Appendix C

A. Datum Access Control API

The A. Datum Access Control API threat model illustrates how threat modeling can be applied to a software library that is used by other applications. A. Datum's Access Control API might be integrated into any application that needs to protect user and group access to resources. This example shows how threat modeling can be performed at the software feature level. In this case, the feature is the access control class library.

Table C-1 contains the high-level information about the threat model being developed for the A. Datum Access Control API application. This basic information includes the type of product and its location, the owner of the threat model and team members, and any available milestone information.

Table C-1 Threat Model Information

Product	Access Control API
Milestone	RTM
Owner	Ann Beebe
Participants	Kari Hensien, Magnus Hedlund, Scott Gode
Reviewer	Bradley Beck
Location	A. Datum Libraries\A. Datum Access Control API
Description	The A. Datum Access Control API is a class library that can be used by application developers to protect user and group access to resources. The class library has an abstracted concept of a hierarchical resource tree that can have access control lists applied on a resource-by-resource basis. Library users must supply a class that maps this resource tree to the underlying resources being protected. Access control lists are inherited from the parent unless otherwise specified. The library can also implicitly generate resource nodes if Implicit mode is set.

Use Scenarios

Table C-2 lists the use scenarios for the application—in other words, information about its expected use. Using or deploying the application in a way that violates these use scenarios can impact the security of the component.

Table C-2 Use Scenarios

ID	Description
1	*ACADatumOSFilePathResolver* is an implementation of *IACPathResolver* that resolves pathnames on A. Datum Operating System 1.0 but will not work for other operating systems. Using this implementation on other operating systems could result in incorrectly mapping pathnames to the A. Datum Access Control API canonical form.

External Dependencies

Table C-3 lists the external dependencies on other components or products that can impact the Access Control API's security. These dependencies are assumptions made about the usage or behavior of those other components or products. Inconsistencies in these assumptions can lead to security weaknesses in the API.

Table C-3 External Dependencies

ID	Description
1	The *ACADatumOSFilePathResolver* depends on file paths in A. Datum Operating System 1.0 being resolved in the same manner as implemented in this class. If the file paths resolve differently in the operating system, the mapping that the *ACADatumOSFilePathResolver* performs could result in an incorrect resource being checked for access. At worst, this could allow an adversary to access a file he could not normally access.

Implementation Assumptions

Table C-4 describes the implementation assumptions made about the internal workings of the component during the specification phase, but before implementation has started. These assumptions should not be violated. Typically, the threat modeling team will further review these assumptions once implementation is in place.

Table C-4 Implementation Assumptions

ID	Description
1	If Deny is added to the Access Control API, any Deny should have precedence over any Allow. Thus, if a user is denied access to a resource or the user belongs to a group that is denied access, the denial of access takes precedence over any Allow granted to the user or a group she belongs to.

External Security Notes

Table C-5 lists the external security notes—the information that an application user should be aware of to prevent possible vulnerabilities. These notes can include features that, if used incorrectly, could cause security problems for application users.

Table C-5 External Security Notes

ID	Description
1	*ACADatumOSFilePathResolver* is intended for use on A. Datum's Operating System 1.0 only. Using *ACADatumOSFilePathResolver* on other platforms is not supported and could cause errors.
2	Implicit mode in the *ACResources* class allows implicitly generated resources. If enabled, the *ACResources* class will generate resource nodes according to the documentation for *ACResources.SetExplicitMode*. Users who want only explicit resources to be resolved should not use Implicit mode. Furthermore, enabling Implicit mode could generate nodes for resources that do not actually exist. This is because the Access Control API does not actually reference the underlying system when generating these nodes.
3	Requesting access to a resource with no flags set will always return access granted. (Flag settings are *FlagRead* = 1, *FlagWrite* = 2, *FlagReadWrite* = 3.) Because no access rights were requested, this result is considered correct behavior. However, programmers using the API should be aware of this case and ensure that they use it appropriately.
4	Users who implement the *IACPathResolver* interface should use the same parsing and normalization as the underlying system that stores the resource. This will prevent inconsistencies between path resolution in the resolver and the underlying system, which could result in access checks being done on one resource in the Access Control API and another resource being opened by the underlying system.

Internal Security Notes

Table C-6 lists the internal security notes, which contain security information relevant only to people reading the threat model. These notes can be used to explain choices and design decisions that impact security but were made for overriding business needs.

Table C-6 Internal Security Notes

ID	Description
1	To prevent a user from causing a denial of service when Explicit mode is disabled by repeatedly requesting resources that do not exist, the implicitly generated resource nodes are not stored in the *ACResources* tree. Rather, they are created and given back to the caller who is expected to free the *ACResource* instance after using it. This helps mitigate an out-of-memory condition but comes at the cost of having to re-create implicit nodes on each request.

Trust Levels

Table C-7 lists the trust levels and describes privilege levels that are associated with application entry points and assets.

Table C-7 Trust Levels

ID	Name	Description
1	The program using the API	The process identity of the program that uses the library is the direct user of the API.
2	A user with unknown access to a resource	The process using the library passes a user's request for access to resources to the API. The user might have unknown access to the resource.
3	A user with read access to a resource	The process using the library passes a user's request for access to resources to the API. The user might have read access to the resource.
4	A user with write access to a resource	The process using the library passes a user's request for access to resources to the API. The user might have write access to the resource.

Entry Points

Table C-8 lists the entry points and describes the interfaces through which external entities can interact with the component, either via direct interaction or by indirectly supplying the component with data.

Table C-8 Entry Points

ID	Name	Description	Trust Level
1	*ACResources*	A class that represents a tree of resources. Child resource nodes in the tree inherit the access control list of their parent by default.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource
1.1	*Add*	Adds a resource to the tree at the specified path.	(1) The program using the API
1.2	*Remove*	Removes the resource at the specified path.	(1) The program using the API
1.3	*Get*	Returns the resource at the specified path.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource
1.4	*GetRoot*	Gets the root resource.	(1) The program using the API
1.5	*GetChildren*	Gets the child resources at the specified path.	(1) The program using the API
1.6	*GetParent*	Gets the parent resource of the specified path.	(1) The program using the API

Table C-8 Entry Points

ID	Name	Description	Trust Level
1.7	*SetImplicitMode*	Determines whether Implicit mode is used. If Implicit mode is disabled (in other words, if Explicit mode is used), only resources explicitly added to the tree will be used. If Implicit mode is enabled, the class will interpolate resources and automatically generate resource instances that have the access control list of their closest ancestor. For example, in Implicit mode, a user can add the following nodes: \Node1 \Node1\SubNode1\SubNode2 The collection will autogenerate the SubNode1 node with the access control list of Node1. In addition, if the user requests a resource such as \Node1\SubNode1\SubNode2 \SubNode3, the class will autogenerate SubNode3 with the access control list of SubNode2.	(1) The program using the API
1.8	*SetPathResolver*	Sets an optional path resolver that maps paths supplied to the class from an arbitrary form to the normalized form used by the *ACResources* class.	(1) The program using the API
2	*ACResource*	A resource that can have an access control list applied to it.	(1) The program using the API
2.1	*GetAcl*	Gets the access control list for the resource.	(1) The program using the API
2.2	*SetAcl*	Sets the access control list for the resource.	(1) The program using the API
3	*ACGroups*	A class that represents all groups of users. Resources can be restricted by groups, users, or both.	(1) The program using the API
3.1	*Add*	Adds a group to the list.	(1) The program using the API
3.2	*Get*	Gets a group from the list.	(1) The program using the API
3.3	*Remove*	Removes a group from the list.	(1) The program using the API

Table C-8 Entry Points

ID	Name	Description	Trust Level
4	*ACGroup*	A group of users. This class is used to simplify applying access control to a resource by grouping users with the same access rights.	(1) The program using the API
4.1	*AddMember*	Adds a user to the group.	(1) The program using the API
4.2	*GetMembers*	Returns a list of users in the group.	(1) The program using the API
4.3	*RemoveMember*	Removes a user from the group.	(1) The program using the API
5	*ACUsers*	A class that represents all users.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource
5.1	*Add*	Adds a user to the list of users.	(1) The program using the API
5.2	*Get*	Retrieves a user based on login name.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource
5.3	*Remove*	Removes a user from the list.	(1) The program using the API
6	*ACUser*	A user that can be granted access to a resource.	(1) The program using the API
6.1	*GetGroup-Membership*	Returns a list of groups the user belongs to.	(1) The program using the API
7	*ACCheck*	A class with static members that will return the access rights for a user on a specified resource.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource

Table C-8 Entry Points

ID	Name	Description	Trust Level
7.1	*CheckAccess*	Determines whether a user has read or write access to the specified resource.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource
8	*IACPathResolver*	An interface that normalizes a given string to its canonical form so that the *ACResources* class can map arbitrary string-based paths to its internal resource path.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource
8.1	*Resolve*	Takes a string path and returns a normalized string path in the form that is used by the *ACResources* class.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource
8.2	*ACADatumOS-FilePathResolver*	A class that translates file paths on A. Datum Operating System 1.0 to the *ACResources* normalized resource path.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource
8.2.1	*Resolve*	Takes a string path of the specified file and returns a normalized string path in the form that is used by the *ACResources* class.	(1) The program using the API (2) A user with unknown access to a resource (3) A user with read access to a resource (4) A user with write access to a resource
9	*ACAcl*	Represents the access control list for a resource.	(1) The program using the API

Table C-8 Entry Points

ID	Name	Description	Trust Level
9.1	*AddGroup*	Adds a group with read and/or write access.	(1) The program using the API
9.2	*AddUser*	Adds a user with read and/or write access.	(1) The program using the API
9.3	*RemoveGroup*	Removes a group.	(1) The program using the API
9.4	*RemoveUser*	Removes a user.	(1) The program using the API
9.5	*GetGroups*	Gets the groups with access.	(1) The program using the API
9.6	*GetUsers*	Gets the users with access.	(1) The program using the API
9.7	*CheckGroup-Access*	Checks whether a group has read or write access.	(1) The program using the API
9.8	*CheckUser-Access*	Checks whether a user has read or write access.	(1) The program using the API

Assets

Table C-9 describes the data or functionality—the assets—that the component needs to protect. The table lists the minimum access category (or trust level) that should be granted access to the resource.

Table C-9 Assets

ID	Name	Description	Trust Level
8	Process	Assets that relate to the process that the library is running in.	None
8.6	Ability to execute arbitrary code as the identity of the process	The library executes in the process space of the program using it. Buffer overflows and similar implementation issues in the library could allow a user to execute arbitrary code.	(1) The program using the API
8.1	Availability	The access control library should not cause performance degradation to the process. Additionally, users should not be able to crash the process through the access control library.	(1) The program using the API
9	Library	Assets that relate specifically to the library.	None
9.1	Read access to resources	The ability to read from a resource.	(3) A user with read access to a resource

Table C-9 Assets

ID	Name	Description	Trust Level
9.2	Write access to resources	The ability to write to a resource.	(4) A user with write access to a resource
9.3	Group membership	A user in a group is granted the access that the group is granted.	(1) The program using the API
9.4	ACL entries for resources	Access control lists are used to define access for a resource. They should not be tampered with.	(1) The program using the API
9.5	Accuracy of the normalized resource path	The path must be normalized in the same way that it is normalized by any underlying system, such as the file system.	(1) The program using the API
9.6	Path discovery	A user who does not have access to a resource should not be able to determine whether that resource exists in the resource tree.	(3) A user with read access to a resource (4) A user with write access to a resource
9.7	Audit data	Knowing who attempts to access a resource is important to track both legitimate and illegitimate use.	(1) The program using the API

Data Flow Diagrams

The A. Datum Access Control API application uses two data flow diagrams (DFDs) in its threat model. Figure C-1 shows the context diagram, and Figure C-2 shows the Level 0 diagram. The context diagram shows the library being used by an application. The Level 0 diagram shows the data flow for an access check on a resource.

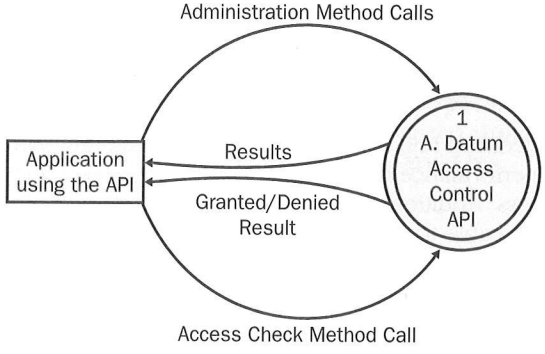


Figure C-1 Context diagram.

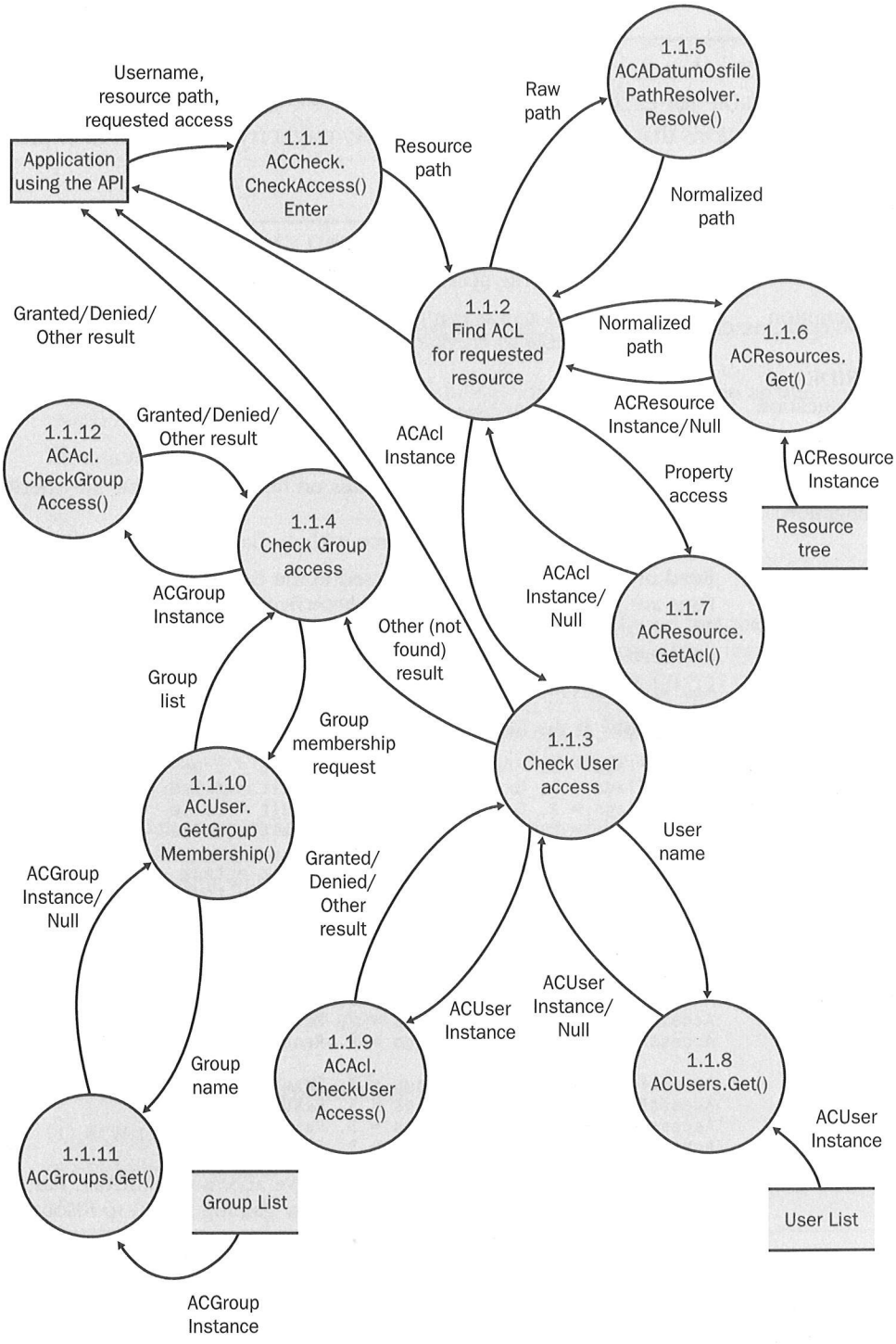


Figure C-2 Level 0 diagram.

Threats

The threats to the application are listed in this section in a series of tables—one table for each threat. These threats do not imply vulnerabilities. Rather, they are actions that a malicious external entity might try to perform to exploit the system.

Table C-10 Threat: Gaining Write Access

ID	1
Name	Adversary gains write access to a resource that he only has read access to
Description	A user with read access can try to write to the resource, changing data that he should not be able to change.
STRIDE classification	■ Tampering ■ Elevation of privilege
Mitigated?	Yes
Known mitigation	See corresponding investigation notes on how access rights are checked. Related external security notes: (3) Requesting access to a resource with no flags set....

Investigation notes	Read or write access are flags passed in the call to *ACCheck.CheckAccess*. These flags are passed to calls to *ACAcl.CheckGroupAccess* and *ACAcl.CheckUserAccess*. The algorithm for testing access is as follows (where a read flag is 1, write flag is 2, and read/write is 3):
	`boolean AccessResult = !(AccessFlags & ~(GrantedFlags));`
	This results in the following table of possibilities:
	`AccessFlags = 0, GrantedFlags = 0, Result = true`
	`AccessFlags = 1, GrantedFlags = 0, Result = false`
	`AccessFlags = 2, GrantedFlags = 0, Result = false`
	`AccessFlags = 3, GrantedFlags = 0, Result = false`
	
	`AccessFlags = 0, GrantedFlags = 1, Result = true`
	`AccessFlags = 1, GrantedFlags = 1, Result = true`
	`AccessFlags = 2, GrantedFlags = 1, Result = false`
	`AccessFlags = 3, GrantedFlags = 1, Result = false`
	
	`AccessFlags = 0, GrantedFlags = 2, Result = true`
	`AccessFlags = 1, GrantedFlags = 2, Result = false`
	`AccessFlags = 2, GrantedFlags = 2, Result = true`
	`AccessFlags = 3, GrantedFlags = 2, Result = false`
	
	`AccessFlags = 0, GrantedFlags = 3, Result = true`
	`AccessFlags = 1, GrantedFlags = 3, Result = true`
	`AccessFlags = 2, GrantedFlags = 3, Result = true`
	`AccessFlags = 3, GrantedFlags = 3, Result = true`
	This table shows that the check for positive access is sufficient. For more details, see the threat of an adversary with a Deny gaining access to resources (Threat 11, outlined in Table C-19).
	Furthermore, the calling application must ensure that it does not grant access to a resource that was not checked through a call to *ACCheck.CheckAccess*. Callers should ensure that when they open the resource, they open it using only the access flags that were checked in the call to *ACCheck.CheckAccess*.

Table C-10 Threat: Gaining Write Access

Entry points	(9) *ACAcl*
Assets	(9.2) Write access to resources
Threat Tree	None

Table C-11 Threat: Malformed Path Data

ID	2
Name	Adversary supplies malicious or malformed data as a path to the library, targeting the path parsing code
Description	An attacker could use malicious input such as long resource paths in an attempt to overflow a character buffer or integer length field.
STRIDE classification	Elevation of privilege
Mitigated?	No
Known mitigation	None
Investigation notes	A code review of *ACADatumOSFilePathResolver.Resolve* found that the buffer allocated for the file path that is passed in did not include space for a terminating (null) character. Because of allocation alignment, this omission was not caught during testing. However, if a path with a length that is a multiple of the alignment is provided (8 bytes on A. Datum Operating System 1.0), when copied to the buffer, that path will overwrite one character's worth of memory adjacent to the buffer with a null character. Furthermore, the path length is calculated by using an unsigned short integer (16 bits). This means that a path longer than 64 k will cause integer rollover so that the calculated length will actually be: ActualLength MOD 65536 Thus, for an actual length of 65,537 characters, only 1 character will be allocated. When the path is copied, arbitrary memory beyond the one-character allocation will be overwritten with the input path.
Entry points	(8) *IACPathResolver* (8.2) *ACADatumOSFilePathResolver* (1) *ACResources*
Assets	(8.6) Ability to execute arbitrary code as the identity of the process
Threat tree	None

Table C-12 Threat: Path Normalization Attack

ID	3
Name	Adversary supplies a path that normalizes incorrectly, allowing access to an incorrect resource
Description	If a malicious user finds a discrepancy in the way the implementation of *IACPathResolver* normalizes paths compared to the underlying system, she might be able to trick the Access Control API into granting her access.
STRIDE classification	Elevation of privilege
Mitigated?	Yes
Known mitigation	The path normalization for the supplied *ACADatumOSFilePathResolver* is directly copied from AC Datum Operating System 1.0. These routines are kept in sync. For other mitigations, see the mitigating external security notes in this table. *Related use scenarios:* (1) *ACADatumOSFilePathResolver* is an implementation of *IACPathResolver* that resolves path names on A. Datum Operating System 1.0; it will not work for other operating systems. Using it on other operating systems could result in incorrect mapping of path names to the A. Datum Access Control API canonical form. *Related external dependencies:* (1) The *ACADatumOSFilePathResolver* depends on file paths in A. Datum Operating System 1.0 to be resolved in the same manner as implemented in this class. If the file paths resolve differently in the operating system, the mapping that the *ACADatumOSFilePathResolver* performs could result in an incorrect resource being checked for access. In the worst case, this could allow an adversary to access a file he would not normally have access to. *Related external security notes:* (1) *ACADatumOSFilePathResolver* is intended to be used on A. Datum Operating System 1.0 only. Using it on other platforms is not supported and could cause errors. (4) Users who implement the *IACPathResolver* interface should be sure to use the same parsing and normalization as the underlying system that stores the resource. This will prevent inconsistencies between path resolution in the resolver and the underlying system that could result in access checks being done on one resource in the Access Control API and another resource being opened by the underlying system.
Investigation notes	None
Entry points	(8.2) *ACADatumOSFilePathResolver*
Assets	(9.5) Accuracy of the normalized resource path (9.1) Read access to resources (9.2) Write access to resources
Threat tree	None

Table C-13 Threat: Spoofing Another User

ID	4
Name	Adversary spoofs user to gain another user's access rights
Description	A malicious user might try to impersonate another user to gain that user's access rights.
STRIDE classification	Spoofing
Mitigated?	Yes
Known mitigation	The application using the API must supply the correct username in the call to *ACCheck.CheckAccess*. Usernames in the Access Control API are compared by using a literal string comparison and are case sensitive.
Investigation notes	None
Entry points	(5) *ACUsers*
Assets	(9.1) Read access to resources
	(9.2) Write access to resources
Threat tree	None

Table C-14 Threat: Spoofing Group Membership

ID	5
Name	Adversary spoofs group membership to gain that group's access rights
Description	A malicious user might try to spoof group membership to gain the access rights of the group.
STRIDE classification	Spoofing
Mitigated?	Yes
Known mitigation	The application using the Access Control API must configure and maintain group membership. This is done directly via the API and is not persisted to disk or any other store that could be tampered with by the adversary.
Investigation notes	None
Entry points	(3) *ACGroups*
Assets	(9.3) Group membership
	(9.1) Read access to resources
	(9.2) Write access to resources
Threat tree	None

Table C-15 Threat: Causing a Denial of Service by Providing a Complex Pathname

ID	6
Name	Adversary causes denial of service by providing a complicated pathname
Description	An attacker might use a resource path that requires greater-than-average processing or memory resources to parse, causing a denial of service to other users.
STRIDE classification	Denial of service
Mitigated?	Yes
Known mitigation	See investigation notes.
Investigation notes	A code review and performance test of the supplied implementation of the *IACPathResolver* (*ACADatumOSFilePathResolver*) did not find any compute-intensive operations. A code review and test of the *ACResources.Get* API showed that the implementation used a recursive function call each time a path separator character was encountered. If a long enough path comprising path separator characters is supplied, the application could run out of memory on the stack, causing a fault. Because the fault is not handled, it crashes the application.
Entry points	(8) *IACPathResolver* (1.3) *Get*
Assets	(8.1) Availability
Threat tree	None

Table C-16 Threat: Denial of Service Attack

ID	7
Name	Adversary causes denial of service by repeatedly requesting implicitly generated nodes
Description	Because implicitly generated nodes are not persisted, an attacker could try repeatedly requesting access to one of these nodes. This could result in a denial of service if the implicitly generated nodes are not created efficiently.
STRIDE classification	Denial of service
Mitigated?	Yes
Known mitigation	The Access Control API test team has tests that cover this scenario. Performance in such cases does not degrade significantly.
Investigation notes	None
Entry points	(1) *ACResources*
Assets	(8.1) Availability
Threat tree	None

Table C-17 Threat: Determining if a Path Exists

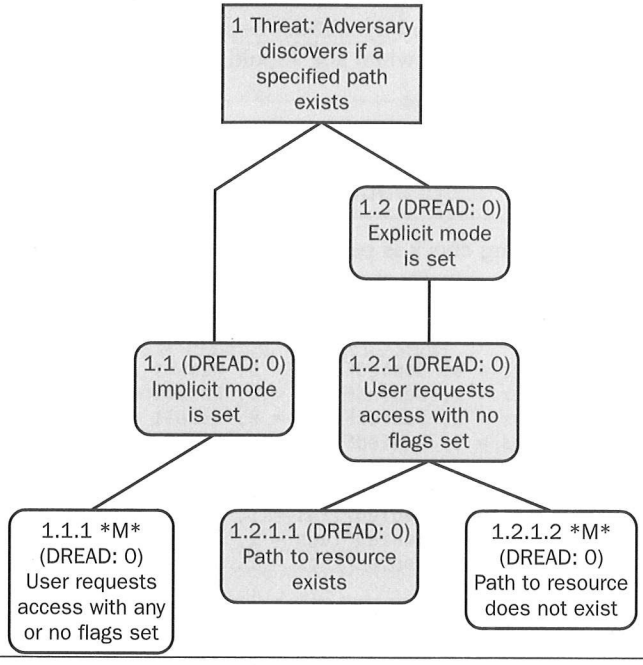
ID	8
Name	Adversary determines whether a specified path exists
Description	The adversary might want to determine whether a specific path that he should not have access to exists. This can include finding out whether the path exists in the resource tree or in the underlying system.
STRIDE classification	Information disclosure
Mitigated?	Partially
Known mitigation	This is partially mitigated because the Access Control API never interacts with the underlying system that stores the resources.
Investigation notes	None
Entry points	(1) *ACResources*
Assets	(9.6) Path discovery
Threat tree	

Table C-18 Threat: Access Without Auditing

ID	9
Name	Adversary accesses a resource without being logged or audited
Description	An adversary who can attempt to access a resource without being logged or audited could perform an attack without the administrator knowing or being able to take action against the attack.

Table C-18 Threat: Access Without Auditing

STRIDE classification	Repudiation
Mitigated?	No
Known mitigation	None
Investigation notes	None
Entry points	(7) *ACCheck*
Assets	(9.7) Audit data
Threat tree	None

Table C-19 Threat: Bypassing a Deny Access Control Entry

ID	10
Name	Adversary with a Deny gains access to a resource
Description	Denies have precedence over Grants. In complex configurations of Grants and Denies, the adversary should not be able to take advantage of a logic error granting her access when she should be denied.
STRIDE classification	Elevation of privilege
Mitigated?	No
Known mitigation	None
Investigation notes	The following check is performed for the read/write access on the *DenyFlags:*

```
boolean DenyResult = (AccessFlags & DeniedFlags);
```

This results in the following table of possibilities:

```
AccessFlags = 0, DeniedFlags = 0, Result = false
AccessFlags = 1, DeniedFlags = 0, Result = false
AccessFlags = 2, DeniedFlags = 0, Result = false
AccessFlags = 3, DeniedFlags = 0, Result = false

AccessFlags = 0, DeniedFlags = 1, Result = false
AccessFlags = 1, DeniedFlags = 1, Result = true
AccessFlags = 2, DeniedFlags = 1, Result = false
AccessFlags = 3, DeniedFlags = 1, Result = true

AccessFlags = 0, DeniedFlags = 2, Result = false
AccessFlags = 1, DeniedFlags = 2, Result = false
AccessFlags = 2, DeniedFlags = 2, Result = true
AccessFlags = 3, DeniedFlags = 2, Result = true

AccessFlags = 0, DeniedFlags = 3, Result = false
AccessFlags = 1, DeniedFlags = 3, Result = true
AccessFlags = 2, DeniedFlags = 3, Result = true
AccessFlags = 3, DeniedFlags = 3, Result = true
```

This table shows that the flag comparison is correct. However, the way that the user/group Deny is calculated is incorrect. (See the corresponding DFD for *ACCheck.CheckAccess* and the threat tree in this table.)

Table C-19 Threat: Bypassing a Deny Access Control Entry

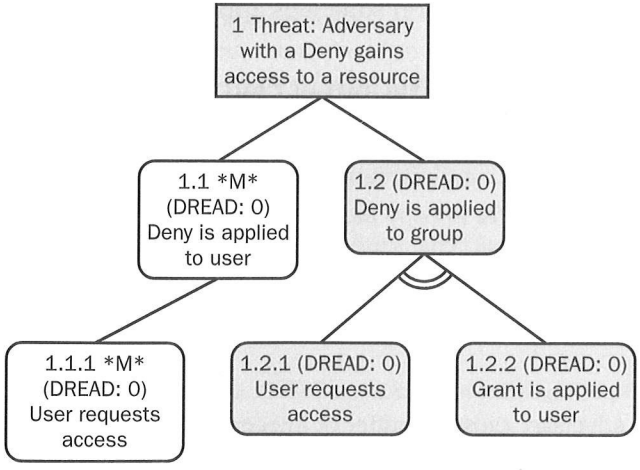
Entry points	(7) *ACCheck*
Assets	(9.1) Read access to resources (9.2) Write access to resources
Threat tree	

Table C-20 Threat: Denial of Service by Malicious Input

ID	11
Name	Adversary causes denial of service by forcing a null pointer dereference
Description	A user might try requesting nonexistent resources or other items to cause a null dereference, crashing the application.
STRIDE classification	Denial of service
Mitigated?	Yes
Known mitigation	A code review for this type of error was performed, and no problems were found.
Investigation notes	A. Datum's standard code review sessions look for null-dereference errors. All code in the Access Control API was reviewed to this standard, and no errors were found.
Entry points	(1) *ACResources*
Assets	(8.1) Availability
Threat tree	None

Vulnerabilities

The known vulnerabilities to the Access Control API system are listed in a series of tables in this section—one table for each vulnerability. Each table includes the risk associated with not fixing the vulnerability so that the threat modeling team can choose mitigation strategies appropriately.

Table C-21 Vulnerability: Buffer Overflow in Path Processing

ID	1
Name	Null-character overflow in path buffer allocation
Description	When the buffer to copy the supplied path is calculated, it does not include the terminating (null) character. This results in a one-character overflow when the path is copied (because the null character is copied to the destination buffer).
STRIDE classification	Elevation of privilege
DREAD rating	10
Corresponding threat	2: Adversary causes a buffer or integer overflow by providing a path of a overly long length
Bug	6654

Table C-22 Vulnerability: Integer Overflow in Path Processing

ID	2
Name	Integer overflow in path length computation for paths greater than 65,535 characters
Description	The path length is calculated by using an unsigned short integer (16 bits). This means that a path longer than 64 k will cause integer rollover so that the calculated length will actually be ActualLength MOD 65536 Thus, for an actual length of 65,537 characters, only 1 character will be allocated. When the path is copied, arbitrary memory beyond the one-character allocation will be overwritten with the input path.
STRIDE classification	Elevation of privilege
DREAD rating	10
Corresponding threat	2: Adversary causes a buffer or integer overflow by providing an overly long path
Bug	6656

Table C-23 Vulnerability: Denial of Service Through Path Processing

ID	3
Name	Long path of path separator characters supplied to *ACResources.Get* can cause the application to crash
Description	The *ACResources.Get* API uses a recursive function call each time a path separator character is encountered. If a long enough path comprising path separator characters is supplied, the application could run out of memory on the stack, causing a fault. Because the fault is not handled, it crashes the application. This recursive algorithm should be removed. Usage information culled from A. Datum's customer activity shows that most use limits path lengths, typically to less than 4096 characters. This number is typically less than that required to run out of space for stack frames; therefore, the affected number of users is low.
STRIDE classification	Denial of service
DREAD rating	9
Corresponding threat	6: Adversary causes denial of service by providing complicated pathname
Bug	6657

Table C-24 Vulnerability: Determining if a Path Exists

ID	4
Name	Requesting no access flags for a resource that exists when Explicit mode is set
Description	If no access flags are set when access is requested for a resource that exists and Explicit mode is set, the API will always return true. If the resource does not exist, it will return an error code indicating that the resource was not found. This behavior should be changed so that it does not allow an adversary to determine whether a path exists in the resource tree.
STRIDE classification	Information disclosure
DREAD rating	4.8
Corresponding threat	8: Adversary discovers whether a specified path exists
Bug	6670

Table C-25 Vulnerability: No Auditing on Requests

ID	5
Name	No logging or auditing occurs in the Access Control API
Description	The Access Control API currently has no logging or auditing facilities. Instead, the application using the host must log data. However, it is the responsibility of the Access Control API to promote users creating secure, robust systems.
STRIDE classification	Repudiation
DREAD rating	5
Corresponding threat	9: Adversary accesses a resource without being logged or audited
Bug	6673

Table C-26 Vulnerability: Deny Access Control Entries Handled Incorrectly

ID	6
Name	Deny applied to a group and Grant applied to user incorrectly calculates access
Description	If a user is granted access to a resource, that access will override the check made by *ACCheck.CheckAccess* and return access granted. (See the corresponding DFD.) If that user is a member of a group that is denied access, the check for the group will never occur. Because Denies must take precedence, regardless of whether they are applied to a user or to a group, this results in an exploitable condition.
STRIDE classification	Elevation of privilege
DREAD rating	7.2
Corresponding threat	10: Adversary with a Deny gains access to a resource previously inaccessible to him
Bug	6679