Adaptive Time-based Access Control for Privileged Accounts

Building on Break-glass Security, a Banking Perspective

Paul M. Wright

1 Introduction

“Unlimited power is apt to corrupt the minds of those who possess it”

”root is my own account...”
– BOFH 2006

“Time is the master of humanity”
– Rex E. Alford

Break-glass within computing is a term used to describe the act of checking out a system account password for use by a human. It is generally used for highest level system accounts such as root for *nix or SYS/SA for database. These accounts are highly privileged and not in themselves individualized to a specific human, so instead break-glass limits them by the password time duration, with the aim of controlling and reducing the account’s usage to that which is necessary. Break-glass has been examined in a number of publications applied to medical systems. What is currently missing is an accurate translation of original break-glass concepts, especially applied to high security environments such as banking. This paper will provide that along with a description of how break-glass is evolving into a broader method of time-based access control. Finally a proposal for how time-based access control and break-glass can be varied adaptively based on threat level will be exemplified with real-world examples.

This paper is influenced by the Author’s own experience with break-glass systems in Banking security. Whilst conducting the academic review prior to writing this paper it was found that the experience of break-glass systems in banking was not completely in agreement with the current limited number of papers largely based on the medical industry. In order to confirm exactly what break-glass means an etymological study of break-glass origins will be constructive.
2 Origin of Break-Glass, related to modern computing systems

The origin of the term Break-Glass is from publicly accessible fire alarms. It will be useful to use this context to aid in the definition of the term, and compare the evolution of Break-Glass within fire alarms to the evolution of break-glass to modern computer systems, as we may be able to predict the future from previous historic trends.

The first fire alarm networks were installed in Berlin by Siemens in 1951[1][2][3], closely followed by Boston in 1852[4]. The Boston system was based on a telegraphic network of 45 boxes which enabled a local person to electronically signal to the central fire department that there was a fire so they should come and put it out quickly[5]. The ability to set off the publicly situated Boston alarms was through a key bestowed to a small number of local responsible individuals (police officers etc). This was to avoid false alarms by accident or malice. The unfortunate counter effect of this was that in the 1872 Great Fire of Boston, the fire department were delayed by 20 minutes due to a lack of a key-holders to raise the alarm[6] resulting in the deaths of 30 people, to whom this paper is dedicated.

Subsequent to this event, new public fire alarm systems were installed in Glasgow 1878, and in London 1880. The use of a purposefully designed, Break-Glass mechanism, with contemporary documentation is the 1880 system introduced by Charles Bright[7][8][9]. The main advantage of this system being that the ability to “Raise Fire Alarm” privilege could be granted to the public without increasing false alarms to the point of making the system unusable. In other words a publicly available fire alarm system would be less abused due to the break-glass protection.

After success in London and in Glasgow[10][11] with built-in breakglass the US followed with break-glass add-ons in about 1900[5]. Because the US fire alarms had already been installed prior to the use of break-glass a small box had to be added onto the front of the already existing Fire Alarm Boxes as show in the following photos.
Figure 1 Original Gamewell Fire Alarm, subsequent break-glass adaption and Bright’s built-in breakglass.

N.B. Note how the system is basically the same except for the bolted on glass key guard.

From this we learn that from the beginning, Break-Glass has had the following properties.

1. **Shares a single privilege** between many users (the “raise fire alarm privilege”).
2. **Identified initiating individual** by creating loud noise thus calling attention to the initiator and increasing the risk they will be identified and caught in the case of a deliberate false alarm.
3. **Time limiting** the use of the privilege, as the breakglass would be reset after the fire.

The above is important so that we can derive the essence of what break-glass actually means. So for instance it can be said that Break-glass is not a method of escalating privilege – it is a way to reduce abuse of a shared privilege, which is not in itself individualised.

To be of use the break-glass request should be easy to do quickly, not denied, with the emphasis on preserving safety rather than preventative security. Safety in the case of computer systems is analogous to preserving availability i.e. the health of a system. Security questions and ramifications are examined after the break-glass event and punitive measures taken if necessary. This has been described as optimistic security[12], though this should not be mistaken as being a naïve approach – in fact it is pragmatic and in many cases the only way of realistically decreasing risk for system accounts in a large distributed network, where historically the shared system accounts have been allowed to be passed from colleague to colleague and not reset regularly, resulting in administrators no longer employed by the organization, still knowing the password.
3 Academic Review relevant to break-glass and time-based access control (starting from 2000 through to 2012 in chronological sequence)

The earliest dated document describing a functioning break-glass design is Charles Bright’s London network of publicly accessible fire alarms [7]. In the field of computer science the first reference that posits break-glass as a potential idea for computer systems is Povey – 2000[12]. If one digs into commercial practice it is possible to see that the first commercial company to sell a break-glass solution publicly was CyberArk[13] with their EPV product launched in 2003. The Author is aware that prior to CyberArk’s publicly available commercial software there were inhouse solutions within the banking industry that carried out the same functionality. This is to be expected as break-glass is the most practicable method of gaining control of a shared credential for a privileged system account. If an organization cannot limit the account to a single individual i.e. it’s geography, then the other dimension to reduce risk with is to limit the time for which that credential will be effective – hence break-glass and time-based access control. (Banking and Military keep new security innovations non-public for obvious reasons).

Povey’s paper whilst groundbreaking does not fully encapsulate the break-glass notion within the term “optimistic security”. Povey says: “The basic approach of an optimistic security system is to assume that any access is legitimate and should be granted.” More accurately what Break-Glass provides is a method to associate human identity to a system account e.g. root, and to limit the potential insecurity of allowing use of an uncontrollable privilege by limiting how much time it has been used for, by automatically changing the password at the end of the break-glass period, and by warning the user that their actions will be audited, with the threat of punishment if abused. The key feature of a break-glass system is that it automatically resets the password of the account in a set time frame. Typically a checked out root password would be reset within 24 hours. So this is not an “assumption of legitimacy”, as Povey describes, it is a risk limitation exercise given a highly insecure account which has to be used. Having read the small number of papers on break-glass, none of them precisely and fully define either the concept as per the original break-glass concept, or the practice as experienced within financial services - hence the requirement to write this paper. However it is useful to list the papers that intersect this topic which are medical industry based.

The automated reset of an account password based on time is discussed in [14] and called “timely revocation of trust”. Whilst related to break-glass this is different because break-glass does not care if the revocation of the privilege is at an appropriate time, it just does it in 24 hours – whether the user needs the privilege any more or not. If the user gets locked out during their work, they need to break-glass again. Another related concept is authorization based on time of day e.g. Fred is allowed access in the day but not at night. There are a number of these authorization context papers
which are worthy but different from break-glass in that they are dealing with a fixed portion of the day which is a repeated authorized window, rather than a break-glass session which is – "you have 1 hour and then your kicked out the system". Again another related temporal concept is the limit on synchronous privileges. For example a user can only have one role at a time or a role can only have 5 users at a time[18]. This is related in that one of the benefits of break-glass as a time-based access control mechanism is that for that break-glass period there would normally only be one person on that machine at that time. However if a team effort were needed break-glass systems do not prevent the recipient of the password for that time period delegating the password to their colleagues – but at the end of the break-glass time period the password will forcibly automatically change. This concept is named the ‘Emergency Lifetime’ of a privilege in Georgakakis 2011 and is a useful term, applicable to the traditional meaning of break-glass. What is interesting is to see how the use of “Lifetime” of an account credential and access is being transferred from the emergency only scenario to the Business As Usual access (BAU). The idea being that all human access either individualized or through a system account should be on time limited basis by default. Instead of accounts that by default last forever, which is the current position. Accidental non-removal of Leavers is one of the greatest sources of risk in financial services systems.

One of the most recent context-based break-glass papers is Marinovic 2011 which essentially lists rules upon which to either permit or deny break-glass access. Interestingly the scenarios suggested in the paper do not include the scenario of denial. The largest discrepancy between the Author’s understanding of what break-glass is, based on literature review and 15 year’s experience, is that the break-glass request process does not include the ability to deny. That is the point of break-glass. If you have the ability to request i.e. you are a human physically in the vicinity of a break-glass point then you have the right and the privilege and indeed the duty to the break-glass to report a fire – and quickly. That is what break-glass is. A way of enabling access whilst reducing the probability of false alarm. Break-glass secures the safety mechanism by alerting the vicinity to the initiator through the noise of the glass breaking, and the event is time limited as the fire brigade reset the alarm. Or in the case of our IT systems an SA (system administrator) who has the ability to request i.e. is on the list of SAs, quickly checks out the root password because production is about to fall over and then afterwards fills in the incident report and their manager approves the checkout.

The fact that Marinovic-2011 proposes a mechanism for denying break-glass access defeats the object of the mechanism, however, I am not a Medical professional, so there may be ontological differences between the two subject areas that explain this difference, and it would be interesting to develop this conversation in the future as time allows.

On this note, the final paper reviewed was a paper specifically detailing the use of OWL to define Temporal Access Control constraints within ontologies such as medical and financial[19]. OWL is the Web Ontology Language which provides mark up language to define semantic ontological means for different subject domains. For a concise and usefully abbreviated form of OWL please see the Manchester Syntax[20].
Observations on the academic body of knowledge related to commercial banking practice.

The main observation of the current corpus of knowledge on break-glass is that it is entirely Medical Industry based and that the concept of break-glass is one of privilege escalation[21]. From the Author’s experience of information security and break-glass within financial services, break-glass should not really be classified as privilege escalation. An example of privilege escalation is where a user is not allowed root but they manage to get it somehow either through a buffer overflow exploit, replaying a network logon or socially engineering the password from an authorized user. So for instance sudo’ing is not privilege escalation as such – it is a method to carry identity with the already granted root-like privilege (via the sudo’ers file). In break-glass the only control is that the system will identify who has checked out the password, audit and warn them, and forcibly change the value of the password at the end of the session. So really break-glass is just labeling a time segment of a system account’s usage with a specific human. It is matching human activity to system account so that the managers can see who is doing the administration. This then means that accountability, performance pay, discipline and all the quality management metrics can be applied to the human workforce in an intelligent and meaningful manner to improve the organization. In this way management and more specifically Business management i.e. the people paying for the IT systems can enforce control over the IT workforce as they can see from the centralized break-glass server how the main admin accounts are being used. For root this is changing as most work can be done as sudo’d individual users and newer technologies such as Fox’s keon enable fine grained root tasks to be delegated to individualized Unix users. Unix user management is maturing – but there are still tasks that have to be done as root – hence the need for the break-glass system. For database, privilege access control is not so mature, and CyberArk, Oracle (OPAM) and SAP have products which are still experiencing active development i.e. not perfected yet, as will be discussed towards the end of this paper.

In general system access via break-glass has two main trends. The first is for more categories of access to be time limited so that access to OS, DB and even MS Office software is becoming managed via a time-based access control mechanism[22]. The notion of all accounts being time-limited is positive for reducing risk. One of the most common security issues in a system are the open accounts from employees that have left the organization for reasons, such as churn, redundancy or death. The lifecycle of human accounts over time is moving away from the default – “this account lasts forever” stance, to a “this account has to be recertified every year” stance. Considerations for handling digital identity in the case of human death are discussed in this innovative paper published at the same NSPW conference as Povey’s original Optimistic Security paper, referenced at the beginning of our academic review. See http://www.nspw.org/proceedings/2011 [23].

Secondly, the categorization of privileges which in the past may have been recognized as BAU, but are now being moved under break-glass shows a trend towards removal
of human intervention in terms of ongoing system administration and maintenance. This consolidation is the expected result of autonomic computing[24].

Given that this trend towards consolidation and automation is increasing with future software such as the Cloud based 12c database being released by Oracle in 2013, it can be seen that the removal of human intervention will be an increasingly interesting topic, especially from a security point of view. The word Sabotage originates from the introduction of automatic weaving equipment which was deliberately damaged by the employees paid to work the machines. They used their sabots (clogs) to wreck the new looms, that were about to relieve them of their jobs. Business, vendors and security professionals will be mindful to avoid alienating workforces during this consolidation of human resource, and also keen to be able to exert real control over privileged admin accounts – hence this paper.

5 Break-Glass security applied to business scenario

The scenario is a large estate of 10,000 databases in a financial services organization and the vendor platform is Oracle Database on Red Hat Linux.

The key account that would be subject to break-glass on Oracle Database is the SYS account as it is a non-individualised system account with very high privileges. Additionally the SYS account in Oracle is immune to all security controls managed within the database i.e. there is no password complexity verification, history, account locking or failed logon delay (aka connection throttling). The reason for this is that Oracle are very availability focused and wish to avoid the scenario where the administrator is accidentally or purposefully locked out from the server. Unfortunately this also means that the password for SYS could be weak and an attacker may be able to get in. Counter intuitively all the other accounts in Oracle do have proper security controls on them. The reason being that these are less vital to remain unlocked. So we have a situation where the most security sensitive account has the least security controls. This is where are a centralized break-glass server has a role to play, as the break-glass passwords are set externally from the database they can be set to be long random values and verified to be secure. Additionally they can be automatically changed on a regular basis. This means that the main security weakness in Oracle is fixed by the use of a centralized break-glass server such as OPAM or CyberArk EPV.

There are some technical caveats to this.

Firstly the network communication between break-glass server and the databases needs to be encrypted to protect the automated password changes.

Secondly the break-glass server itself needs to be secured, as it contains all the passwords.

Thirdly OS access to the DB server needs to be secured to prevent DB access locally. These are all achievable though not fully realized at this stage, though there are extra security considerations for controlling advanced administrative access.
Importantly, an opportunity for greater efficiency exists, because break-glass time-based access control lends itself to being adaptively varied depending on security level.

6 Adaptive Break-glass

The main objection to break-glass systems, voiced by humans, is that the action of having to break-glass takes too long and slows down emergency response and general day to day administration. This objection tends to gain traction when an estate has never had a security issue in it’s history. The psychology of the human teams involved tends towards laissez-faire security i.e. just enough[25]. Because database estates have low frequency of security events the tendency is to drop the guard. Problem is that if a security event does happen it could well be catastrophic i.e. end of business. Therefore the risk profile is broadly similar to that of a nuclear power station.

If we analogize DB security with personal human security, it would be strange for a human to walk round with their guard up wearing a crash helmet all day when walking the street, having a coffee or sat at their desk. This security posture is inappropriate for day to day living. So why is it that database estates have a single security posture that is set at high all the time. Surely it would be more sensible to adapt security level dependent on the threat level which varies over time. A framework for measuring, controlling and responding to threat level has already been discussed in Mutually Adaptive Databases paper by the Author at http://www.journalofdatabasesecurity.com/ [26]. All that needs to be done is for the break-glass mechanism to be integrated into that adaptivity mechanism, so that the length of a break-glass session is varied depending on the measured level of threat. For instance, normally a break-glass ticket could last 1 day, but if there were a lot of failed logons detected that could automatically shorten to one hour, thus increasing security when it is needed. This would be of great business benefit to the adopter as they could have more efficient systems when it was safe to do so.

The other big objection to break-glass systems is that they are usually separate servers managed by separate teams often on different platforms (e.g. MS Windows/AD) and therefore not fully trusted by the Unix/Oracle team both in political and reliability terms. The future for break-glass technology is to build in the break-glass authorization mechanism to the database itself, like Bright’s Fire posts in London[9].
Coded example of an adaptive break-glass system implemented on a commonly used relational database (Oracle on Linux).

### 7.1 New Cron job to call once a day

```bash
#!/bin/sh
export ORACLE_SID=orcl
export PATH=/home/oracle/app/oracle/product/12.1.0/dbhome_1/bin
export ORACLE_HOME=/home/oracle/app/oracle/product/12.1.0/dbhome_1
export TNS_ADMIN=/home/oracle/app/oracle/product/12.1.0/dbhome_1/bin
sqlplus -s /nolog <<EOF
    set heading off
    set feedback off
    conn sys/lowsec@192.168.0.4/orcl_plug as sysdba
    exec
    begin EXECUTE IMMEDIATE 'alter user sys identified by values ''break_glass_unbroken''';
    end;
    /
    exit;
EOF

"~/cronacle_breakglass.sh"
```

--oracle crontab using crontab -e

```bash
0 0 * * * /cronacle_breakglass.sh
0 0 * * * /home/oracle/bginteg/bg_integ.sh
```

### 7.2 Break-glass procedure and role to group those allowed to break-glass

```sql
Create role break_glass_role;
Grant execute on break_glass_proc to break_glass_sysdba_role;
```

```sql
create or replace procedure break_glass_to_sys
is
    pwd varchar2(10);
begin
    pwd := dbms_random.string('p',10);
    execute immediate 'alter user SYS identified by '''||pwd||'''';
    dbms_output.put_line('You have broken glass to SYS - This will be recorded and your actions audited => new SYS password: ''||pwd||''');
end;
/
```
SQL> create or replace procedure break_glass_to_sys
2  is
3    pwd varchar2(10);
4  begin
5    pwd := dbms_random.string('p',10);
6    execute immediate 'alter user SYS identified by ''|pwd|''';
7    dbms_output.put_line('You have broken glass to SYS - This will be recorded and your actions audited => new SYS password: '||pwd);
8  end;
9  /
Procedure created.

SQL> exec break_glass_to_sys();
You have broken glass to SYS - This will be recorded and your actions audited => new SYS password: @vQS(@wnX$

PL/SQL procedure successfully completed.

SQL> conn sys/"@vQS(@wnX$"orcl as sysdba
Connected.
SQL> create role break_glass_role;
Role created.
SQL> grant execute on break_glass_to_sys to break_glass_role;
Grant succeeded.
SQL> grant break_glass_role to scott;
Grant succeeded.
SQL> conn scott/lowsec
Connected.

SQL> exec sys.break_glass_to_sys();
You have broken glass to SYS - This will be recorded and your actions audited => new SYS password: $#VEOC#M_(

PL/SQL procedure successfully completed.

Scott can now break-glass to SYS and his actions will be audited whilst doing so as part of the SYSDBA audit trail (Note that the above is for 11g and will not work as-is for 12c).
7.3 Adaptive break-glass procedure

Now let’s make the break-glass adaptive by lowering the break-glass session from a day to 1 hour IF there are a large number of failed logons in the audit trail - suggesting that someone is trying to break-in. We will build upon the same adaptive procedure used in Mutually Adaptive Databases paper[26], shown below with an extra line added to the sys_throttle_switch procedure.

```sql
--create the procedure to read the syslog and choose relative delay time.
create or replace procedure sys_throttle_switch IS
  l_dir VARCHAR2(10) := 'SYSLOG_DIR';
  l_filename VARCHAR2(25) := '1017.txt';
  b_exists BOOLEAN;
  n_length NUMBER;
  n_bsize NUMBER;
begin
  UTL_FILE.FGetAttr(l_dir, l_filename, b_exists, n_length, n_bsize);
  IF ( b_exists ) THEN
    IF ( n_length > 23011) THEN
      DBMS_OUTPUT.Put_Line('Greater than 100 failed sys logons in file '||n_length);
      EXECUTE IMMEDIATE 'ALTER TRIGGER systhrottle.tra_servererror_ora1017_10 ENABLE';
      EXECUTE IMMEDIATE 'ALTER TRIGGER systhrottle.tra_servererror_ora1017 DISABLE';
      EXECUTE IMMEDIATE 'alter user sys identified by values ''break_glass_unbroken''';
      --impossible password set to "put glass" over SYS account
    ELSEIF (n_length < 1) THEN
      DBMS_OUTPUT.Put_Line('Zero failed logins recorded'||n_length);
      EXECUTE IMMEDIATE 'ALTER TRIGGER systhrottle.tra_servererror_ora1017_10 DISABLE';
      EXECUTE IMMEDIATE 'ALTER TRIGGER systhrottle.tra_servererror_ora1017 DISABLE';
    ELSE
      DBMS_OUTPUT.Put_Line('There have been less than 100 failed SYS logons but more than zero');
      EXECUTE IMMEDIATE 'ALTER TRIGGER systhrottle.tra_servererror_ora1017 ENABLE';
      EXECUTE IMMEDIATE 'ALTER TRIGGER systhrottle.tra_servererror_ora1017_10 DISABLE';
    END IF;
  ELSE
    DBMS_OUTPUT.Put_Line('There have been no failed logons recorded for SYS');
    EXECUTE IMMEDIATE 'ALTER TRIGGER systhrottle.tra_servererror_ora1017_10 DISABLE';
    EXECUTE IMMEDIATE 'ALTER TRIGGER systhrottle.tra_servererror_ora1017 DISABLE';
  END IF;
END;/
```

Now if there are more than 100 failed logons per hour the break-glass period will shorten to 1 hour which will recycle the password more frequently thus denying a brute force attack.
8 Conclusion

This paper has consolidated the definition of break-glass access control using the historical meaning from fire-alarms extended to contemporary banking security and drawn comparisons to how each technology evolves to having break-glass built-in. Then this paper proposed break-glass as control that could be responsive to threat level. An example using a commonly used relational database was demonstrated. Hopefully this paper will contribute to understanding future directions for access control.

9 Future work

Break-glass access control depends on the physical ability to turn off the session at the end of the break-glass period. There are technical difficulties with doing this securely when dealing with a knowledgeable administrator who wants to bypass access control. In order to increase the security of the break-glass control an integrity check would be required at the beginning and end of the break-glass session to verify that the control had not been bypassed. This will be the subject of the third paper in the series to be published privately.

10 References


