Building Chinese Walls in Standard Unix*

Simon N. Foley
Department of Computer Science,
University College, Cork, Ireland.
(s.foley@cs.ucc.ie)

Abstract

The set-user-id facility in Unix can be used to form the basis for the implementation of a wide variety of different security policies in Unix. We show how the Chinese Wall security policy can be implemented using this facility. The approach is not appropriate for security critical applications: it serves to illustrate that it can be done in a rather simple way, and may be useful for less critical applications. Our technique also provides an approach to implementing dynamic segregation of duties in Unix.
KEYWORDS: SECURITY POLICY MODEL, CHINESE WALL, UNIX PROTECTION, ACCESS CONTROL, CLARK-WILSON INTEGRITY MODEL.

1 Introduction

The Chinese Wall security policy is an excellent example of a commercial non-disclosure policy. Consultants, or market analysts, consult for organizations that are quoted on the Stock Exchange. An analyst must maintain the confidentiality of organization information and is not permitted to advise an organization where she has insider knowledge of another competing organization.

Brewer and Nash [2] formalize the Chinese Wall policy in terms of a mandatory computer security model: organization information is stored in datasets (objects) and a consultant (user) is not allowed access organization information if it leads to a violation of the Chinese Wall policy. Initially, each user has the potential to access *any* dataset, but as accesses are made to particular datasets, the potential to access others is lost. This ensures

*Registered Trademark

that no conflict of interest can arise, that is, a consultant may not access information of competing organizations.

We capture a Chinese Wall security policy in terms of a conflict-of-interest relation [7] $(ORG, \_ \wr \_)$, where ORG is the set of organizations, and given $a, b \in ORG$, then $a \wr b$ means that a is in competition with b. A system enforces a Chinese Wall policy if it ensures that it is not possible for a consultant (user) to access information about $a, b \in ORG$, with $a \wr b$ (conflict of interest). We assume that the $\_ \wr \_$ relation is symmetric.

Example 1 Oil companies x and y and banks z and w are quoted on the stock exchange. The conflict-of-interest relation may be defined as

$\_ \wr \_$	x	y	z	w
x		$\surd$		
y	$\surd$			
z				$\surd$
w			$\surd$	

A consultant, Smith, working for bank, z, may not have access to bank w information. Similarly, consultant, Jones, working for bank, w, may not access bank z information. But both have the potential to access oil company x and y information.

Information dissemination must be considered in the Chinese Wall policy. For example, a system must ensure that it is not possible for consultant Jones to pass on any bank w information to Smith, leading to a conflict of interest. Whereas Smith and Jones can conduct insider trading outside the security perimeter of the system, possible Trojan Horse attack should be considered. For example, a Trojan Horse embedded in software run by Jones that forwards bank w information to Smith. $\triangle$

Since the publication of the original Chinese Wall policy model [2] there have been a number of generalizations and interpretations made; these include [4, 5, 7, 8, 9, 12]. Lin [8] and Kessler [7] generalize the original model and consider non-transitive conflict of interest, whereby $a \wr b$ and $b \wr c$ does not necessarily imply $a \wr c$. In this paper we use the conflict-of-interest relation [7] to specify a Chinese Wall policy. In [4, 9, 12] the Chinese Wall policy is modeled in terms of lattice-based Bell-LaPadula [1] style requirements, implying that a Chinese Wall policy can be enforced by a suitably configured multilevel secure system. A Chinese Wall policy can be viewed as a form of label-based aggregation policy, and a non-interference based semantics for such policies is given in [5].

In this paper we consider how the Chinese Wall policy can be enforced using the standard Unix protection mechanisms. Section 2 describes a simple method of encoding the policy in terms of Unix users and groups. Since standard Unix enforces discretionary access controls, it cannot, in general, constrain the arbitrary dissemination of data. This is unlike the mandatory access control models [4, 7, 8, 9, 12] which do constrain the dissemination of data. Thus an inherent problem with the approach proposed in Section 2 is that it open to Trojan Horse attack such as that considered in Example 1.

This class of attack can be limited if, instead of providing arbitrary access to organization files, files may be accessed only via special secure applications. Section 3 considers how such secure applications should be supported. The Clark-Wilson model [3] is used to provide a basis for modeling and evaluating these secure applications. We show how the Chinese Wall policy can be encoded in a slightly modified version of the Clark-Wilson model and consider how this, in turn, can be supported within Unix. Section 4 outlines how the Chinese Wall policy can be applied to existing secure applications that are already described in terms of the Clark-Wilson model.

Section 5 illustrates that a Chinese Wall policy, which is usually thought of as an aggregation policy, can alternatively be viewed as a segregation of duty policy, and thus our proposed mechanism can be used for supporting such policies.

2 Capturing Conflict of Interest

2.1 User-Groups and Access-Control in Unix

Unix supports simple access-control based protection policies. These are defined in terms of *user-ids*, *group-ids* and *access-permissions* on files.

Let *UID* represent the set of all user-ids in a Unix system. This is effectively the user-id entries contained in the file `/etc/passwd`. Let *GID* represent the set of all group-ids in a Unix system. Each user is a member of one or more of these groups. Details on group membership is contained in files `/etc/passwd` and `/etc/group`. For simplicity we characterize this by the function $mbrs$, where $mbrs(g)$ gives the set of user-ids of users that are members of the group g.

Associated with each file f is the file owner's user-id, group-id and access permissions. The latter specify the access rights that users may have on the file. Access permissions (including, `r`, `w` and `x` access) may be set for: the owner of the file (user-access); any user who is a member of the file's group-id (group-access), and any other user that is not the owner or in the file's group (other-access). The reader is referred to [6] for further information on the Unix protection mechanisms.

Example 2 A system has user-ids `smith` and `jones`, and group-ids `sales` and `admin`, where $mbrs(\texttt{sales}) = \{\texttt{smith}, \texttt{jones}\}$, $mbrs(\texttt{admin}) = \{\texttt{smith}\}$. A file, `comms`, of sales commissions is created by `jones` and may be read by anyone in the `sales` group. Permission bits, and so forth, can be captured in a self-explanatory way by the following protection profile.

	user	group	other
`comms`	`rw-`	`r--`	`---`
	`jones`	`sales`	

△

Example 3 Only the system administrator (user `root`) may create group-ids and specify group membership. Thus we have the protection profile:

	user	group	other
`/etc/group`	`rw-`	`r--`	`r--`
	`root`	`wheel`	

△

2.2 Representing Chinese Walls as User-Groups

In our initial scheme every consultant is allocated a unique user-id and every organization $a \in ORG$ is allocated a unique phantom user-id, denoted as $uorg(a)$ and a unique group-id denoted $gorg(a)$ (functions $uorg$ and $gorg$ are one-to-one). A phantom user-id has no associated user. Examples of existing phantom user-ids include `uucp` and `daemon`.

Each file (dataset) f containing data from organization a has owner user-id $uorg(a)$, group-id $gorg(a)$, and protection bits `rw` set for group access only. Thus, at any moment $mbrs(gorg(a))$ represents the users (consultants) who may access files of organization a.

Initially $mbrs(gorg(a))$ is empty for every organization a. The system administrator (`root`) is willing to add a consultant to an organization group if, based on current organization group membership, no conflict of interest can arise as defined by $(ORG, \_ \wr \_)$.

Example 4 The conflict-of-interest relation described in Example 1 can be enforced by our scheme. A file with name `f`, containing information about bank z, has protection profile:

	user	group	other
`f`	`---`	`rw-`	`---`
	`uz`	`gz`	

where $uorg(z) = \texttt{uz}$ and $gorg(z) = \texttt{gz}$, and initially $mbrs(\texttt{gz}) = \{\}$. The consultant with user-id `cons` can be granted access to Bank z files by setting $mbrs(\texttt{gz}) = mbrs(\texttt{gz}) \cup \{\texttt{cons}\}$. However, given the conflict-of-interest relation, this consultant may not be subsequently added to the group of bank w. $\triangle$

2.3 Automated Arbitration for Conflict of Interest

The arbitration procedure carried out by the system administrator can be automated by implementing it as a set-user-id (suid) root program. Such a program (file) is owned by root, and has its suid permission bit set. When executed by an arbitrary user, it temporarily runs with an effective user-id of that of `root` (rather than that of the invoker).

Given the conflict-of-interest relation $(ORG, \_ \wr \_)$, implemented in a file `/etc/conflict`, Figure 1 defines an atomic suid root operation $consult1(c, o)$, which may be invoked by a consultant c wishing to access organization o files. Use of suid root programs make a system vulnerable to certain types of attack. Section 6 considers this and proposes an implementation that is resilient to these attacks.

$$
\begin{array}{|l|}
\hline
consult1(cons : UID; o : ORG)\{ \\
\quad consfor \leftarrow \{a : ORG | cons \in mbrs(gorg(a))\}; \\
\quad \text{if } \ (\forall a \in consfor \bullet \neg a \wr o) \ \text{ then} \\
\qquad mbrs(gorg(o)) \leftarrow mbrs(gorg(o)) \cup \{cons\} \\
\quad \text{else} \\
\qquad \text{reject: } \textit{conflict of interest} \\
\} \\
\hline
\end{array}
$$

Figure 1: Granting Organization Access: Unix Encoding.

Proposition 1 It is not possible for a consultant to gain *access* to files owned by competing organizations.

This follows from the definition of *consult*1, given the assumptions: that all organization files and consultants have protection profiles as outlined above; that organization groups may grow according only to operation *consult*1, and that user `root` is trusted not to modify the protection profiles or violate the conflict-of-interest relation in any way. □

3 Controlling Insider Trading

A problem with the proposed scheme is that insider trading is still possible. The mechanism works at the granularity of access-rights: Proposition 1 states that a consultant cannot gain *access* to files owned by competing organizations; it does not constrain the *propagation* of organization data. A consultant can copy organization data into a public area. This could be done inadvertently, deliberately or by a Trojan Horse.

Standard Unix enforces discretionary access control which in general does not constrain the propagation of data. The attack can be limited if, instead of providing arbitrary access to organization files, the files may be accessed (read and/or write) only via special secure applications. These applications support the Chinese Wall policy and are expected not to permit arbitrary copying of organization data. Standard Unix provides support for secure applications. For example, a user may have (write) access to the password file only when running the `passwd` program.

3.1 Modeling and Evaluating Secure Applications

Clark and Wilson [3] propose a model for (integrity) security that can be used for systems where security is enforced across the operating system *and* the application systems. Their model is based on commercial data processing practices and can be used as a basis for evaluating the security of a complete application system.

The Clark-Wilson (CW) model is defined in terms of *enforcement rules* and *certification rules*. Enforcement rules specify security requirements that should be supported by the protection mechanisms in the underlying operating system. The certification rules specify security requirements that the application system should uphold. There are nine rules in total, but for reasons of space we will consider only those rules that are immediately relevant to supporting Chinese Walls. The reader is referred to [3] for a more in-depth discussion.

The model components include: the *Users* of the system; *Constrained Data Items* (CDIs) representing data objects with integrity, and *Transform Procedures* (TPs) that operate on CDIs and represent the well-formed transactions that provide the functionality of the application system.

Enforcement Rule E2. The main access-control requirement underlying the CW-model is that users may not directly access CDIs, but only via TPs, and then only if the access is specified in the *E2-access* relation. For our purposes, the *E2-access* relation is a set of access-triples configured for a particular application system. An access-triple, given as (u, t, c), is interpreted to mean that the user u may use the TP t to access the CDI c[1]. This requirement forms of the basis of the CW model enforcement Rule $E2$ and must be enforced by any system supporting an application to be evaluated according to the CW-model.

Certification Rule C2. The application system TPs must be certified to operate correctly on the CDIs. This is documented in the *C2-access* relation which is a set of certification tuples of the form (t, c), where t is a TP and c a CDI. Given a TP t, then *C2-access* is interpreted to mean that t has been certified to to correctly operate on the CDIs given in the set $\{\ c : \mathit{CDI} | (t, c) \in \mathit{C2\text{-}access}\ \}$. This requirement forms the basis of the CW model certification Rule $C2$.

Example 5 Organizations x and y store data in CDIs `cdx` and `cdy`, respectively. A statistical application may be used to access these CDIs. The TP `stats` provides appropriate statistics about an organization, while TP `modify` is used to update an existing CDI. We have

$$\mathit{C2\text{-}access} \triangleq \{(\mathtt{stats}, \mathtt{cdx}), (\mathtt{stats}, \mathtt{cdy}), (\mathtt{insert}, \mathtt{cdx}), (\mathtt{insert}, \mathtt{cdy})\}$$

reflecting that the TPs `stats` and `insert` have been certified to correctly operate on the given CDIs. Given users `smith` and `jones`, define

$$\begin{aligned} \mathit{E2\text{-}access} \triangleq\ & \{(\mathtt{smith}, \mathtt{stats}, \mathtt{cdy}), (\mathtt{smith}, \mathtt{insert}, \mathtt{cdy}), \\ & (\mathtt{jones}, \mathtt{stats}, \mathtt{cdx}), (\mathtt{jones}, \mathtt{insert}, \mathtt{cdx})\} \end{aligned}$$

Enforcement Rule $E2$ requires that (the system ensures that) user `smith` may only use TP `insert` to update CDI `cdy`. Certification Rule $C2$ represents the requirement that the implementation of `insert` appropriately

[1]This is a simplification of the usual definition of access triple [3]. It is sufficient for our exposition and has a closer correspondence to its Unix implementation.

updates CDI `cdy` and does not, for example, store a copy of it in a public place. $\triangle$

3.2 Encoding the Chinese Wall policy in Clark-Wilson

To limit Trojan Horse attacks we will formulate the Chinese Wall policy in terms of a CW-model, where access to organization data may be made only via certified secure applications.

Consider an application that is used to access organization data. Let the set *CDI* represent the set of organization files (CDIs), *TP* represent the set of application operations (TPs), and let $own(c)$ give the organization that owns the CDI c.

Certification Rule C2. We consider the most general case where every application TP may potentially access any collection of organization CDIs[2]. It must be certified that each TP correctly accesses the CDIs, in particular, each TP does not copy information between CDIs owned by different organizations. Furthermore, the TPs must not copy any CDI information such that it becomes visible to all users, that is, data is not copied to objects that do not correspond to CDIs. This certification carried out for Rule $C2$ attempts to ensure that TPs do not contain Trojan Horses. Thus for the Chinese Wall policy we define

$$\mathit{C2\text{-}access} \triangleq \mathit{TP} \times \mathit{CDI}$$

Enforcement Rule E2. Consultants may access CDIs only via TPs. Initially consultants may not access any CDI.

$$\mathit{E2\text{-}access} \triangleq \{\}$$

A consultant *cons* wishing to access organization o data invokes operation $consult2(cons, o)$, which appropriately updates the *E2-access* relation if it does not result in a conflict of interest. Operation $consult2(c, o)$ is defined in Figure 2. If there is no conflict of interest then a series of access triples are added to *E2-access*, giving the requesting consultant TP access to all CDIs owned by the specified organization. Note, the syntax for set specification is interpreted as: given variable declarations D, predicate P and expression E then the components of $\{D|P \bullet E\}$ are the values taken by expression E

[2]For the sake of clarity we do not consider the case where the application has its own CW-model based security requirements. Supporting such applications is considered in Section 4.

$$
\begin{array}{|l|}
\hline
consult2(cons: UID; o: ORG)\{ \\
\quad consfor \leftarrow \{\ t: TP; c: CDI | (cons, t, c) \in \textit{E2-access} \bullet own(c)\ \}; \\
\quad \text{if } (\forall a: consfor \bullet \neg a \wr o) \text{ then} \\
\quad\quad \textit{E2-access} \leftarrow \textit{E2-access} \cup (\{cons\} \times TP \times \{c: CDI | own(c) = o\}) \\
\quad \text{else} \\
\quad\quad \text{reject: } \textit{conflict of interest} \\
\} \\
\hline
\end{array}
$$

Figure 2: Granting Organization Access: CW-Model Encoding.

when the variables introduced by D take all possible values that make the predicate P true.

Proposition 2 Given that the *E2-access* relation may change according only to operation *consult*2 defined in Figure 2 then it follows that it is not possible for a consultant to gain access to CDIs owned by competing organizations. □

Example 6 Consider the statistical application system in Example 5 supporting the Chinese Wall policy described in Example 1. There are CDIs `cdx`, `cdy`, `cdz` and `cdw`, owned by organizations x, y, z and w, respectively.

TPs `insert` and `stats` must be certified to uphold that they do not copy data from `cdx` to `cdy`, and so forth. Given consultants `smith` and `jones` we have, initially

$$\textit{E2-access} \triangleq \{\}$$

The requests $consult2(\texttt{smith}, x)$ and $consult2(\texttt{jones}, y)$ result in

$$
\begin{aligned}
\textit{E2-access} \triangleq\ & \{(\texttt{smith}, \texttt{insert}, \texttt{cdx}), (\texttt{smith}, \texttt{stats}, \texttt{cdx}), \\
& (\texttt{jones}, \texttt{insert}, \texttt{cdy}), (\texttt{jones}, \texttt{stats}, \texttt{cdy})\}
\end{aligned}
$$

Note that `smith` will be subsequently refused the request $consult2(\texttt{smith}, y)$ due to a conflict of interest between banks, that is, $x \wr y$. However, both consultants may request to consult for organization z, resulting in

$$
\begin{aligned}
\textit{E2-access} \triangleq\ & \{(\texttt{smith}, \texttt{insert}, \texttt{cdx}), (\texttt{smith}, \texttt{stats}, \texttt{cdx}), \\
& (\texttt{smith}, \texttt{insert}, \texttt{cdz}), (\texttt{smith}, \texttt{stats}, \texttt{cdz}), (\texttt{jones}, \texttt{insert}, \texttt{cdy}), \\
& (\texttt{jones}, \texttt{stats}, \texttt{cdy}), (\texttt{jones}, \texttt{insert}, \texttt{cdz}), (\texttt{jones}, \texttt{stats}, \texttt{cdz})\}
\end{aligned}
$$

While both users share access to `cdz`, the certification Rule $C2$ reflects that it is not possible for a Trojan Horse running as `smith` to copy `cdx` data to `cdz` where it could be accessed by `jones`. $\triangle$

3.3 Implementing Clark-Wilson Chinese Walls in Unix

The Unix set-user-id mechanism can be used to support CW access-triples [11, 14]. The essence of the strategy is as follows. Given an access triple (u, t, c), then by appropriate configuration of user-groups, the user u has execute and suid access to the TP t, TP t has access to CDI c, and only when executing the suid program t does the user u gain access to CDI c.

As proposed earlier, every organization a is allocated a unique phantom user-id $uorg(a)$ and a unique group-id $gorg(a)$. Each CDI c corresponds to a file and has owner user-id $uorg(own(c))$, group-id $gorg(own(c))$, and has protection bits `rw` set for group access only.

Initially, assume that there is just one consultant `cons`. We allocate an additional unique phantom user-id and group-id, denoted `utp` and `gtp`, respectively. Each TP t is taken to represent an executable program stored in a separate file which has owner user-id `utp`, group-id `gtp`, and has protection bit `x` set for group access and protection bit `sr` set for owner. No other protection bits are set.

Consultant `cons` may invoke any $t \in TP$ and thus $mbrs(\mathtt{gtp}) = \{\mathtt{cons}\}$. Initially, application TPs may not access any CDI, that is, for every organization o then $mbrs(gorg(o)) = \{\}$. When consultant `cons` wishes to consult for organization o she requests the Unix suid root operation $consult1(\mathtt{utp}, o)$, as defined in Figure 1. If there is no conflict of interest then the user-id `utp` is added to group $gorg(o)$, and thus user `cons` may subsequently use the TPs to access all files owned by organization o. It follows from Proposition 1 that, for this case of one consultant, the Chinese Wall policy is enforced.

Example 7 Consider just one user `smith`, CDI `cdx` (owned by organization x), and TPs `insert` and `stats` from Example 5. We have the following initial user-groups and protection profiles

$$\begin{array}{ll} uorg(x) = \mathtt{ux} & mbrs(\mathtt{gx}) = \{\} \\ gorg(x) = \mathtt{gx} & mbrs(\mathtt{gtp}) = \{\mathtt{smith}\} \end{array}$$

	user	group	other
cdx	---	rw-	---
	ux	gx	

	user	group	other
insert	sr-	x--	---
	utp	gtp	

And the TP **stats** has a protection profile similar to TP **insert**. Under this configuration **smith** may invoke the TPs, but cannot access any CDI. This corresponds to the initial state $\textit{E2-access} = \{\}$.

When **smith** requests $\textit{consult}1(\texttt{utp}, x)$, there is no conflict of interest and thus $\textit{mbrs}(\texttt{gx})$ is updated to $\{\texttt{utp}\}$. This has the effect of granting the TPs access to organization x files. In this case we have

$$\textit{E2-access} \triangleq \{(\texttt{smith}, \texttt{insert}, \texttt{cdx}), (\texttt{smith}, \texttt{stats}, \texttt{cdx})\}$$

If another user **jones** requests access to organization y files then she must access these files using a *different* copy of TP **insert**. Otherwise, granting the existing **insert** TP access to organization y would result in user **smith** indirectly acquiring access to organization y files; a conflict of interest. $\triangle$

To generalize the implementation to many consultants we require unique copies of TPs for each consultant or use TP wrappers, as proposed in [11]. This is done as follows. For every consultant *cons* we allocate an additional unique phantom user-id and group-id, denoted $\textit{utp}(\textit{cons})$ and $\textit{gtp}(\textit{cons})$, respectively. Each consultant *cons* is allocated their 'own' unique copies of TPs, each stored in a separate file with owner user-id $\textit{utp}(\textit{cons})$, group-id $\textit{gtp}(\textit{cons})$, and protection bits set as before.

Each consultant *cons* may invoke their copies of the TPs, that is, we have $\textit{mbrs}(\textit{gtp}(\textit{cons})) = \{\textit{cons}\}$. Initially these TPs may not access any CDI, that is, for every organization o then $\textit{mbrs}(\textit{gorg}(o)) = \{\}$. To consult for organization o a consultant *cons* requests $\textit{consult}3(\textit{cons}, o)$, which is defined in Figure 3.

$$\boxed{\begin{array}{l} \textit{consult}3(\textit{cons}: \textit{UID}; o: \textit{ORG})\{ \\ \quad\quad \textit{consult}1(\textit{utp}(u), o); \\ \} \end{array}}$$

Figure 3: Granting Organization Access: Unix CW-model Encoding.

Proposition 3 The user-group encoding of the Chinese Wall policy based on *consult*3 correctly implements the CW-model encoding of the Chinese Wall policy based on *consult*2. The proof is given as an appendix. $\square$

4 Supporting Secure Applications

Sections 3.2 and 3.3 capture and implement the Chinese Wall policy in terms of a slightly modified CW-model, but do not consider the case where the original application is itself already modeled and evaluated according to the CW-model.

Our scheme can be generalized to support such secure applications, that is, to support an application's security requirements *and* the Chinese Wall policy. In this case, the certification Rule $C2$ is applicable only to the original secure application's *C2-access* relation, which we represent by the relation *AC2-access*.

If *AE2-access* specifies the access-triples for this secure application, then $consult2(cons, o)$ should update *E2-access* only with access triples that do not contradict the application's original *AE2-access* access triples, that is, if there is no conflict of interest, then

$$\begin{aligned} \textit{E2-access} \leftarrow \textit{E2-access} \cup \; & \textit{AE2-access} \\ & \cap \{\, t : TP; a : CDI | (t, c) \in \textit{AC2-access} \wedge own(c) = o \\ & \qquad \bullet (cons, t, a) \,\} \end{aligned}$$

It is straightforward to implement this revised encoding of a Chinese Wall policy in Standard Unix given the guidelines in [11].

5 Dynamic Segregation of Duties

Dynamic segregation of duties [10] may be viewed as an integrity dual of aggregation policies (such as Chinese Walls) [5]. For example, a clerk may *either* generate a purchase order *or* process an invoice, but not both. Representing this as a conflict-of-interest ($\texttt{inv} \wr \texttt{po}$) allows it to be supported by our mechanism. Given initial *E2-access* $= \{\}$, the clerk may access invoices ($consult(\texttt{inv})$) or purchase orders ($consult(\texttt{po})$), but not both.

6 Discussion

The effectiveness of the proposed technique relies heavily on the suid feature of Unix. Use of suid makes a system vulnerable to certain types of attack [6]. For example, a suid shell-script owned by `root` cannot be made secure; there is always the potential that the invoking user will interrupt it and subsequently gain arbitrary `root` access. Sometimes suid programs have to be written, and for these situations, [6] suggests using *tainted Perl* [15] which provides a degree of security against subversion that is not available in shell or C programs. Appendix B gives a tainted Perl implementation of *consult*1.

Individual TP's are also suid programs and need to be checked for vulnerabilities to suid attack. However, they are not owned by `root`, and a compromise may be considered less critical than compromise of suid `root` programs. As noted in [11], this checking should form part of the certification performed on TPs during the CW-model evaluation of the application.

Security kernel based systems implementing the multilevel formulations of the Chinese Wall policy [4, 5, 9, 12] assume that applications running on the system are untrusted, and therefore the application programs do not require security policy certification. The approach in this paper adopts the Clark-Wilson philosophy that the security of an application system is spread across both the underlying system and the application(s) running on it.

In terms of encoding a Chinese Wall policy within the CW-model an issue is whether it is feasible and/or desirable to require that consultants access organization files only via secure applications. In a highly structured and proceduralized information system where consultants may use only predefined programs to access organization files it would be reasonable to adopt this approach. However, if consultants require arbitrary access to organization files then it becomes more difficult to model the system as an application that can be realistically evaluated.

7 Conclusion

This paper illustrated that it is possible to enforce the Chinese Wall security policy using standard Unix protection mechanisms. The approach is not appropriate for security critical applications: it serves to illustrate that it can be done in a rather simple way, and may be useful for less critical applications.

References

[1] D. E. Bell and L. J. La Padula. Secure computer system: unified exposition and MULTICS interpretation. Report ESD-TR-75-306, The MITRE Corporation, March 1976.

[2] D.F.C. Brewer and M.J. Nash. The Chinese Wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214. IEEE Computer Society Press, May 1989.

[3] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security models. In *Proceedings 1987 IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, April 1987.

[4] S.N. Foley. Secure information flow using security groups. In *Proceedings of the Computer Security Foundations Workshop*, pages 62–72. IEEE Computer Society Press, 12–14 June 1990.

[5] S.N. Foley. Aggregation and separation as noninterference properties. *Journal of Computer Security*, 1(2):159–188, 1992.

[6] S. Garfinkel and G. Spafford. *Practical Unix & Internet Security.* O'Reilly & Associates, 1996.

[7] V. Kessler. On the chinese wall model. In *European Symposium on Research in Computer Security*, pages 39–54. Springer Verlag, LNCS 875, 1992.

[8] T. Lin. Chinese wall security policy - an aggressive model. In *Proceedings Aerospace Computer Security Applications Conference*, pages 282–289. IEEE Computer Society Press, 1990.

[9] C. Meadows. Extending the Brewer Nash model to a multilevel context. In *Proceedings Symposium on Security and Privacy*, pages 95–102, Oakland, CA, 1990. IEEE Computer Society Press.

[10] M.J. Nash and K.R. Poland. Some conundrums concerning separation of duty. In *Proceedings Symposium on Security and Privacy*, pages 201–207, Oakland, CA, May 1990. IEEE Computer Society Press.

[11] W.T. Polk. Approximating Clark-Wilson access triples with basic UNIX controls. In *Unix Security Symposium IV*, pages 145–154, 1993.

[12] R.S. Sandhu. Lattice based access control models. *IEEE Computer*, 26(11):9–19, November 1993.

[13] J. M. Spivey. *The Z Notation: A Reference Manual.* Series in Computer Science. Prentice Hall International, Hemel Hempstead, UK, second edition, 1992.

[14] D.J. Thomsen and J.T. Haigh. A comparison of type enforcement and Unix setuid implementation of well-formed transactions. In *Computer Security Applications Conference*, pages 304–312. IEEE Computer Society Press, 1990.

[15] L. Wall and R. Schwartz. *Programming with Perl.* O'Reilly & Associates, 1991.

A Proof of Proposition 3

We prove that the user-group encoding of the Chinese Wall policy based on *consult*3 is a refinement, in the sense of [13], of the CW-model encoding of the Chinese Wall policy based on *consult*2.

A.1 System State

The set *E2-access* characterizes the state of the CW-model encoding of the Chinese Wall policy. At any moment in time this defines what accesses a consultant may make. At any moment in time the Unix user-group configuration as constructed in Section 3.3 is intended to implement *E2-access*, defining the accesses that a consultant may make.

Much of the Unix user-group configuration remains unchanged from its initial configuration: permission-bits, owner user-ids and group-ids. Only the group membership function *mbrs* changes, and for the sake of simplicity we think of this function as characterizing the state of the Unix user-group implementation.

A.2 Data Refinement

Given a state of the user-group implementation that is characterized by *mbrs* we can re-construct the abstract CW-model state it implements using the abstraction mapping $E2\text{-}access = Abs(mbrs)$, where

$$\begin{aligned} Abs(mbrs) \triangleq \{\ & cons: UID; t: TP; c: CDI \\ & \mid utp(cons) \in mbrs(gorg(own(c))) \\ & \bullet\ (cons, t, c)\ \} \end{aligned}$$

Since, by definition, we have $cons \in mbrs(gtp(cons))$ then *cons* may invoke its own copy of t. Thus $utp(cons) \in mbrs(gorg(own(c)))$ implies that *cons* may access c using t, that is, we have the access-triple $(cons, t, c)$. If

$utp(cons) \notin mbrs(gorg(own(c)))$ then *cons* cannot use any t to access CDI c, that is, we do not want the access-triple $(cons, t, c)$.

Under this interpretation of state implementation, it follows that every (legal) user-group state configuration implements a corresponding CW-model state.

We also have that the initial state configurations for both schemes are equivalent.

A.3 Operation Refinement

We must prove that *consult3* correctly implements the *consult2* operation. That is, given a request *consult3* that changes *mbrs* to *mbrs'*, and given that *mbrs* implements CW-model state *E2-access* ($E2\text{-}access = Abs(mbrs)$) then applying *consult2* to *E2-access* produces a state *E2-access'* that is properly implemented by *mbrs'* ($E2\text{-}access' = Abs(mbrs')$).

Consider $consult3(cons, o)$ requested in state *mbrs*. This is equivalent to the request $consult1(utp(cons), o)$. In this implementation state the set of organizations that *cons* may currently access is defined as

$$consfor = \{\ o : ORG | utp(cons) \in mbrs(gorg(o))\ \}$$

and, since every TP can potentially access every CDI, we can rewrite the above as

$$\begin{aligned} consfor &= \{o : ORG | (\exists c : CDI \bullet utp(cons) \in mbrs(gorg(own(c))))\} \\ &= \{t : TP; c : CDI | utp(cons) \in mbrs(gorg(own(c))) \bullet own(c)\} \end{aligned}$$

Applying the abstraction mapping to *mbrs* we thus have, in the abstract CW-model state,

$$consfor = \{t : TP; c : CDI | (cons, t, c) \in E2\text{-}access \bullet own(c)\}$$

which corresponds to its definition in *consult2*.

Therefore, if *consult3* is applied to state *mbrs* and there is a conflict of interest, then by its definition $mbrs' = mbrs$. If *consult2* is applied to state $Abs(mbrs) = E2\text{-}access$, then we similarly have a conflict of interest and by its definition $E2\text{-}access = E2\text{-}access'$.

If there is no conflict of interest then $consult1(utp(cons), o)$ changes *mbrs* to *mbrs'*, where $mbrs(o)' = mbrs(o) \cup \{utp(cons)\}$, and for $a \neq o$, $mbrs'(a) =$

$mbrs(a)$. Thus we have

$$\begin{aligned}
Abs(mbrs') &= \{u : UID; t : TP; c : CDI | utp(u) \in mbrs'(gorg(own(c)))\} \\
&= \{u : UID; t : TP; c : CDI | utp(u) \in mbrs(gorg(own(c)))\} \\
&\quad \cup \{u : UID; t : TP; c : CDI | utp(u) = utp(cons) \land own(c) = o\} \\
&= \textit{E2-access} \cup (\{cons\} \times TP \times \{c : CDI | own(c) = o\}) \\
&= \textit{E2-access}'
\end{aligned}$$

Therefore, applying $consult2(cons, o)$ to state $\textit{E2-access} = Abs(mbrs)$ produces a state $\textit{E2-access}' = Abs(mbrs')$, and thus $consult3(cons, o)$ is a refinement of operation $consult2(cons, o)$.

B Tainted Perl Implementation of *consult1*

```
#!/usr/local/bin/taintperl
# Command–line format: consult(<organization>)

$ENV{'PATH'}= '/usr/bin:/usr/local/bin:/usr/ucb';
if ($#ARGV≠0){die "usage: $0 <organization>\n"};

$reqorg= $ARGV[0];                 #requesting consult for this organization
$reqcons= getlogin;                                     #user making request
$groupfile= "/etc/group";
$conflict= "/etc/conflict";    #Implements conflict–of–interest relation
                                             #format is similar to /etc/group

# @conflict gives groups in conflict with requested organization
open(COI,"$conflict") || die "Fail: Chinese Wall not installed\n";
flock(COI,2);                                 #start mutex: consult is atomic
while (<COI>)
  {last if /^$reqorg/;}              #scan until line entry for $reqorg found
if (!<COI>) {die "Organization $reqorg not defined in policy\n"};
chop; @conflict= split(/ |:|,/); shift(@conflict);  #extract conflicting groups

# @consfor gives organizations that consultant works for.
$G=`groups $reqcons`; chop($G);          #returns users group membership
@consfor=split(/ /,$G);                                     #format into array
```

```
# Test if there is conflict of interest.
foreach $corg (@conflict){
  foreach $uorg (@consfor){
    if ($corg eq $uorg){
      die "Conflict of interest. Already consulting for $uorg\n\
           which is in competition with organization $reqorg\n"};
}}

# Add user $reqcons to this group.
`rm -f /tmp/group > /dev/null` ;
`mv $groupfile /tmp/group`;
open(OLDGRP,  "/tmp/group") || die "Failure: No group file.\n";
open(NEWGRP, ">$groupfile");
while(<OLDGRP>){
  s/$/,$reqcons/ if (/^$reqorg:/ && !/$reqcons/);
  s/:,/:/;
  print NEWGRP;
}
close(OLDGRP);
`rm -f /tmp/group > /dev/null` ;
close(NEWGRP);
print "Start a new shell to access $reqorg datasets\n";
flock(COI,8);                                    #end of mutex region
close(COI);
```