SIEGE: Service-Independent Enterprise-GradE protection against password scans

Marcel Waldvogel Jürgen Kollek

Computer Science Department
University of Konstanz
Universitätsstr. 10, 78457 Konstanz, Germany
<first>.<last>@uni-konstanz.de

Abstract: Security is one of the main challenges today, complicated significantly by the heterogeneous and open academic networks with thousands of different applications. Botnet-based brute-force password scans are common security threat against the open academic networks. Common defenses are hard to maintain, error-prone and do not reliably discriminate between user error and coordinated attack. In this paper, we present a novel approach, which allows to secure many network services at once. By combining in-app tracking, local and global crowdsourcing, geographic information, and probabilistic user-bot distinction through differential password analysis, our PAM-based detection module can provide higher accuracy and faster blocking of botnets. In the future, we aim to make the mechanism even more generic and thus provide a distributed defense against one of the strongest threats against our infrastructure.

1 Introduction

Account information is in high demand among crooks. Hacked accounts are used (1) as stepping stones to hide your tracks, (2) to abuse legitimate web sites for phishing and distribution of malware and copyrighted material, (3) to send spam without being filtered, (4) for installing keyloggers to obtain more data, including banking information, and (5) to add machines to global botnets for later automated large-scale malfeasance.

One of the favorite uses of botnets is to farm more accounts to increase the power (and rental value) of the botnet [Pur03]. This is generally done by trying to authenticate against a service with (username, password) tuples based on dictionaries. These dictionaries are frequently enhanced by formation rules to e.g. append a few digits to words or replace the letter *e* by the digit *3*.

In the beginning, these (username, password) pairs were fired at the target as quickly as possible. However, tools such as DenyHosts [Den] and Fail2ban [Fai] made it easy to defend against these types of attacks by blacklisting the IP address of the assailant.

Nowadays, the attacks happen much more subtly. A large set of bots, often several hundred thousand, jointly and covertly go stalking potential victims: Each of them only uses a few shy attempts, hoping this will not alert the subject. Later, another member of the hunting party will tickle the prey, while the first attacker approaches the next target.

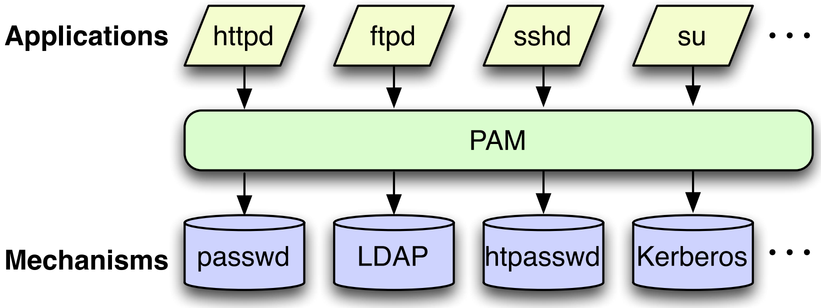


Figure 1: Flexibility of Pluggable Authentication Modules (PAM)

As it is necessary to allow a user to mistype her password or switch to the correct password for the system, a single failed attempt cannot be used as a direct indication of an attack. Identifying user-generated typos helps distinguishing between botnet attacks and failed user logins.

Our PAM[SS95]-based approach includes several firsts:

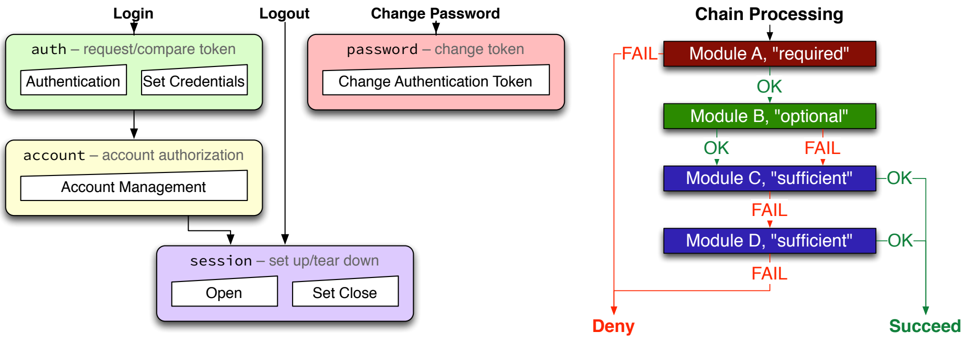
1. Retaining the power of PAM, allowing arbitrary authentication and authorization mechanisms to be used (Section 3)
2. Allowing geographical risk differentiation (GRID, Section 4.2)
3. Hierarchical risk grouping (Section 4.2)
4. Coordinated organisation-wide defense (CROWD, Section 4.1)
5. Dictionary-attack defense (Section 4.3)
6. Secure typo detection for user-bot differentiation (Section 4.4)

Based on flexible, realtime in-app tracking and combined with the use of DNS-based black lists, SIEGE provides low-maintenance, high-impact defense against password-guessing attacks.

2 Related work

2.1 Authentication

Around 1990, it dawned to administrators and system programmers that traditional `/etc/passwd`-based login approach was not flexible enough. Kerberos [KN03], LDAP [WHK97] and derivatives such as Microsoft's Active Directory started coming of age in the following



(a) PAM facilities

(b) Requirements in the module chain

Figure 2: PAM architecture

decade. The Open Software Foundation came up with Pluggable Authentication Modules (PAM) [SS95] as the way of making authentication more flexible (Figure 1).

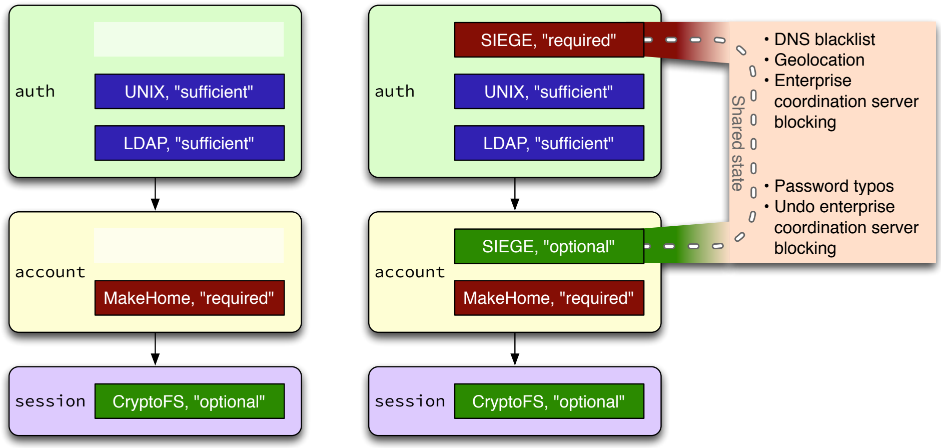
For each application, the system administrator can individually specify which authentication mechanisms to allow as part of the module chain (Figure 2 (b)). In addition to authentication, other account-specific functions, session management and change of authentication token (mostly passwords) can be controlled (Figure 2 (a)). Common functions include limiting access by date and time; creating and populating home directories on the first login; mounting special volumes such as encrypted homes (and securely unmounting them on logout); and many more (Figure 3).

Due to its versatility, PAM has rightly become the de-facto authentication mechanism for most Unix-like desktop and server systems [Mor97].

2.2 Attack/defense

For more than a decade, brute force account/password guessing attacks against SSH (and other services requiring authentication) have been going on [Cid13]. In the early days, a single host attacked its target with a large set of login attempts. This was simple to mitigate: Specialized host-based intrusion detection systems (IDS) like Fail2ban or DenyHosts [Fai, Den] started scanning log files for IP addresses which exceeded a given failed login rate threshold. The culprits were blocked from further attempts for a specified length of time.

Even though these are still not standard practice when setting up new hosts, the attackers moved away from individual source hosts to entire botnets. This helped them also to cover their tracks. Nowadays, each of the several ten or hundred thousands members of a botnet only probe a small set of username/password combinations before focusing on the next victim. This renders source-based limits essentially useless. Furthermore, log file analysis has several drawbacks: It must be flexible enough to cope with changes of the



(a) Unprotected PAM configuration

(b) PAM with SIEGE protection

Figure 3: PAM protection

authentication software, but it must not be fooled into misinterpreting log entries.

For example, an ssh error message documenting a failed attempt for user test might look as follows

Jan 08 13:39:55 testhost sshd[555]: Invalid user test from 111.222.33.44 .

Malicious users can try to lock out users from a different address, say, 22.33.44.55, by trying to authenticate with the non-existant user test from 22.33.44.55 . The resulting message

Jan 08 13:39:59 testhost sshd[559]: Invalid user test from 22.33.44.55 from 111.222.33.44

and its relatives were misinterpreted by many earlier versions of most software packages, as they did not have the benefit of the differing background colors.

pam_abl [ABL] provides a similar function, but implements it as part of the PAM chain. Therefore, it does not need to parse log files and thus is not vulnerable to the above log injection attacks. However, a host with pam_abl still fights alone in a steep uphill battle against a fierce horde of coordinated botnet members.

In more restricted environments (stronger rules, fewer services, or fewer users) than in academia, it is also possible to limit the hosts which may login or use certificate-based logins (including ssh keys). However, the diverse student and staff body of most universities makes the necessary education and support problems look like an unsurmountable problem, especially as this diverse group of 'locals,' together with many academic guests, make traditional security perimeters impossible: *inside* is the same as *outside*.

3 Architecture

SIEGE brackets the actual authentication modules (Figure 3 (b)).[1] With this trick, it can remain mechanism-agnostic while learning as much as possible about successful and failed authentications: A successful authentication will be visible to both module parts, while a failed attempt will only call the top half.

The functions are divided into the top and bottom halves as follows.

Top half

1. Check whether the remote host is known to third-party DNS-based blacklists, such as Spamhaus XBL [Spa]. Fail immediately on match.
2. Compare the password against a dictionary of common passwords (Section 4.3)
3. Determine remaining number of failed attempts for this address:
   (a) Look up source address in the CROWD (CooRdinaated Organization-WiDe, Section 4.1) server. If unreachable, use local cache. Return this information, if any. Otherwise:
   (b) Determine geographic origin of the connection. Use administrator-set likelihood for legitimate login attempts modified with optional user preferences to determine maximum number of login attempts (Section 4.2).
4. Decrease the number of remaining login attempts, indicating that a (so far unsuccessful) attempt is in progress. The amount of reduction is tuned by the likelihood of a user password mistake (Section 4.4).
5. Perform hierarchical blocking propagation (Section 4.2).
6. Store this information in the local cache and update the CROWD.

To simplify the process of writing back local results if the CROWD has been temporarily unavailable, the minimum of the remaining attempts of CROWD and the local cache can be used.

Bottom half

1. Undo the attempted reduction and update cache and CROWD accordingly.

This unique three-level approach of (1) global, long-term DNS blacklist, (2) organization-wide CROWD, and (3) local system information, is able to strike an excellent balance between security, manageability, control, and privacy. Details about the implementation can be found in [Kol13].

[1] In a future version, we might consider bracketing it inside the authentication facility, between the *Authentication* and *Set Credentials* functions, similar to `pam_abl`

4 Defense mechanisms

The challenge in the defense against password-guessing attacks is finding a balance between

1. locking out as many of the attackers as possible, to reduce their guessing rate as much as possible, and
2. allowing legitimate users to log in even when an attack is ongoing.

Our approach at achieving this is described in the remainder of this section.

4.1 Coordinated organization-wide defense

In the spirit of Indra [JWZ03], CROWD uses the wisdom of the (cooperating) masses within an organization to coordinate their defense against the seemingly unsurpassable superiority of the botnet. Each host contains a local secret, which is used to authenticate against the central CROWD server through a secure connection.

The race conditions in the protocol are hard to exploit, as the problem windows are very short-lived. This approach still allows the clients to remain in full control, keep the server simple (get, put, expire) and close gaping accounting loopholes in other approaches. For example, `pam_abl` (and, depending on the configuration, also the other mechanisms), allow a single-source attacker to open a large number of connections simultaneously. When they are open and ready to receive the password, hundreds or thousands of parallel guesses can be made, before the source is blacklisted. Due to the pre-decrementing used in SIEGE, even with some decrements missing due to the race condition, only the first few attempts will be evaluated before the source is stopped.

4.2 Geographical risk differentiation

Even at a university with global outreach, the services requiring authentication have a local user base: Most requests will be from within the campus, the city, state, or country. Therefore, remote locations are less likely to be the source of legitimate logins, making password mismatches from other locations more likely to be from attackers. As a result, the threshold to block a given source for exceeding the number of bad password attempts can be set smaller while maintaining a high probability that legitimate users will not be locked out.

SIEGE uses this property to differentiate failure thresholds and their effects. A sample set of parameters is given in Table 1: For the university's home country, after 10 failed attempts within a defined period (typically 10 minutes), the host is blocked for a configurable time (10 minutes as well). As soon as 10 hosts within the same subnet (/24 in the case of IPv4) are blocked, the entire subnet is blocked for 20 minutes. When the blocked

	Failure thresholds				Blocking periods (minutes)			
Class	Host	Subnet	Net	Country	Host	Subnet	Net	Country
Home	10	10	10	∞	10	20	30	—
Neighbors	5	5	5	20	10	20	30	60
Others	2	2	2	10	10	20	30	60

Table 1: Example escalation parameters for password failures

subnets in a /16 network reaches 10, the entire /16 network is blocked for 30 minutes. Even when many networks in the home country are blocked, the country itself is never completely blocked. For neighbor and other countries, entire countries can be blocked.

This mechanism can be too aggressive for a member of the university currently visiting a country where a large number of bots are currently operating. Therefore, we allow individual users to list countries they are visiting.[2] For those users, the login threshold will be set to 2 (or the minimum of the country) and they will be exempt from network size escalation.

This allows the trade the small risk of that user's account being attacked against the convenience of the user. Because this has to be a user-driven step, the user can also be educated about security precautions at this time.

With this adaptive combination of reactions, the GRID component of SIEGE can ensure a very high level of security without compromising user experience.

4.3 Dictionary-based botnet identification

As a rule of thumb (and often as part of the security policy), passwords should not be derived from dictionary words or well-known frequently-used passwords. This can be enforced by the CrackLib PAM module [Gaf] or other similar tools. Conversely, this also means that a user is unlikely to use a dictionary word as their password. SIEGE includes a simple test whether a password tried is known to CrackLib and immediately classifies a host trying to use such a word as an attacker, putting it on a temporary blacklist. This significantly increases the efficiency of the defense perimeter.

4.4 User-/bot password differentiation

For processes supporting manual password entry, a user has a chance of mistyping the password. Common typing mistakes include:

Transposing two keys on the keyboard: Two adjacent characters in the password are swapped.

[2] This is best done before leaving the home country, to avoid a lockout in the case of an acute attack.

Duplicating. Bouncing a key on the keyboard: A character is in the password twice instead of once.

Missing. Not hitting a key strong enough: The resulting password is missing a character.

It is unlikely that an attacker will try the correct password modified by one of these rules early in the process. Thus, seeing a password modified by one of these rules is likely to come from a user. When we detect this, the try will not count as much as a full incorrect password attempt, thus reducing the chance of a legitimate user being locked out incorrectly.

As the above error-generating processes can all be reversed, this is easy, even when the passwords are encrypted (which they should be):

Transposing. All possible single transpositions can be tried in $L-1$ password encryption attempts locally on the machine, where L is the length of the password just received.

Duplicating. All sequences of duplicate successive characters can be eliminated each in D password encryption attempts locally on the machine, where D is the number of duplicate successive characters ($D < L$).

Missing. All possible missing characters can be tried in $(L+1)*A$ password encryption attempts locally on the machine, where A is the size of the likely alphabet.

The first two mechanisms are very cheap, the third can be rather costly if an expensive encryption function was chosen for the password, so in some environments, this costly step might be undesirable and can thus be deactivated.

Mechanisms which perform weak matches against the password open up new attack vectors. One prominent mechanism are timing attacks, where a shorter password verification time would indicate that the correct password has been found after only a subset of the operations. To thwart such timing attacks, all the configured operations must be performed for an incorrect input password. As the number of password verification options then only depend on the input password, the timing leaks no information about the proximity of the input to the actual password.

As the weak matching does not directly return information whether the password was a close match or not, this information is very hard to abuse. Therefore, we believe that this option only strengthens the defense, not weakens it, as it allows for significantly narrower limits before blacklisting.

5 Evaluation

We built a simulator of the GRID and hierarchical blocking propagation logic to show the large-scale effects (Table 2). For this simulation, we assumed three botnet sizes, ranging from 10 thousand to 100 million hosts. The botnets are following either of two goals: A targeted attack at a single host, or a sweeping attack, which ‘just’ tries to collect accounts,

Target hosts	single			1k		
Botnet hosts	10k	1,000k	100,000k	10k	1,000k	100,000k
Unprotected	10k	1,000k	100,000k	10,000k	1,000,000k	100,000,000k
Fail2ban/DenyHosts	0.2k	16k	1,630k	163k	16,300k	1,630,000k
SIEGE, 1 home	0.0k	0.1k	5k	0.0k	0.1k	5k
SIEGE, 5 neighbors	0.0k	0.2k	**0.5k**	0.0k	0.2k	**0.5k**
SIEGE, 250 others	0.0k	**0.4k**	**0.4k**	0.0k	**0.4k**	**0.4k**
SIEGE, total	0.0k	0.7k	6.5k	0.0k	0.7k	6.5k

Table 2: Password trial rate [s^{-1}] (bold numbers indicate country blocking)

which has identified 1,000 hosts on the campus. We also estimate that due to protocol overhead and built-in backoff timers, each bot can try a single password per second per target host.

For the unprotected hosts, this means that the total number of password attempts is the product of botnet size and target hosts, resulting in up to stunning $10^1$1 password attempts per second!

Fail2ban and DenyHosts are assumed to block a host after 10 failures for the following 10 minutes, so the effective trial rate is 10 attempts every 610 s, improving on the unprotected hosts by a factor of 61.

SIEGE uses the parameters from Table 1. We assume that there are 256 countries, one home country, five neighbors, and the remaining 250 in the ‘other’ class. Each country consists of 256 /16 nets, subdivided into /24s. Attacking hosts are uniformly distributed about the resulting 2^{32} addresses. This is actually the worst case for SIEGE, as any clustering will greatly increase the impact of the hierarchical blocking. We also assume that the attackers will carefully avoid all passwords found in the targets’ dictionary, to avoid multiplying SIEGE blocking rate (and thus significantly reducing attack rate).

The results are impressive: SIEGE limits background-type scans to a few dozen password attempts per second. But even under heavy attack, no more than 7k attempts are successful, while still offering service to most legitimate users. GRID and CROWD combined offer 7 orders of magnitude protection over vanilla systems and more than 5 magnitudes compared to Fail2ban/DenyHosts-style.

6 Conclusions

Even a small campus without any specific protection can be subject to roughly $10^1$7 password scans per day, especially on the weekend, when monitoring is not as close. This is equivalent to trying *all(!)* twelve-digit lowercase passwords, ten-digit alphabetic passwords or nine-digit passwords perfectly chosen from all 95 ASCII printable characters. With a large number of unprotected hosts, you no longer should assume that even excellent passwords are safe. Therefore, alternate mechanisms are needed. GRID and CROWD mechanisms of SIEGE provide several orders of magnitude improvement to current approaches. When you know that your campus is free of dictionary-derived passwords, the attacker identification can be significantly sped up, providing even stronger protection.

We are currently working on putting the code into a releasable format to make the SIEGE benefits available to the community.

References

[ABL] PAM ABL: Automatic defense against brute force attacks. http://pam-abl.sourceforge.net, accessed 2013-01-08.

[Cid13] Daniel Cid. SSH brute force – The 10 year old attack that still persists. Sucuri Blog, July 2013. http://blog.sucuri.net/2013/07/ssh-brute-force-the-10-year-old-attack-that-still-persists.html, accessed 2013-01-08.

[Den] DenyHosts. http://denyhosts.sourceforge.net.

[Fai] Fail2ban. http://www.fail2ban.org.

[Gaf] Cristian Gafton. pam_cracklib - checks the password against dictionary words. http://www.linux-pam.org/Linux-PAM-html/sag-pam_cracklib.html, accessed 2013-01-08.

[JWZ03] Ramaprabhu Janakiraman, Marcel Waldvogel, and Qi Zhang. Indra: A Peer-to-Peer Approach to Network Intrusion Detection and Prevention. In *Proceedings of IEEE WETICE 2003*, Linz, Austria, 2003.

[KN03] J. Kohl and C. Neumann. The Kerberos Network Authentication Service (V5). RFC 1510, Internet Engineering Task Force (IETF), September 2003. http://tools.ietf.org/html/rfc1510.

[Kol13] Jürgen Kollek. Ein verbesserter PAM-basierter Ansatz um “brute force”-Passwort-Angriffe auf den “secure shell”-Service zu erkennen und zu verhindern. Bachelor’s thesis, University of Konstanz, 2013. http://kops.ub.uni-konstanz.de/handle/urn:nbn:de:bsz:352-259480.

[Mor97] Andrew G. Morgan. Pluggable Authentication Modules for Linux. *Linux Journal*, December 1997.

[Pur03] Ramneek Puri. Bots & Botnet: An Overview. SANS Institute InfoSec Reading Room, August 2003. http://www.sans.org/reading-room/whitepapers/malicious/bots-botnet-overview-1299, accessed 2013-01-08.

[Spa] Spamhaus. XBL – Exploit and Botnet filter. http://www.spamhaus.org/xbl/.

[SS95] V. Samar and R. Schemers. Unified Login with Pluggable Authentication Modules (PAM). Request for Comments 86.0, Open Software Foundation, October 1995. Archived at http://www.kernel.org/pub/linux/libs/pam/pre/doc/rfc86.0.txt.gz, accessed 2013-01-08.

[WHK97] M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3). RFC 2251, Internet Engineering Task Force (IETF), December 1997. http://tools.ietf.org/html/rfc2251.