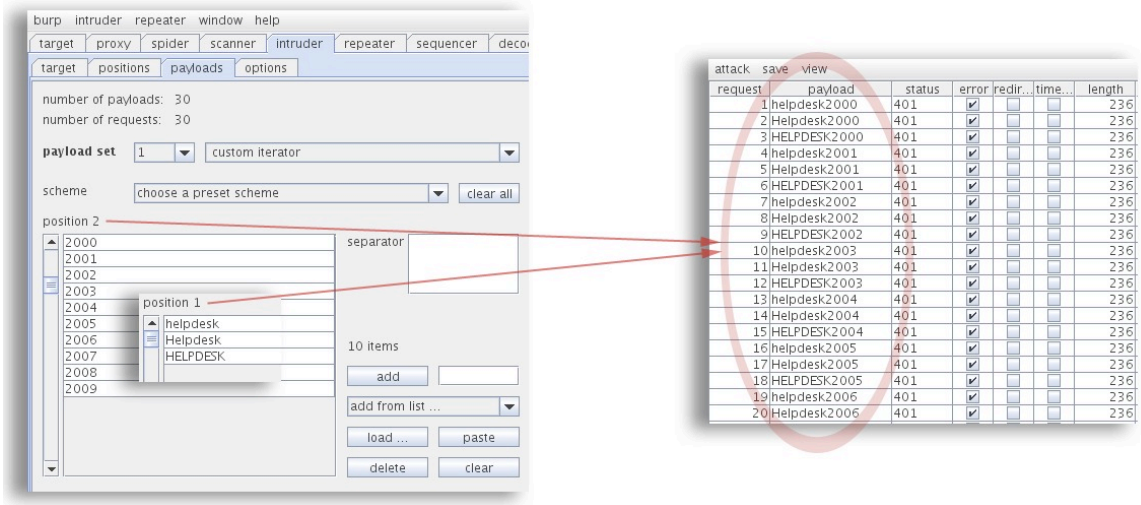


*Figure 28 - Payload - Custom Iterations*

In the example shown in Figure 28, the password is assumed to be based on the word 'helpdesk' and the year the password was created. Three variations of the word 'helpdesk' have been loaded into position 1, with the years 2000 through to 2009 loaded into position 2. The final attack will iterate through each of the items listed in position one in conjunction with the first record in position two, followed by each of the items in position one in conjunction with the second record in position two, and so on until all iterations are complete. This could be used equally well in the generation of username payloads.

The Custom Iterator also comes loaded with some preset iteration schemes. One of these, the 'passwords + digit' scheme, is applicable for credentials generation and is shown in Figure 29. This preset iteration uses Burp Intruder's default password list in conjunction with the values 0 – 9 and a – z. The preset values can be modified if required, such as removing the characters a – z so that only numbers are appended to each of the passwords. Unless the underlying source files for the passwords or digits are changed,

these preset values will be reloaded the next time Burp Intruder is started.

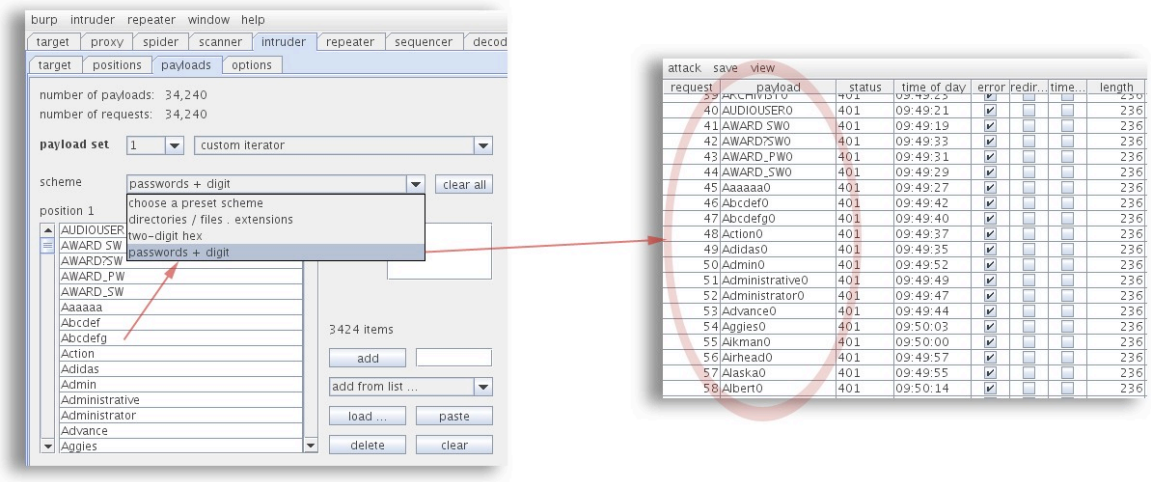


*Figure 29 - Payload - Preset Iterations*

## 10. Saving Results

Saving results doesn't mean the end of the process, but rather is something that is likely to occur at multiple points throughout the testing exercise. The results of various attacks may need to be compared with each other to identify further patterns or anomalies that can be exploited further, or they may just need to be saved for auditing at a later date. Attack configurations may also need to be saved so that they can be reliably repeated at a later date either for the current exercise, or as part of a future one.

The 'Save' option within the Results window provides these abilities as shown in Figure 30. Three options are provided: save the attack, save the results table, or save the server responses; the option chosen will depend on the purpose and the size of the attack.

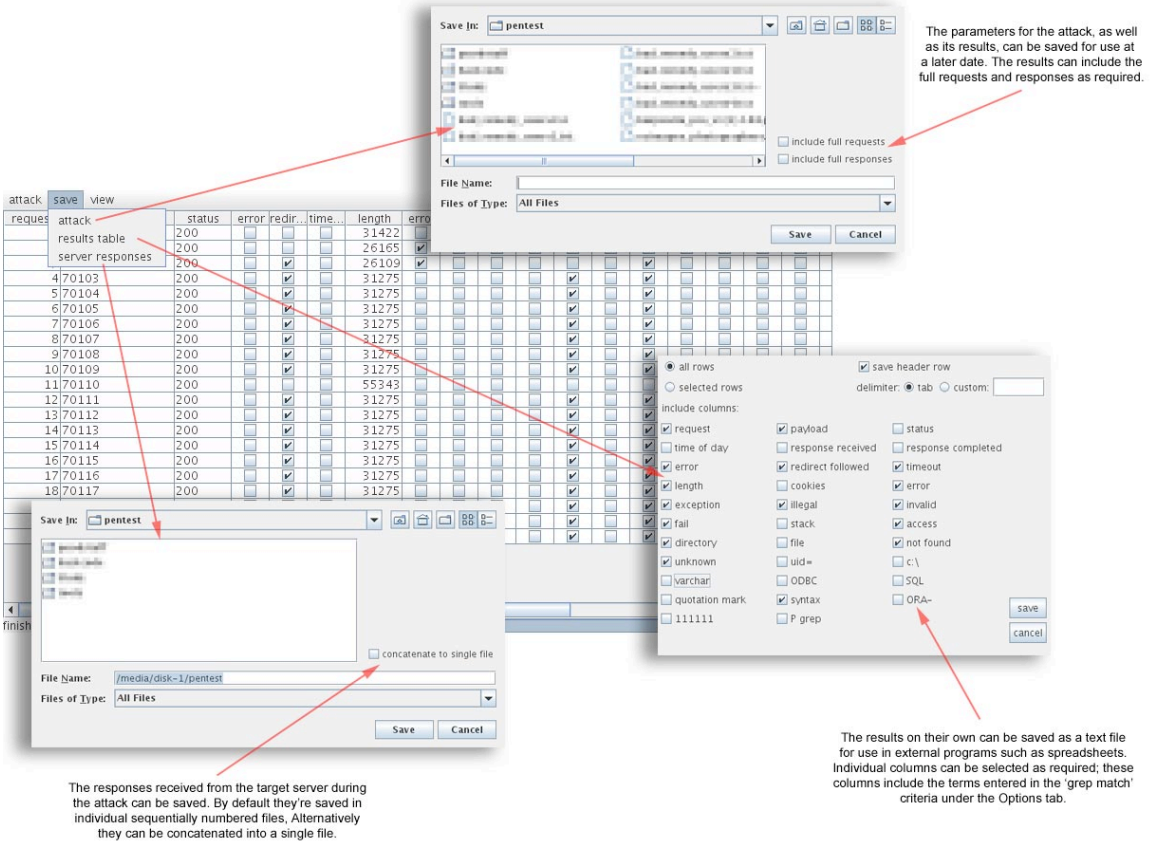


*Figure 30 – Burp Intruder 'Save' options*

## 10.1. Save – Attack

The 'Save – Attack' option saves all of the parameters associated with the attack, as well as the results generated during the attack. This save method also provides the option of saving either the full requests sent to the target or the full responses from the target, or both. The saved attack can be reloaded by selecting 'Intruder/open' from the main Burp menu. The results saved in this format aren't readily available for viewing with an external program; they would also need to be saved using the Results Table option if this capability was required.

The ability to reload all of the parameters associated with a particular attack can help to save time during an exercise, particularly if multiple attacks against different systems are being conducted at the same time.

### 10.2. Save – Results Table

The 'Save – Results Table' option saves the attack results in a format that's readily viewable in external programs such as spreadsheets. This option also allows for all information to be saved, or for the selection of only those columns that are relevant to the attack, as shown in Figure 30. The columns displayed include those that relate to terms that have been entered in the 'grep match' criteria under the Options tab. The selected columns may therefore only need to be the payload, such as the username and/or password, and the column showing whether the grep function detected a match.

This save method should be used at the completion of each attack.

### 10.3. Save – Server Responses

The 'Save – Server Responses' option effectively provides a debugging capability by saving all of the responses returned by the server during an attack. This is particularly useful during the discovery process as a way of identifying possible keywords. It can also be used to search through the site's source code to look for username and password structures and boundaries. By default, server responses are saved in individual sequentially numbered files, but can also be grouped into a single file by selecting the 'concatenate to single file' option.

## 11. Passwords Discovered

Each aspect of the credentials discovery has now been discussed, and so an example of the end to end process can now be demonstrated. This example is based on the results of a number of real-world engagements, all of which ended in the successful compromise of the target systems by using the described approach. For the sake of a name, the target organization will be referred to as IntruderCorp. IntruderCorp is a mid-sized organization that provides an online portal to a variety of backend applications for its employees and some customers. These applications include web-based e-mail, asset management systems, and directory services. No multi-factor authentication solution had been implemented.

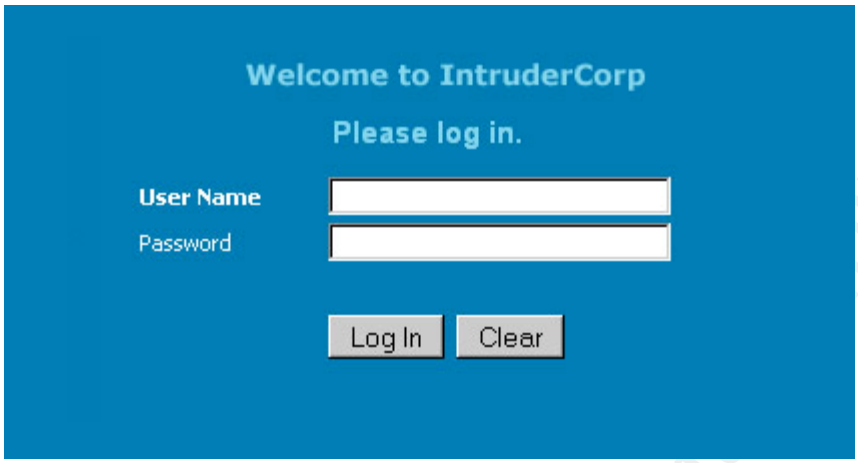


*Figure 31 - IntruderCorp login*

## 11.1. Usernames

IntruderCorp's portal had been designed not to leak information regarding the differences between valid and invalid usernames, however an analysis of the server responses showed that the username was required in the format of an e-mail address. Researching the organization allowed a list of a few hundred e-mail addresses to be compiled. Using these addresses a first pass attack was conducted, in conjunction with the passwords 'password' and 'password1'. A valid account was found, but it only had limited access. It did however have access to the IntruderCorp corporate address book, and from that a complete list of current e-mail addresses was obtained. Testing with this account also showed that account lockouts were disabled.

## 11.2. Using the Human Element

Using the name, address and phone number details associated with the compromised account, a call was made to the IntruderCorp help desk claiming that the password had been forgotten, and needed to be reset. The help desk obliged, and reset it to the corporate default of 'Helpdesk2009'. In addition to this, no change of the password was required upon first login.

## 11.3. Preparing Intruder

The complete list of e-mail addresses totaled more than 770, and this was loaded in as a custom username payload file. It was now suspected that passwords needed to

contain numbers, since both of those discovered contained them (password1 and Helpdesk2009), and there were no occurrences of the password 'password'.

With no account lockouts to worry about, four passwords were loaded as a second custom payload. The additional two passwords were chosen from the list of 'Top 500 Worst Passwords' (Burnett, 2005). In order to test all passwords against all usernames, the cluster-bomb attack type was selected, as shown in Figure 32. This results in more than 3,000 username/password combinations being tried.

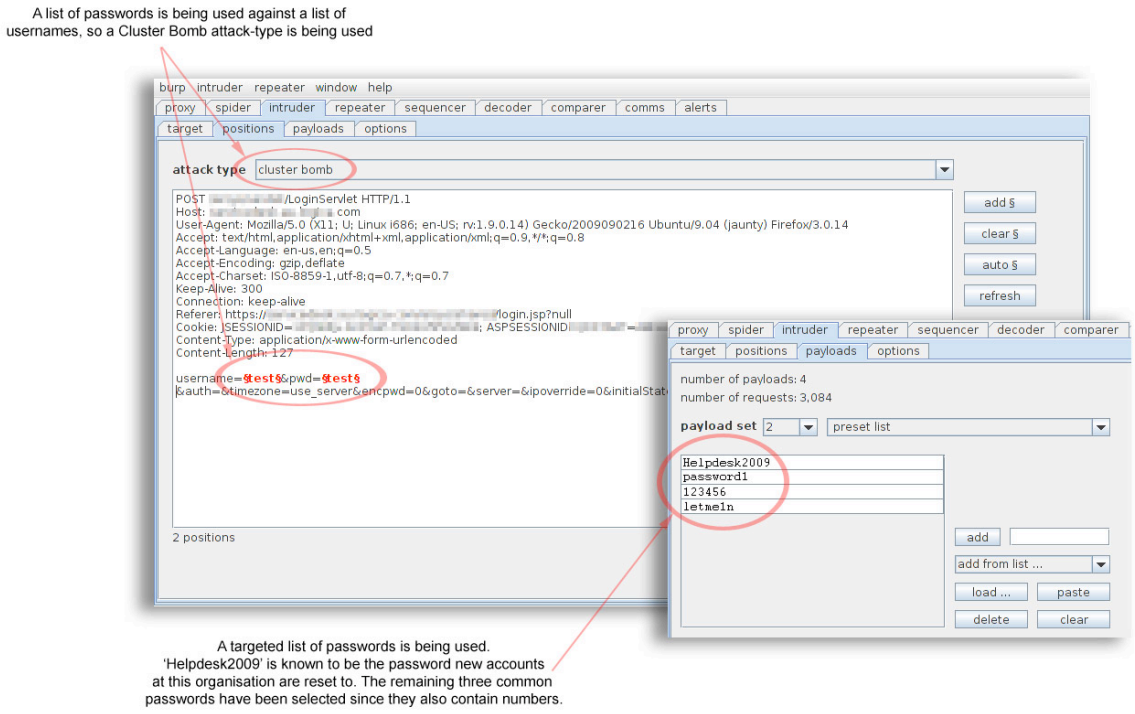


*Figure 32 - Preparing the attack*

If none of these passwords were successful, further searches of social networking sites using the list of e-mail addresses would take place, with a view to compiling a more extensive word list. In this instance however, that wasn't required.

## 11.4. Launching the Attack

An updated attack was launched, with the results appearing as shown in Figure 33. This provided access to an additional 20 accounts; one extra with a password of 'password1', and 19 with a password of 'Helpdesk2009'. In addition to these, there was also another account that responded with a different length to all others, and required

further investigation.

To speed up the testing of accounts and to reduce the volume of network traffic, Burp Intruder was configured not to follow 3xx redirects. This meant that the authentication process was taking place, but the full versions of the resulting web pages weren't being returned. A consequence of this was that the grep function was unable to detect keywords in the responses; in this instance though that wasn't an issue.

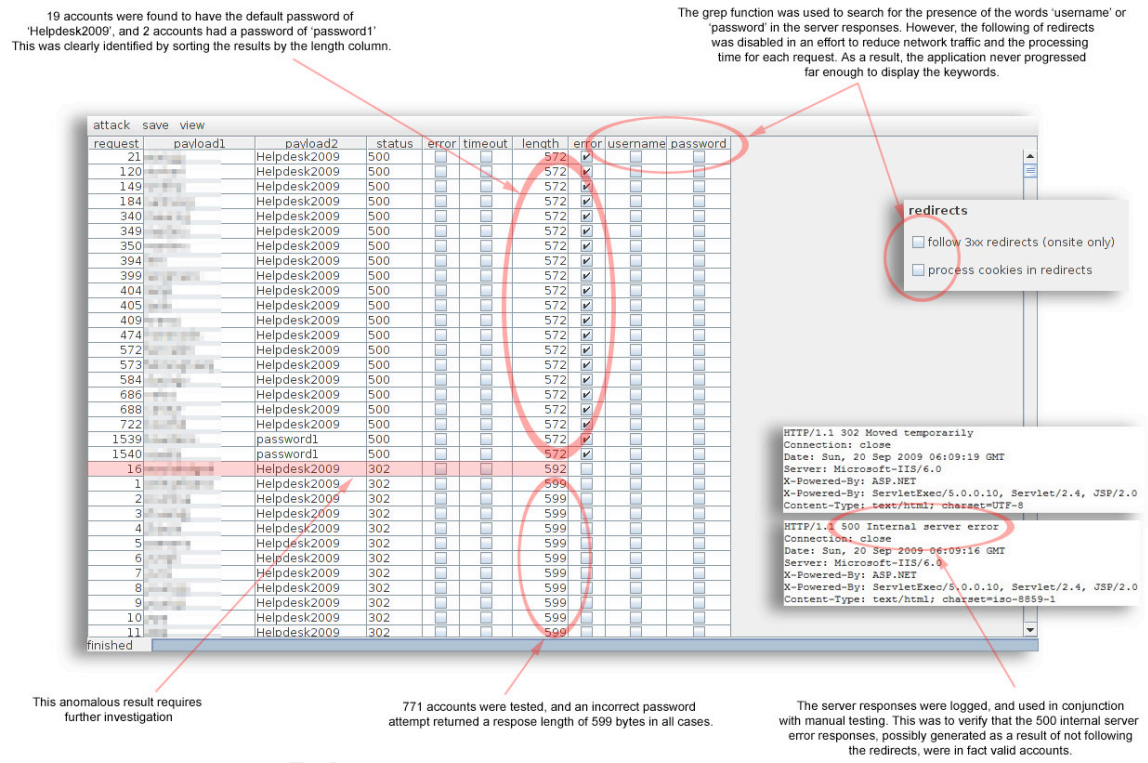


*Figure 33 – Passwords Discovered*

A total of 21 compromised accounts provided enough access to a variety of the portal's systems that further password attempts weren't required at this stage.

## 11.5. Investigating Anomalies

Further investigation into the anomalous result was still required. In a lot of cases this may have been the result of a corrupted request or response, but there's no harm in checking.

As shown in Figure 34, further testing found that this account logged into the portal, but was then presented with another login, which required different credentials.

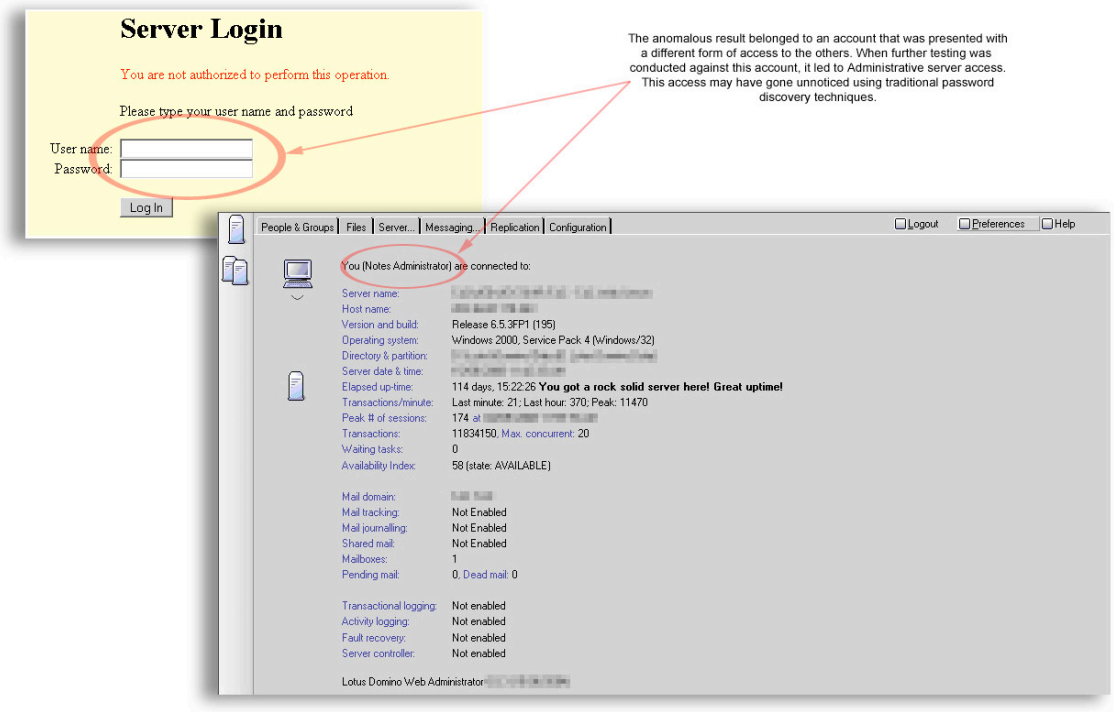


*Figure 34 - Anomaly uncovered*

Another round of testing was conducted on this second portal, following the same methodology as before. Unfortunately for IntruderCorp the same structures and controls were used across both environments, and a correct login was soon discovered. Unlike the rest of the accounts however, this second portal provided access to the administrative console of the first portal, allowing full control of the environment. If a traditional password cracking methodology had been used, this level of access may have gone unnoticed.

## 12. Conclusion

Gaining access to a web application via compromised credentials is as much about the knowledge of the target system and the users of that system as it is about the tools and techniques used to carry out the compromise. Cracking the credentials of an online system has inherent speed limitations when compared to off-line cracking, and so the methodology used needs to cater for that.

Burp Intruder, an HTML fuzzer, can be used to not just crack credentials, but to

discover their format, the pages where they can be used, and to exploit them in a way that minimizes disruption to the target system. This is achieved by looking for differences in the responses returned for valid as opposed to invalid credentials. The primary methods of detecting this are to look for patterns in either the number of bytes returned, a notable difference in the length of time taken to process a particular request, or by looking for keywords.

The human brain is excellent at detecting these patterns and anomalies (Teoh, Soon Tee, 2001) and Burp Intruder's simple but effective attack output leverages off this to make the discovery process so much easier.

This of course continues to be helped by poor password selection techniques. Password usage is subject to the vagaries of human memory, and the need for people to retain an increasingly large number of passwords for a variety of disparate systems. Added to this is the need for an organization to make sure that *all* passwords are maintained at a suitable level *all* of the time, whereas an attacker may only need to compromise *one* account, and they have time on their side to do it. Burp Intruder can be configured to exploit these poor selection techniques through the use of a variety of character, case, number, and complex customized iteration payloads.

A target system can be patched and have unnecessary services removed, and be placed behind firewalls and intrusion detection systems. But it can also still be compromised by a bit of research, a weak password, and Burp Intruder.

## 13. References

Kessler, Gary C. (1996) "Passwords – Strengths and weaknesses." Retrieved from http://www.garykessler.net/library/password.html

Grimes, Roger A. (2006) "MySpace password exploit: Crunching the numbers (and letters)" Retrieved from *InfoWorld*, http://www.infoworld.com/d/security-central/myspace-password-exploit-crunching-numbers-and-letters-983

Schneier, Bruce. (2006) "Real-World Passwords" Retrieved from http://www.schneier.com/blog/archives/2006/12/realworld_passw.html

Brown, Stuart. (2006) "Top 10 Most Common Passwords"
Retrieved from *Modern* Life, http://modernl.com/article/top-10-most-common-passwords

anameless.com. (2003) "Default Admin Username and Password List" Retrieved from http://www.anameless.com/blog/default-passwords.html

Rootsecurity. (2009) "Zero For 0wned Summer Of Hax" Retrieved from http://r00tsecurity.org/files/zf05.txt

OWASP. (2009) "Testing for Default or Guessable User Account (OWASP-AT-003)." Retrieved from http://www.owasp.org/index.php/Testing_for_Default_or_Guessable_User_Account_(OWASP-AT-003)

Web Application Security Consortium. (2005) "Threat Classification – Brute Force." Retrieved from http://www.webappsec.org/projects/threat/classes/brute_force.shtml

Hitachi ID Systems. (2009) "Password Management Best Practices." Retrieved from http://www.psynch.com/docs/password-management-best-practices.html

Fielding, R et al. (1999) "Hypertext transfer Protocol – HTTP/1.1"
Retrieved from *IETF*, http://tools.ietf.org/rfcmarkup?doc=2616#page-57

Braue, David. (2005) "Biometrics: Still searching for a pulse"
Retrieved from *ZDNet Australia*,
http://www.zdnet.com.au/insight/security/soa/Biometrics-Still-searching-for-a-pulse/0,139023764,139187129,00.htm?feed=pt_biometrics

Felker, Mikhael. (2007) "Analysis of FFIEC Guidance: Technologies and Decisions on Authentication" Retrieved from *ISACA*,
http://www.isaca.org/Template.cfm?Section=Home&CONTENTID=46065&TEMPLATE=/ContentManagement/ContentDisplay.cfm

OWASP. (2009) "Fuzzing" Retrieved from http://www.owasp.org/index.php/Fuzzing

OWASP. (2009) "OWASP Testing Guide Version 3" Retrieved from http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf

OWASP. (2009) "HTTP proxying / editing" Retrieved from http://www.owasp.org/index.php/Phoenix/Tools

Password Research Institute. (2006) "Authentication Statistic Index" Retrieved from http://passwordresearch.com/stats/statindex.html

NIST. (1985) "FIPS PUB 112 – Password Usage" Retrieved from http://www.itl.nist.gov/fipspubs/fip112.htm

Burnett, Mark. (2005) "Perfect Passwords: The Top 500 Worst Passwords of All Time" Retrieved from http://www.smashingpasswords.com/top-500-worst-passwords-all-time

Openwall.com. (2009) "Openwall wordlist collection" Retrieved from http://www.openwall.com/passwords/wordlists/

University of Oxford. (2009) "Wordlists" Retrieved from ftp://ftp.ox.ac.uk/pub/wordlists/

Outpost9.com. (2006) "Word Lists" Retrieved from http://www.outpost9.com/files/WordLists.html

Theargon.com. (2007) "Wordlists" Retrieved from http://www.theargon.com/achilles/wordlists/

Packetstormsecurity.org (2009) "Wordlists" Retrieved from http://packetstormsecurity.org/Crackers/wordlists/

Sourceforge.net. (2008) "Kevin's Word List Page" Retrieved from http://wordlist.sourceforge.net/

TechRepublic. (2008) "Chapter 8 – Password Cracking / Brute-force Tools. Reproduced from the book "Anti-Hacker Tool Kit, Third Edition." Copyright © 2006, The McGraw-Hill Companies, Inc." Retrieved from http://techrepublic.com.com/i/tr/downloads/home/0072262877_chapter_8.pdf

Codenomicon. (2009) "The BUZZ on FUZZING" Retrieved from http://www.codenomicon.com/products/buzz-on-fuzzing.shtml

Teoh, Soon Tee. (2001) "Computer Science Department University of California. A Visual Technique for Internet Anomaly Detection." Retrieved from http://www.laas.fr/METROSEC/DOC/CGIM02.pdf

CIRT.net. (2008) "Default Passwords." Retrieved from http://www.cirt.net/passwords

OWASP. (2009) "Testing for user enumeration (OWASP-AT-002)" Retrieved from http://www.owasp.org/index.php/Testing_for_user_enumeration_(OWASP-AT-002)

Askoxford.com. (2009) "What is the frequency of the letters of the alphabet in English?" Retrieved from http://www.askoxford.com/asktheexperts/faq/aboutwords/frequency