Supplementary Material

Multiview: Finding Blind Spots in Access-Deny Issues Diagnosis

Bingyu Shen* Tianyi Shan* Yuanyuan Zhou
University of California, San Diego

1 Implementation Details

We discussed the general implementation methodology for all applications in our main paper. This supplementary material provides additional details on implementation efforts for each individual applications in a point-to-point basis for all three stages, including toggle analysis (Section 1.1), delta generation (Section 1.2) and change impact analysis (Section 1.3).

1.1 Implementation for toggle analysis

Multiview requires APIs to be implemented for each software to perform the toggle analysis at different levels, including components, objects and specific rules. We discuss the detailed implementation at each level as follows.

Toggle analysis at components level . Toggle analysis has the assumption that each component's access-control is independent of other components. In a common web service structure, the access control in common applications are independent from each other (e.g., file systems, web servers, databases). Each component's access control result is a black box to other components to achieve better modularity and isolation. Therefore, Multiview treats each application as one single component and implements the function to turn off the access control in each application as follows.

1. File system. Multiview grants 777 recursively on the accessed objects to allow every user access to the file.
2. Apache. Multiview replaces the access-control configuration blocks with a configuration block matching from root directory with `Require all granted` to allow all requests.
3. Nginx. Multiview replaces the configuration blocks with one location block matching all requests with only `return 200;` to allow all requests.
4. Vsftpd. Multiview removes all the access-control configuration entries in the file (the default behavior is allow).
5. Proftpd. Multiview replaces all the access-control configuration blocks with a `Directory` block matching all file paths with only `AllowAll` directive.
6. MySQL. Multiview executes `GRANT ALL PRIVILEGES on *.* to user` to allow user access to all databases and all data.
7. PostgreSQL. Multiview executes `GRANT ALL` to the denied user at database, schema and table (or other objects) to grant all privileges.
8. MongoDB. Multiview grants `root` role to the denied user to allow all database read/write and administration operations.
9. Squid. Multiview removes all the access-control configuration entries in the config file and add `http_access allow all` to the beginning of file.

In summary, Multiview requires developers to implement methods to toggle the protection of one component to test whether the access denial is related to this component. The implementation idea usually involves two parts (1) temporarily disable the protection and (2) grant the highest privilege to the denied user. While this requires domain expertise for each application, the implementation is usually simple and is a one-time effort.

Toggle analysis at object level . The access-control rules in the applications are often associated with the objects. For example, (1) the file permissions are associated with files; (2) the web server and FTP configuration are based on files or directories;(3) in databases, the permissions are associated with the tables. As such, the access-control rules may be tested separately to identify which object cause denial.

Multiview performs toggle analysis in two steps. First, Multiview extracts all the relevant objects based on the characteristics of the denied object. Second, Multiview uses the toggle analysis to change the access-control rules of each object to identify the object that lacks permissions. The implementation details are as follows.

1. File system. From the denied object, Multiview first collects the object and all upper level directories as relevant objects. Second, for each object, Multiview grants `777` to all the other objects to see if the object is related to denial.
2. Apache. Multiview treats all configuration blocks (e.g., `<Directory>`) as relevant objects as it is hard to simulate

*Co-first authors

the regex matching process. Second, for each configuration block, Multiview adds a directive `Require all granted` to all the other blocks to relax the access control in other blocks, to see if the configuration block is related to denial.

3. Nginx. Multiview treats all configuration blocks (e.g., `<location>`) as relevant objects as it is hard to simulate the matching process. Second, for each configuration block, Multiview adds a directive `return 200;` to the beginning of all the other blocks to relax the access control in other blocks, to see if the configuration block is related to denial.

4. Vsftpd. The vsftpd configuration file is a list of configuration entries with values. Multiview directly perform toggle analysis at rule level.

5. Proftpd. The configuration rule in Proftpd is similar to Apache. Multiview treats all configuration blocks (e.g., `<Directory>`) as relevant objects. Second, for each configuration block, Multiview adds a directive `AllowAll` to all the other blocks to relax the access control in other blocks.

6. MySQL. From the denied object, Multiview first collects the database and the denied object as relevant objects. Second, for each object, Multiview grants `ALL` to all the other objects to see if the request lacks permission at which level.

7. PostgreSQL. From the denied object, Multiview first collects the database, schema, and table (or other object) as relevant objects. Second, for each object, Multiview grants `ALL` to all the other objects to see if the request lacks permission at which level.

8. MongoDB. From the denied object, Multiview first collects the cluster, database and collection as relevant objects. Second, for each object, Multiview grants all privileges to all the other objects to see if the request lacks permission at which level.

9. Squid. The Squid configuration file is a list of configuration entries with values. Multiview directly perform toggle analysis at rule level.

In summary, Multiview requires developers to implement methods to toggle the access control of relevant objects to identify the objects causing the denial. To avoid complex implementation of specific matching process, Multiview applies toggle analysis to find relevant options. The implementation idea is to test each object' access control by granting all the other objects with sufficient privileges. These mutation logics for the objects can be implemented as a one-time effort to handle common syntax.

Toggle analysis at rule level . After identifying the objects that user lacks proper permissions, Multiview performs toggle analysis of the rules associated with the objects to understand each rule is allowed or denied. The results of each rule can be used to guide the rule mutation and combination in delta generation.

We detail the toggle analysis for individual access-control rules in different applications as follows. The idea is to only isolate one single rule and replay it to get the access-control result for this rule.

1. FS. For file system, each object (file or directory) has read, write, and execute permissions for three different roles (owner, group, other). Multiview utilize `stat` and `test` command to get the subject's permissions on the object.
2. Apache. To get the access-control result for one rule inside the configuration block, Multiview removes all the other access-control rules and combination operators, to keep one rule. The web request's replay result is the rule's access-control result.
3. Nginx. Similar to the toggle analysis in Apache, Multiview removes all the other access-control rules and combination operators, to keep one rule. The web request's replay result is the rule's access-control result.
4. Vsftpd. Since Vsftpd's configuration file is a list of configuration entries with values, Multiview removes all the other access-control rules and only keep one rule. The FTP request's replay result is the rule's access-control result.
5. Proftpd. Proftpd's configuration format is similar to Apache. Multiview removes all the other access-control rules and combination operators, to keep one rule. The FTP request's replay result is the rule's access-control result.
6. MySQL. Multiview uses SQL commands to query the privilege table for the denied DB user to get the privilege status on database and denied object (e.g., table or stored procedure).
7. PostgreSQL. Multiview uses predefined SQL functions (e.g., `has_database_privilege`.) to test the denied DB user's privilege status on the database, schema and table .
8. MongoDB. MongoDB manage the permission status by granting roles to the user. Multiview collects the access-control rules by collecting the privileges on all the roles granted to the denied user, as well as each role's inherited privileges.
9. Squid. Squid's configuration is a list of configuration entries with first match wins. Therefore, Multiview removes all the other access-control rules and only keep one rule. The replay result is the rule's access-control result.

1.2 Implementation for delta generation

With toggle analysis, Multiview finds objects for which the user does not have enough permission. Delta generation help sysadmins find possible directions by (1) relaxing access-control rules to generate rule mutations and (2) combining them together with combination operators. For the first step, Multiview prune mutations with excessive permissions with a predefined partial security order. We discuss the implementation details for each application as follows.

Implementation for access-control rule mutation Multiview mutates each rule to lose the restriction on different attributes of the denied request based on the access-control rules to find the possible modifications. We classify methods of mutating access control into three categories: relaxing subject, action, and object based on the rule type. Developers need to write the mutation for each rule type as a one-time effort. We provide the details for the implementation as follows.

1. FS.
    - (a) Subject: For subjects, Multiview can change the subject (FS user) into three groups, `owner`, `group` and `other`. There are three possible ways to change the subject: (1) Change the denied user to be the `owner` of file; (2) Add the denied user to file's `group`; (3) Change user to be `other`. Multiview has the default security order that `owner` is more secure than `group`, and `group` is more secure than `other`. Therefore, if the user already has the `owner` role, Multiview will not try to change the user to `group` or `other` role.
    - (b) Action: For file system, there are three permissions `rwx` for three roles. After the user's role has been determined, there are multiple ways to grant the permissions, such as granting (1) execute, (2) execute and read, or (3) execute, read and write. Multiview applies a partial security order that less privilege is more secure. For two mutations, if the granted permissions of one mutation are subset of the other mutation, Multiview eliminates the one with excessive permissions.
    - (c) Object. Since the previous steps for file system have determined the objects that lack permission, Multiview does not grant permissions for other objects.
2. Apache.
    - (a) Subject. If the access-control rule is on the subject, Multiview will try to get all the groups of the denied user from the user database. Then Multiview can add a rule to allow the groups of the denied user, as well as an exception to only allow the denied user. The security order to compare the users' groups is that, if the members of one group are a subset of another group, allowing the subset is more secure.
    - (b) Action. If the access-control rule is related to actions, Multiview will grant the permissions based on rule category (e.g., HTTP method). Multiview will perform the mutations by granting the permissions. The partial security order is similar to FS case, Multiview compares and removes the mutation that grants excessively privileges to solve the issue.
    - (c) Object. Since Apache's access control is based on configuration blocks for each directory, Multiview can create an exception by adding a new configuration block to narrow down the scope of files, and only grant the denied user to access the block.
3. Nginx.
    - (a) Subject. Multiview currently does not handle the authorization related access-control rules in Nginx. Nginx supports different authentication methods but does not support complex authorization within the Nginx configuration. Instead, the authorization is configured in third party app, such as JWT authorization servers. Multiview can not handle such cases.
    - (b) Action. Similar to Apache, Multiview can mutate the rules related to the web request's HTTP actions. The partial security order is similar to FS case—Multiview will grant the minimum permission required to allow the web request by comparing and pruning the permissions.
    - (c) Object. Multiview can create a new configuration block which maps to the requested objects and only grant the denied user to access the block.
4. Vsftpd.
    - (a) Subject. Most of the access control on the subject (FTP users) are done with the file system permission. Each FTP user is also a system user which subjects to the FS permission constraints. Multiview handles this in the FS category.
      The configuration entry related to subject is `userlist_file` which can be used as block list to ban individual users. Multiview can perform the mutation on this rule by replacing user list with an empty list.
    - (b) Action. Vsftpd controls some FTP actions with boolean flags, including `write_enable`, `download_enable`. Multiview performs mutations on these rules by toggling the boolean values.
    - (c) Object. Most of the access control on the object (files) is done with the file system permission. The configuration related to object is `deny_file` which denies access to certain files matching the regex

expressions. Multiview can perform the mutation on this rule by replacing deny_file with an empty list.

5. Proftpd.
   (a) Subject. Proftpd has two kinds of access-control on the subject. The first type is through FS permissions. Multiview handles this through FS category. The second type is authentication/authorization rules on the subject. Similar to Apache, Multiview tries to get all the groups of the denied user from the user database. Then Multiview adds a rule to allow the groups of the denied user with `AllowGroup`, as well as an exception to only allow the denied user with `AllowUser`. The security order to compare the users' groups is that, if the members of one group are a subset of another group, allowing the subset is more secure.
   (b) Action. Proftpd controls some FTP actions with `<Limit>` inside the configuration blocks. Multiview mutates the rules related to the web request's HTTP actions. The partial security order is similar to FS case—Multiview grants the minimum permission required to allow the FTP request by comparing and pruning the permissions.
   (c) Object. Most of the access control on the object (files) is done with the file system permission. Multiview can create a new configuration block which maps to the requested objects and only grant the denied user to access the block to reduce the allowed objects.

6. MySQL.
   (a) Subject. Multiview collects the available roles in the system and tries to mutate the subjects by granting the roles to the denied user. The security order to compare the roles is that, if the role's privileges is a subset of another role, Multiview will prune the role with more privileges. Multiview will not try to grant `root` to the denied user for security reasons.
   (b) Action. If the access-control rule is related to actions, Multiview will grant privileges based on the object type. For each object type (e.g, database, table), Multiview mutates the permissions by granting relevant permissions. Multiview uses the partial security order that less privilege is more secure to prune the mutations.
   (c) Object. Since the previous steps for MySQL have determined the objects that lack permission, Multiview will not mutate permissions for other objects.

7. PostgreSQL.
   (a) Subject. The mutation for subjects in PostgreSQL is similar to MySQL database by grating the roles to the denied user. The security order compares the roles by checking whether the role's privilege is a subset of another role.
   (b) Action. The mutation for actions (privileges) is similar to MySQL. PostgreSQL has an additional layer between the database and other objects, which is the schema privileges. Multiview uses the partial security order that less privilege is more secure to prune the mutations.
   (c) Object. Since the previous steps for PostgreSQL have determined the objects that lack permission, Multiview will not mutate permissions for other objects.

8. MongoDB.
   (a) Subject. The mutation for subjects in MongoDB is similar to MySQL database by grating the roles to the denied user. The security order compares the roles by checking whether the role's privilege is a subset of another role.
   (b) Action. MongoDB can only grant roles to users, not individual privileges. Therefore, Multiview first creates roles with `db.createRole` to create the resource and actions pair, then grants the fine-granted roles to users. Multiview-created roles have less privileges than the built-in roles [**?**], as the built-in roles contain a set of actions that are not needed for the denied request.
   (c) Object. Since the previous steps for MongoDB have determined the objects that lack permission, Multiview will not mutate permissions for other objects.

9. Squid.
   (a) Subject. Squid is a caching proxy server that provides proxy and cache services. Squid supports different authentication methods. Multiview currently does not handle authorization-related access-control rules in Squid.
   (b) Action. If the access-control lists are related to actions (e.g., HTTP methods), Multiview adds an exception for the denied request by allowing the method from the request address before the rule. This is because Squid uses first-match wins to process the configuration settings. Multiview use the partial security order to allow as fewer privileges (e.g., HTTP methods) as possible to allow the denied request.
   (c) Object. Squid processes requests that usually contain access-control rules on objects related to IPs, domains or ports. To produce mutations for the rules related to the objects, Multiview will produce an exception to allow the accessed object for the denied user before the access-deny rule. This is because Squid uses first-match wins to process the configuration settings.

Implementation for rule combinations There are in general four combination operators for access-control rules, Satisfy All, Satisfy Any, Negate, and First Match The rules' results are combined with the above operators to get the final access-control result. Therefore, Multiview combines the mutations with based on the combination operators to generate the final change.

We discuss the implementation of rule combinations in two steps. First, we discuss the combination operators in each application as well as how to extract them. Second, we discuss the general data format to manipulate and store the rules.

1. FS. The combination operator in FS is an implicit Satisfy-All relationship for the accessed objects. An FS access should pass the access-control checks for all upper-level directories as well as the accessed file.
2. Apache. Apache provides all three kinds of operators with `RequireAll`, `RequireAny`, and `RequireNone` to combine the results of individual rules.
3. Nginx. Nginx's `satisfy` directive is used to specify the Satisfy All and Satisfy Any relationship. Negate relationship is not supported in Nginx configuration grammar.
4. Vsftpd. Vsftpd's access control is an implicit Satisfy-All relationship for all the entries in the configuration.
5. Proftpd. Proftpd has two modes for combining the rules. `Order allow,deny` and `Order deny,allow`. The result is a combination of First Match wins and Satisfy-Any relationship. For `Order allow,deny`, Proftpd first checks Allow rules, if any allow rules satisfy, the result is allow; Then it checks Deny rules, if any deny rules satisfy, the result is deny. Otherwise, the default result is allow. For `Order deny,allow`, Proftpd denies access by default, unless explicitly granted by an Allow directive.
6. MySQL. MySQL's access control is an implicit Satisfy-All relationship at the database and object level.
7. PostgreSQL. PostgreSQL's access control is an implicit Satisfy-All relationship at the database, schema, and object level.
8. MongoDB. MongoDB's access control is an implicit Satisfy-All relationship at the cluster, database, and collection level.
9. Squid. Squid's access control is an implicit First Match wins relationship. Therefore the order of rules matters in the configuration file.

Second, we talk about how to model the access-control operators. In Section **??**, we detail the configuration parsing process and store the configuration file in a JSON format. The nested JSON format stores rules along with an `operation` field to record the relationships among the rules. For First-Match wins, the rules are stored as a list to preserve the order between the rules. With the unified JSON format, Multiview performs the mutation of the rules, and can convert the JSON format to the original software-specific configuration format.

1.3 Implementation for change-impact analysis

Multiview applies change impact analysis to analyze the impact of configuration changes on the global access control state. To achieve that, Multiview first synthesizes the requests based on the subjects, actions and objects in the system. To reduce the number of requests, Multiview leverages the characteristics of rule change to reduce the number of subjects and objects in the synthesis. We discuss the detailed implementation for the two steps as follows.

1. FS.
   (a) Synthesis. For subject, Multiview collects the users in the OS as the subjects. For action, Multiview synthesizes read, write and execute actions. For object, Multiview performs a directory traversal from the root directory to collect all the files.
   (b) Reduction. If the rule changes an object's access control. Multiview only collects the changed object and all other objects depended on it if the data type is a directory. If the rule grants the denied subject a new role, Multiview collects all the data the role has access to measure how much permission the denied subject gains.
2. Apache.
   (a) Synthesis. For subject, Multiview collects the users from the group file for AuthFile authentication. Multiview currently does not support other third-party authentication providers such as LDAP or DB. For action, Multiview enumerates the HTTP methods for the request. For object, Multiview will synthesize URLs based on the files in the system.
   (b) Reduction. If the rule grants permission to a role, Multiview only need to collect users with the role instead of all users. If the rule changes the access control of an object, Multiview only needs to examine the object and any objects depends on it instead of all objects in the system.
3. Nginx.
   (a) Synthesis. For subject, Multiview collects the users from the group file. Other third-party authentication providers such as LDAP are not supported. For action, Multiview enumerates the HTTP methods for the request. For object, Multiview will synthesize URLs based on the files in the system.
   (b) Reduction. Similar to Apache, Nginx mostly changes the access control of blocks of data. Hence, Multiview only needs to collect all data within the block.
4. Vsftpd.
   (a) Synthesis. For subject, Multiview collects the FS users in the OS as the subjects because Vsftpd

relies on the system users for authentication. For action, Multiview uses all FTP methods. For object, Multiview performs a directory traversal from the root directory to collect all the files.

   (b) Reduction. Vsftpd reduces number of requests similar to file system. However, there are changes which impact the entire system, such as denying files based on regex, and Multiview does not perform any reduction.

5. Proftpd.
   (a) Synthesis. For subject in Proftpd, there are two sources: (1) FS and (2) customized authentication providers. Multiview first collects the FS users in the OS as well as users in `mod_auth_file` . Other authentication methods are not supported. For actions, Multiview uses all FTP methods. For objects, Multiview performs a directory traversal from the root directory to collect all the files.
   (b) Reduction. Since Proftpd's configuration syntax is similar to Apache, Multiview applies similar approach for reduction.

6. MySQL, PostgreSQL and MongoDB.
   (a) Synthesis. For subject, Multiview collects the roles in the database as the subjects. For action, Multiview synthesizes the SQL commands based on the object's type and corresponding privileges. For object, Multiview enumerates all the databases, as well as all the objects in each database (e.g., tables, procedures).
   (b) Reduction. Multiview parses the access control change command and extracts relevant users and objects. If the command changes subject's roles, Multiview only adds the members with the role. If the command changes the object (e.g., database), Multiview only enumerates the objects within the object related to rule change.

7. Squid.
   (a) Synthesis. For subject, Squid supports many authentication providers. Multiview currently only support basic authentication with Unix system users. The users' IP addresses can only be synthesized with simple heuristics that selecting random IPs from different subnets with target coverage . For action, Multiview enumerates the HTTP methods for the request. For object, Squid mostly handles connection to different domains with various ports. Multiview synthesize such requests with common domains with random ports in different ranges. As synthesizing IPs or strings for testing purposes is a hard problem, Multiview suggests using domains or IP addresses from previous logs to achieve a good coverage for historical accesses.
   (b) Reduction. Multiview does *not* perform reduction in Squid as the rule changes in Squid configuration will impact the whole system entirely.