Characterizing Intrusion Tolerant Systems Using a State Transition Model [*]

Katerina Goseva-Popstojanova[1], Feiyi Wang[2], Rong Wang[2],
Fengmin Gong[2], Kalyanaraman Vaidyanathan[1], Kishor Trivedi[1], Balamurugan Muthusamy[3†]

[1] *Department of ECE, Duke University, Box 90294, Durham, NC 27708*
[2] *Advanced Networking Research Group, MCNC, Research Triangle Park, NC 27709*
[3] *Vitesse Corp., Camarillo, CA 93012*

Abstract

Intrusion detection and response research has so far mostly concentrated on known and well-defined attacks. We believe that this narrow focus of attacks accounts for both the successes and limitation of commercial intrusion detection systems (IDS). Intrusion tolerance, on the other hand, is inherently tied to functions and services that require protection. This paper presents a state transition model to describe the dynamic behavior of intrusion tolerant systems. This model provides a framework from which we can define the vulnerability and the threat set to be addressed. We also show how this model helps us to describe both known and unknown security exploits by focusing on impacts rather than specific attack procedures. By going through the exercise of mapping known vulnerabilities to this transition model, we identify a reasonably complete fault space that should be considered in a general intrusion tolerant system.

1. Introduction

Most of the intrusion detection and response approaches to date have focused on the specific manifestation of attacks [2, 3, 5, 10]. Just by focusing our attention on the intrusion attacks themselves, we cannot expect to develop a general protection mechanism because all attacks are not well-defined and there are always unknown attacks. Intrusion tolerance, on the other hand, is inherently tied to the functions and services that require protection (i.e. to be made intrusion tolerant). It is this focus that makes intrusion tolerance a promising approach to build our defense from.

Intrusion tolerance research should leverage results from the fault tolerance community to the fullest extent possible. Fault tolerant designs are built-in in almost every aspect of our critical modern infrastructure, e.g. air traffic control and power grid control systems. However, as described below, there are significant challenges in applying fault tolerance approaches to intrusion tolerance.

- Fault tolerance techniques have mostly been focused on accidental faults and malicious faults planted at the design or implementation stages. This focus allows us to make some reasonable assumptions regarding predictable fault behaviors. Active intrusions, manifested as compromised system components whose behavior is under complete malicious control, make the fault behavior very unpredictable.
- Active intrusions also introduce attacks from outside the system, which have not been considered in the traditional fault tolerance systems.
- Existing fault tolerance work has mostly focused on well-defined hardware or software modules whose fault modes are relatively easy to define. In large distributed service infrastructures (e.g. a database-driven web server), each of the components has complex functions which makes the definition of fault modes more difficult.
- There is a wealth of fault tolerance techniques aimed at building fault tolerance systems from scratch. Some more recent efforts have also begun to consider the protection of COTS systems. While striving to make our new systems intrusion-tolerant, it is critical to

[*] This work is sponsored by the U.S. Department of Defense Advanced Research Projects Agency (DARPA) under contract N66001-00-C-8057 from the Space and Naval Warfare Systems Center - San Diego (SPAWARSYSCEN). The views, opinions and findings contained in this paper are those of the authors and should not be construed as official DARPA or SPAWARSYSCEN's positions, policy or decision.

[†] This work was done while the author was with MCNC.

make the existing information infrastructure built from COTS components more intrusion tolerant at the system level. Therefore, it is highly desirable to develop technologies for intrusion tolerance that are useful both for hardening existing infrastructures and for building better new ones.

The DARPA-funded research project, named SITAR, is developing a scalable intrusion tolerance architecture for distributed services. As a first step in this research, we have developed a state transition model for describing a general intrusion tolerant system.

We focus on a generic class of services (network-distributed services built from COTS components) as the target for protection. This target presents us with enough challenging problems to solve while remaining concrete enough for us to explore specific intrusion-tolerance issues associated with it. The emphasis of the proposed architecture and solution is on continuity of operation since the security precautions cannot guarantee that the system will not be penetrated and compromised. Thus, our goal is to design a survivable system whose components collectively accomplish their mission even under attack and despite active intrusions. With respect to system survivability, distinctions among attacks, failures, and accidents are less important than the event's impact. Effects are more important than causes because a system will have to deal with and survive an adverse effect long before a determination is made as to whether the cause was an attack, a failure, or an accident.

This paper presents the model we have developed so far and discusses how some of the known security exploits fit into this model, and how this model helps us to deal with unknown attacks. The rest of the paper is organized as follows. Section 2 presents the state transition model in detail. Section 3 describes several known exploits, how the target system maps to the state transition model, and how different categories of exploits manifest in terms of the model. We summarize the paper and discuss future research steps in Section 4.

2. Basic model

Figure 1 depicts the state transition model which we propose as a framework for describing dynamic behavior of an intrusion tolerant system. The system enables multiple intrusion tolerance strategies to exist and supports different levels of security requirements. State transition model represents the system behavior for a specific attack and given system configuration that depends on the actual security requirements.

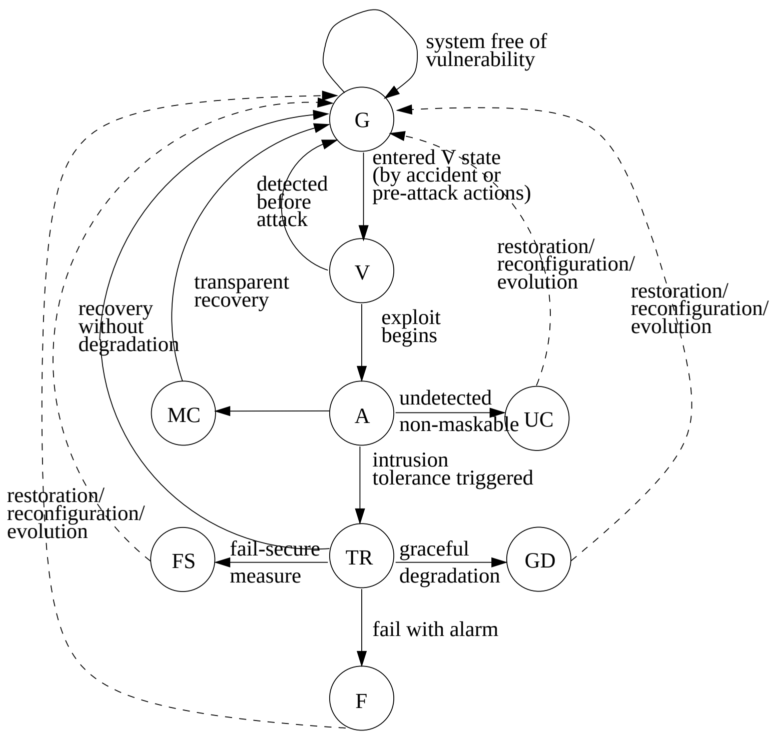
The system is in the vulnerable state V if it enables a user to read information without authorization, modify information without authorization, or grant or deny an entity access to a resource without authorization. Here "without authorization" means in violation of the system's security policy. A vulnerability (also called a flaw or a hole) is the property of the system, its attendant software and/or hardware, or its administrative procedures, that causes it to enter a vulnerable state. Exploiting a vulnerability means that a system is in a vulnerable state and a user (called an attacker) reads or writes the information without authorization, or compromises the system to grant or deny service without authorization. The system enters an active attack state A when a vulnerability is successfully exploited.

Traditional computer security leads to the design of systems that rely on resistance to attacks, that is, hardening for protection. Thus, the system typically oscillates between good G and vulnerable V states [13]. System management seeks to prevent the system from becoming vulnerable or to reduce the time it remains in the vulnerable state. Current strategies for resistance include the use of authentication, access control, encryption, firewalls, proxy servers, strong configuration management, dispersion of data and application of system upgrades for known vulnerabilities.

If the strategies for resistance fail, the system is brought into the vulnerable state during the penetration and exploration phases of an attack. If the vulnerability is exploited successfully, the system enters the active attack state and damage may follow. Thus, the ability of a system to react during an active intrusion is central to its capacity to survive an attack that cannot be completely repelled. It means that intrusion tolerance picks up where intrusion resistance leaves off. The four phases that form the basis for all fault tolerance techniques are error detection, damage assessment, error recovery, and fault treatment and continued service [11]. These can and should be the basis for the design and implementation of an intrusion tolerant system.

Strategies for detecting attacks and assessment of damage include intrusion detection (i.e. anomaly based and signature based detection), logging, and auditing. If the probing that precedes the attack is detected, the system will stay in the good state. The other possibility is to detect the penetration and exploration phases of an attack and bring the system from the vulnerable state back to the good state. Otherwise, the system might enter the active attack state. Traditionally, the resistance and detection of attacks receive most of the attention, and once active attack state is entered damage may follow with little to stand in the way. Thus, during the exploitation phase of intrusion it is critical to assess the damage and try the recovery. The strategies for recovery include the use of redundancy for critical information and services, incorporation of backup systems in isolation from network, isolation of damage, ability to operate with reduced services or reduced user community.

The best possible case is when there is enough redundancy to enable the delivery of error-free service and bring


system free of vulnerability
G
entered V state (by accident or pre-attack actions)
detected before attack
V
transparent recovery
recovery without degradation
exploit begins
restoration/ reconfiguration/ evolution
restoration/ reconfiguration/ evolution
MC
A
undetected non-maskable
UC
intrusion tolerance triggered
restoration/ reconfiguration/ evolution
FS
fail-secure measure
TR
graceful degradation
GD
fail with alarm
F

G	good state	TR	triage state
V	vulnerable state	FS	fail-secure state
A	active attack state	GD	graceful degradation state
MC	masked compromised state	F	failed state
UC	undetected compromised state		

Figure 1. A state transition diagram for intrusion tolerant system

the system back to the good state by masking the attack's impact (MC). This is known as error compensation or error masking in fault tolerance systems. The worst possible case is when the intrusion tolerance strategies fail to recognize the active attack state and limit the damage, leading to an undetected compromised state UC, without any service assurance.

When an active attack in exploitation phase is detected, the system will enter the triage state TR attempting to recover or limit the damage. Ideally, of course, the system should have in place some measures for eliminating the impacts produced by an attack, providing successful restoration of a good state. However, restoration of the good state may not necessarily be an appropriate, cost-effective, or even a feasible recovery technique. In this case, the system could attempt to limit the extent of damage while maintaining the essential services. Essential services are defined as the functions of the system that must be maintained to meet the system requirements even when the environment is hostile, or when failures or accidents occur that threaten the system [12]. Of course, there is no "one size fits all" solution. In intrusion tolerance the impacts are more important than the causes. If the aim is to protect the system from denial of service attack from external entities, the system should enter the graceful degradation state GD, maintaining only essential services. However, if the aim is to protect confidentiality or data integrity the system must be made to stop functioning. This is called fail-secure state FS, analogous to fail-safe state in fault tolerance. If all of the above strategies fail then the system enters the failed state, F, and signals an alarm.

Recovering the full services after an attack and returning to the good state by manual intervention is represented by transitions denoted with dashed lines. Although the system may have returned to a good state, techniques such as reconfiguration or evolution of the system may still be required to reduce the effectiveness of future attacks. This phase can be considered analogous to fault treatment and continued service phase in fault tolerance.

3. Case studies

In this section, we discuss several vulnerability case studies and map them to our previously-discussed state transition model. There has been extensive work reported on categorization of software vulnerabilities, system errors and flaws, and intrusions [4, 7, 8]. These studies cover many attributes of vulnerabilities, though each with a different emphasis. For the development of intrusion tolerance capabilities, we need to focus on the impact of the intrusions exploiting these vulnerabilities. Most importantly, we concentrate on observable impact that affords us opportunities for detection and providing tolerance. We consider the following classes of vulnerabilities based on their impact:

Compromise of confidentiality: These attacks violate the confidentiality requirement for sensitive data.

Compromise of data integrity: These attacks mainly result in corruption of sensitive data.

Compromise of user/client authentication: These attacks cause breach in the normal authentication process between the client and the server.

DoS from external entities: These attacks are mainly aimed at disrupting normal services by directly consuming large amount of service resources such as network access bandwidth and CPU cycles.

DoS by compromising internal entities: These attacks achieve the disruption of service through a secondary effect of a compromise in the commercial-off-the-shelf (COTS) servers.

It should be pointed out that, while we present a set of known vulnerabilities and exploits in this section, the study of these known vulnerabilities only serve the purpose of developing understanding towards a general intrusion tolerance system. The intrusion tolerance system emerging from this study will be able to deal with previously-unknown attacks as long as these attacks produce similar impact on our services. Through the exercise of mapping the currently-known vulnerabilities to the proposed state transition model, we have succeeded in identifying a fairly complete state space for an intrusion tolerant system. We are able to delineate transitions among these states that represent a variety of opportunities for detecting, recovering from, and tolerating an attack. These transitions afford intrusion protections ranging from prevention, to detection with graceful service degradation, and to fail-secure measures. These transitions also suggest viable functionality mappings to different modules of the SITAR architecture [9]. The following case presentations help to illustrate these points.

3.1. Active Server Page (ASP) vulnerability in IIS 4.0 (Bugtraq ID 167)

One of the sample files shipped with Internet Information Server (IIS) 4.0, `showcode.asp`, is meant for viewing the source code of the sample applications through a web browser. The `showcode.asp` file does not perform adequate security checking and anyone with a web browser can view the contents of any text file on the web server by using the URL: `http://target/msadc/Samples/SELECTOR/showcode.asp?source=/path/filename`. The files that can be viewed this way also include files that are outside of the document root of the web server.

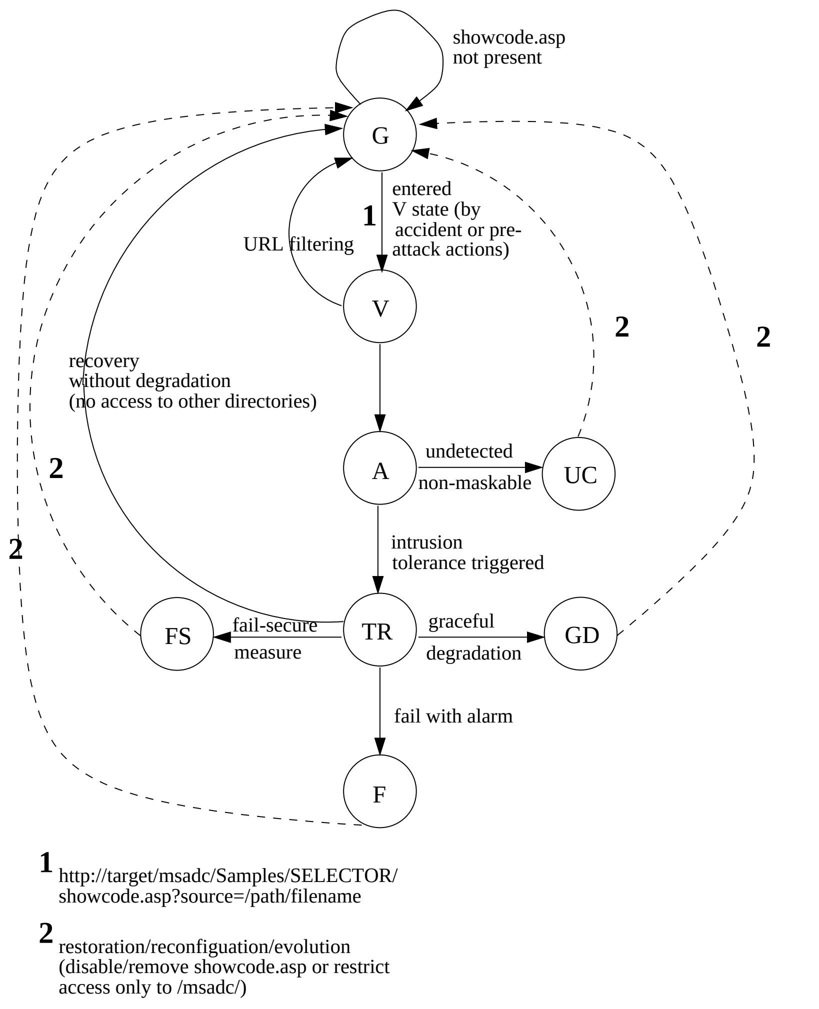
The ASP file is supposed to have a security check which only allows the viewing of the sample files which were in the `/msadc` directory on the system. But the security check does not test for the ".." characters within the URL. The only checking done is if the URL contains the string `/msadc/`. This allows URLs to be created that view, not only files outside of the samples directory, but files anywhere on the entire file system that the web server's document root is on.

For production servers, sample files should never be installed and hence the entire `/msadc/samples` directory should be deleted. However, if `showcode.asp` capability is required on development servers, the file should be modified to also test for URLs with ".." in them and deny those requests.

This vulnerability poses a high security risk, specifically, compromise of confidentiality. Many e-commerce web servers store transaction logs and other customer information such as credit card numbers, shipping addresses, and purchase information in text files on the web server. All these types of data could be accessed by exploiting this vulnerability. Hence the immediate impact is **compromise of confidentiality**.

The mapping of this vulnerability to the state diagram is shown in Figure 2.

- The system is initially in the good state G. If `showcode.asp` is not present, the system stays in state G.
- If `showcode.asp` is present, the attacker brings the system into the vulnerable state V by submitting the URL `http://target/msadc/Samples/SELECTOR/showcode.asp?source=/path/filename`.
- If URL filtering is done to test for "..", then the system goes back to the good state G, from state V. If not, transition "exploit begin" is activated and the attack actually happens in state A.



showcode.asp not present
G
entered V state (by accident or pre-attack actions)
1
URL filtering
V
2
2
recovery without degradation (no access to other directories)
A
undetected
non-maskable
UC
2
2
intrusion tolerance triggered
FS
fail-secure measure
TR
graceful degradation
GD
fail with alarm
F
1 http://target/msadc/Samples/SELECTOR/showcode.asp?source=/path/filename
2 restoration/reconfiguation/evolution (disable/remove showcode.asp or restrict access only to /msadc/)

Figure 2. State transition diagram for ASP vulnerability

- If the attack is successful and goes undetected, the system goes to the undetected compromised state UC.
- If there are intrusion tolerance measures, they are triggered from state A and the system now reaches the state TR. If the fix for the exploit (restriction to all directories except `/msadc`) is present, the system can recover without degradation and can go back to the good state G.
- The system can be taken to the fail-secure state FS (where the system is shut down securely) to limit the damage if the damage was unavoidable.
- If possible, the system can also be taken to the gracefully degraded state GD, where only some services are maintained.
- If the tolerance measures fail in spite of the trigger, the system enters the state F.
- The system is returned to the good state G from states UC, F, FS or GD after restoration/reconfiguration/evolution which includes disabling/removing `showcode.asp` or restricting access only to `/msadc` directory. The next time the same attack happens, the system remains in state G (indicated by a self-loop).

3.2. Common Gateway Interface (CGI) vulnerability in Sambar server (Bugtraq ID 1002)

The Sambar Web/FTP/Proxy Server for Windows NT and 2000 includes the ability to use DOS-style batch programs as CGI scripts. Any batch file used by the server in the `cgi-bin` directory can be used by a remote attacker to run any valid command-line program with Administrator privileges, for example, by providing a URL like `http://target/cgi-bin/hello.bat?&dir+c:\` or

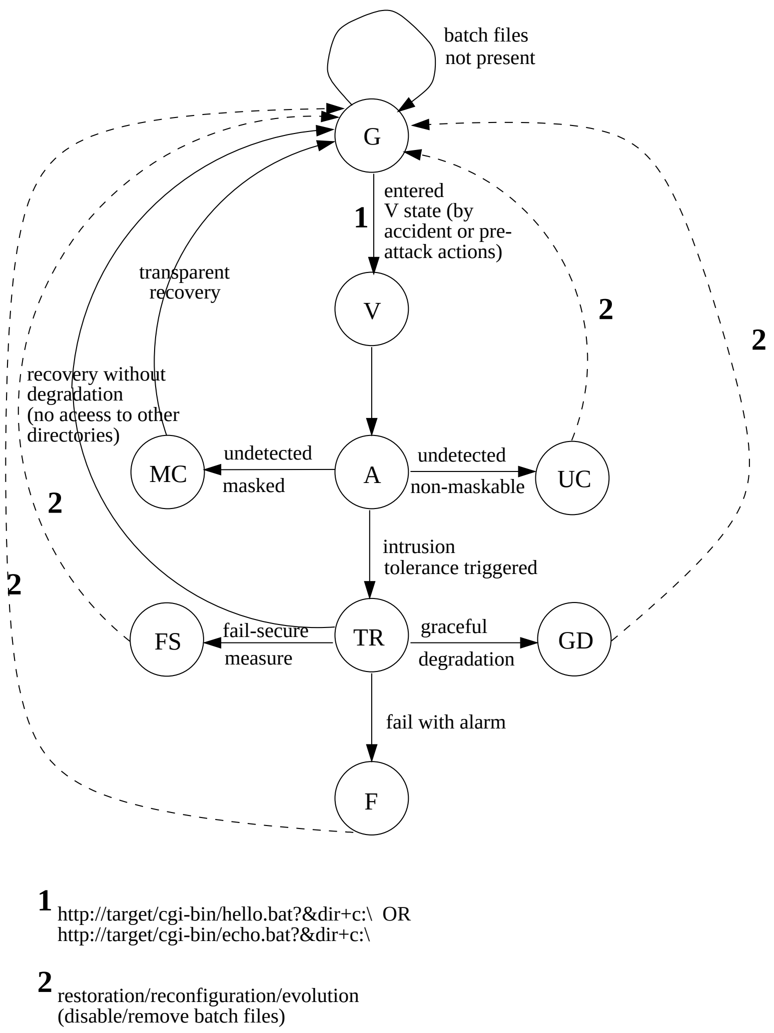

batch files not present
G
entered V state (by accident or pre-attack actions)
1
V
transparent recovery
recovery without degradation (no aceess to other directories)
2
2
2
2
MC
undetected
masked
A
undetected
non-maskable
UC
intrusion tolerance triggered
FS
fail-secure measure
TR
graceful degradation
GD
fail with alarm
F
1 http://target/cgi-bin/hello.bat?&dir+c:\ OR http://target/cgi-bin/echo.bat?&dir+c:\
2 restoration/reconfiguration/evolution (disable/remove batch files)

Figure 3. State transition diagram for CGI vulnerability

`http://target/cgi-bin/echo.bat?&dir+c:\`.

This provides the ability to read, modify, create or delete any file or directory on the system, the ability to create, delete or modify user accounts, etc. Even if the user has not enabled or created any batch files, the software is shipped with two files by default - `hello.bat` and `echo.bat`. The immediate and direct impact of this vulnerability is **compromise of confidentiality and data integrity**. Further indirect impacts could include DoS.

Ideally, the web server should not be shipped with any batch files. However, batch file execution can still be supported and therefore the server can still be vulnerable if batch files are uploaded to the `cgi-bin` directory by any means.

Figure 3 shows the mapping of this vulnerability to the state transition diagram.

- The system is initially in the good state, G. If the batch files are not present, the system stays in state G.

- The attacker brings the system into the vulnerable state V by submitting the URL `http://target/cgi-bin/hello.bat?&dir+c:\`. URL filtering is very difficult in this case, since any batch file can be uploaded even in the absence of `hello.bat` and `echo.bat`. Hence transition “exploit begin” is activated and the attack actually happens in state A.

- If no fix for the exploit is present, the attack can be successful and the system can go to the undetected compromised state UC.

- In some instances (eg. compromise of data integrity), even while a fix is not present, the impact of the attack can be masked (by redundancy) and the system recovers perfectly through the state MC.

- Intrusion tolerance triggers are activated before entering state TR and if fool-proof mechanisms are in place (no access to other directories), the system can go back to the good state G without experiencing degradation.

- The system can be taken to the fail secure state FS (where the system is shut down securely) to limit the damage if the damage was unavoidable.

- If possible, the system can also be taken to the gracefully degraded state GD, where only some services are maintained.

- If the tolerance measures fail in spite of the trigger, the system enters the state F.

- The system is returned to the good state G from states UC, F, FS or GD after restoration/reconfiguration/evolution and for future attacks of this kind, the system is hardened (no batch files present) and hence always remains in the good state, G (indicated by the self loop).

3.3. Sun Java web server bulletin board vulnerability (Bugtraq ID 1600)

The Sun Java Web Server includes two features which when used together can be made to execute arbitrary code at the privilege level of the server. The Web Administration module listens on port 9090 for administrative commands via http. By using the `/servlet/` prefix, it is possible for a remote user to point the servlet `com.sun.server.http.pagecompile.jsp92.JspServlet` to any file in or below the administration web root for compilation and execution.

The server also includes a sample application that provides bulletin board functionality. This application uses the file `board.html` in the web root to store all posted messages. Code can be entered as a posted message through the file `/examples/applications/bboard/bboard_frames.html` and will then be stored as part of `board.html`.

Therefore, it is possible for a remote user to inject JSP code into `board.html`, and then have the server execute it via the Administration module, using the URLs like `http://target:9090/servlet/com.sun.server` and `http.pagecompile.jsp92.JspServlet/board.html`.

The immediate impact of this vulnerability is **compromise of user/client authentication and confidentiality**. Further indirect impacts could include compromise of data integrity and authenticity and DoS.

The mapping of this vulnerability to the state transition diagram is shown in Figure 4.

- Initially, system is in the good state, G.

- The attacker then brings the system to the vulnerable state, V, by injecting code into `board.html` through the URL: `http://target:9090/servlet/com.sun.server`.

- The attacker exploits the vulnerability (executing code) by submitting the URL `http.pagecompile.jsp92.JspServlet/board.html`. Hence transition ”exploit begin” is activated and the attack happens in state A.

- If no fix for the exploit is present, the attack can be successful and the system can go to the undetected compromised state UC.

- In some instances (eg. compromise of data integrity), even though a fix is not present, the impact of the attack can be masked (by redundancy) and the system recovers perfectly through the state MC.

- Intrusion tolerance triggers are activated before the system enters state TR and if a fix is present (execution of the servlets is blocked), the system recovers without any degradation and goes back to the good state G.

- The system can be taken to the fail-secure state FS (where the system is shut down securely) to limit the damage if the damage was unavoidable.

- In some cases, it might also be possible to take the system to the degraded state, GD, gracefully and maintain some services without bringing down the entire system.

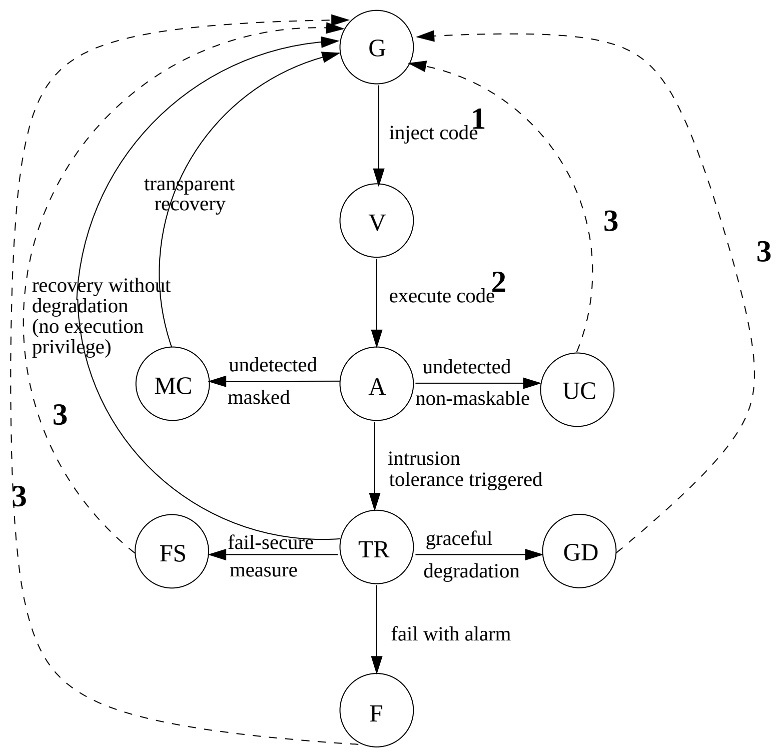
- If the tolerance measures fail in spite of the trigger, the system enters the state F.

- The system is returned to the good state G from states UC, F, FS or GD after restoration/reconfiguration/evolution. The fix for the exploit (no execution privilege in the `bboard` directory) is applied during this procedure and this prevents the same exploit from happening again.

3.4. `'SITE EXEC'` vulnerability in wu-ftpd (Bugtraq ID 1387)

`Wu-ftpd`, developed by Washington University, is a very popular UNIX ftp server program. Recently it was reported [1, 6] that there is an input validation hole in `wu-ftpd` version 2.5.0. This hole could be exploited by using the `'site exec'` command and it could result in root compromise.

The source of the `wu-ftpd` input validation error is due to the fact that the program fails to check the arguments



1 code posted through /examples/applications/bboard/bboard_frames.html

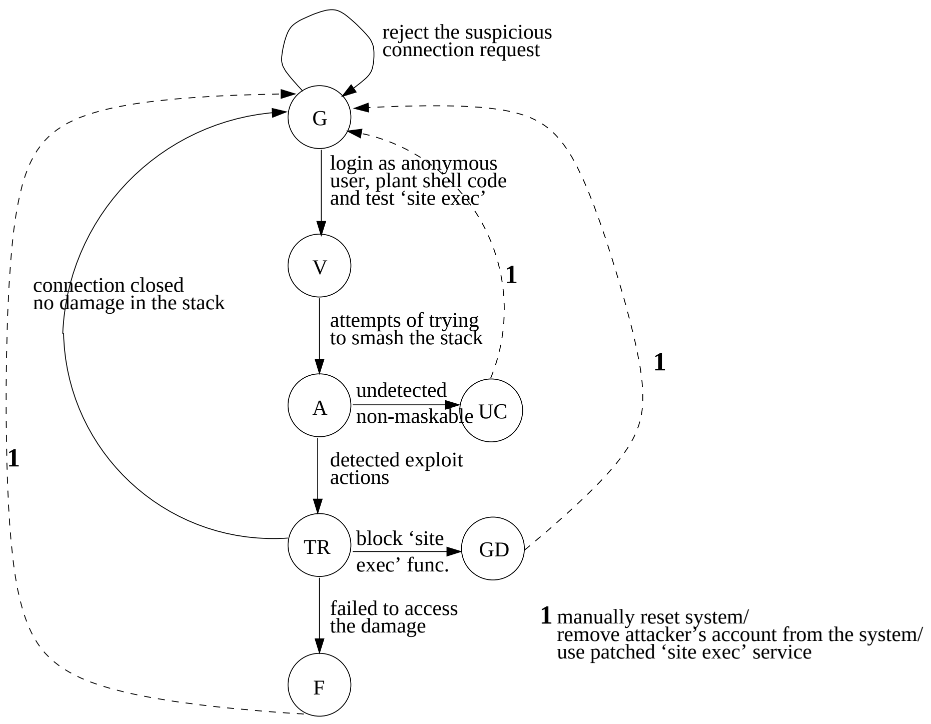
2 http://target:9090/servlet/com.sun.server.http.pagecompile.jsp92.JspServlet/board.html

3 restoration/reconfiguration/evolution (no execution privelge in bboard directory)

Figure 4. State transition diagram for BBoard vulnerability

of some function calls correctly. In particular, the program implementing the `'site exec'` functionality passes the input argument directly into the stack without proper checking and uses it as the character-formatting argument of a procedure call. A malicious user can exploit this defect by providing a deliberately crafted character-formatting argument, which is longer than its presumed size. When this long argument is passed into the stack, it can overwrite the existing data in the stack and by changing the return address in the stack, the user can get the control of the server. By analyzing available code exploiting the `wu-ftpd` problem, we can outline the steps that an attacker need to perform and the vulnerable ftp daemon's responses as follows:

1. An attacker tries to log in to a vulnerable ftp server as an anonymous user.
2. When the vulnerable ftp server requests the password, the attacker enters a password attached with the malicious shell code.
3. The vulnerable ftp daemon accepts the connection and the attacker becomes a legal anonymous user.
4. To confirm that the ftp daemon has the expected `'site exec'` problem, the attacker makes a test by executing command `'site exec'` with character-format argument.
5. The vulnerable ftp server accepts the command and generates an acceptance response to the attacker. An acceptance response for the above requests indicates that the `'site exec'` problem exists.
6. The attacker begins exploiting actions against the vulnerable ftp daemon by entering `'site exec'` command with carefully crafted argument that is long enough to smash the stack and overwrite the return address of the procedure call.
7. If the location of the return address in the stack has not been found out, the attacker will try again with



reject the suspicious connection request
G
login as anonymous user, plant shell code and test 'site exec'
V
connection closed no damage in the stack
attempts of trying to smash the stack
1
1
1
A
undetected
non-maskable
UC
detected exploit actions
TR
block 'site exec' func.
GD
failed to access the damage
F
1 manually reset system/ remove attacker's account from the system/ use patched 'site exec' service

Figure 5. State transition diagram for `wu-ftpd` vulnerability

adjusted argument for `'site exec'` command and send the command with the argument to the server.

8. After several attempts at step (7), the return address has been overwritten and pointed to the malicious shell code that reside in the system.
9. The system runs the shell code and the attacker gets the unauthorized privilege.

The `wu-ftpd` vulnerability can cause direct and indirect impacts. The immediate impact for the system is that any local/remote user can get **unauthorized privilege**. The potential impact is that once intruders gain unauthorized privilege, they can perform further malicious actions such as installing password sniffer, changing syslog configuration files and installing **DDoS** tools. Those actions may threaten the service's availability, confidentiality, authenticity and integrity. Based on the previous discussion of the state transition model and the analysis of the impact of the `wu-ftpd` vulnerability, we can map the state transition diagram as Figure 5 and describe the system with intrusion tolerance capability as follows:

- Before any user exploits the system, the system is in a good state G and functioned as normal. If the system decides to close a connection to an attacker after it detected the attacker's pre-attack actions (as described in step 1,2,3), it can still stay in G state.
- If the pre-attack actions have not been detected by the system, the system will be in a V state because the vulnerability is exposed to the attacker.
- The system is in the A state if the attacker is exploiting the vulnerability.
- The system goes from A state to the UC state if the exploiting activities (as described in steps 6,7) have not been detected and the malicious shell code runs successfully.
- The system is in the TR state if the exploiting activities (as described in steps 6,7) are detected.
- The system enters GD state if it blocks the `'site exec'` function.
- The system enters F state if the attack is detected but the damage caused by the exploiting activities is unknown.
- The system is returned to the good state G from states UC, F or GD by manually resetting the system, removing the attacker's account or by using a patched `'site exec'` service.

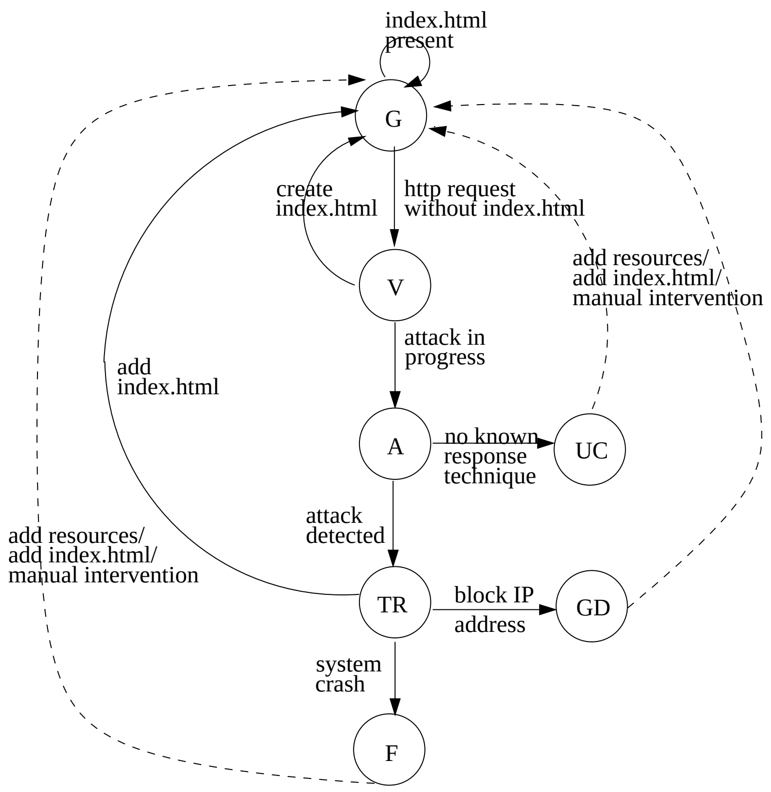

index.html present
G
create index.html
http request without index.html
V
add resources/ add index.html/ manual intervention
attack in progress
add index.html
A
no known response technique
UC
attack detected
add resources/ add index.html/ manual intervention
TR
block IP address
GD
system crash
F

Figure 6. State diagram mapping of `DoS`

3.5. Denial of Service (DoS) vulnerability (Bugtraq ID 1941)

Small HTTP Server is a full service web server which is less than 30 KB and requires minimal system resources. Recently denial of service (DoS) attacks have been identified in these servers. Unlike the conventional type of DoS attacks, here the attack consumes all the system resources. This is an instance of **DoS from external entities**.

When making an http request without a filename specified, the server will attempt to locate `index.html` in that particular directory. If `index.html` does not exist, the server will utilize a large amount of system memory. If numerous http requests, again structured without a filename, are sent to the web server, an attacker could cause it to consume all system memory. A restart of the application is required to gain normal functionality.

The state transitions are captured in our model, as indicated in the figure 6.

- If `index.html` is not present, without any preparation or previous knowledge, the attacker can start the attack by issuing a http request of the form: `http://target/DirectoryWithoutIndex/`. So, the system moves from the good state, to the vulnerable state V.
- With requests getting accumulated, the server moves to the active attack state, A. In this state, it can still respond to the legitimate requests, but with a degraded quality, taking more time for each request.
- When the system is not equipped with detection techniques, it goes to the undetected compromised state, UC, wherein the only course of action is to have manual reboot.
- On the other hand, when the system is equipped with detection techniques, it goes to the triage state, TR, where it tries to recover from the attack.
- If possible, the system then moves to the graceful degradation state, GD, where it can provide essential services only. This can be done by in the following manner. Upon detecting numerous requests without `index.html` from the same network or IP address, requests from the appropriate addresses could be blocked. However, this might prevent genuine requests from getting serviced.
- If recovery is not possible, the system goes to the fail state, F, with an alarm and is destabilized.
- The system is brought back to the good state, G from

states UC, GD or F either by upgrading the system with more resources or by creating an `index.html` file in the directories. This would ensure that this attack does not happen again, and the system does not move from the good state to the vulnerable state.

4. Conclusion

To summarize, we have presented a state transition model to describe the dynamic behavior of intrusion tolerant systems. This model provides a framework from which we can define the vulnerability and threat set to be addressed by the SITAR architecture. We also showed how this model helps us to describe both known security exploits and unknown attacks by focusing on attack impact rather than specific attack procedures. By going through the exercise of mapping from known vulnerability to this transition model, we identified a reasonably complete fault space that should be considered in a general intrusion tolerant system. Our future work includes further investigation on adaptation, reconfiguration and graceful degradation techniques to either ensure a minimum service level or recover the full system functionality. We will also conduct analytical and quantitative assessment on operational security, create a prototype system, and evaluate the prototype through experimental measurements.

5. Acknowledgments

The entire SITAR project team, in addition to the authors, Frank Jou and Dan Stevenson from MCNC, and Hu (Thomas) Pan from Duke University, have contributed to the development of the state machine model. Furthermore, Frank, Dan and Hu have also provided very useful input during the preparation of this paper that helped to improve the presentation significantly.

References

[1] Auscert advisory aa-2000.02. `http://ftp.auscert.org/pub/auscert/advisory/AA-2000.02`, 2000.

[2] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, Carnegie Mellon Software Engineering Institute, 2000.

[3] E. G. Amoroso. *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response*. Intrusion.Net Books, 1999.

[4] T. Aslam, I. Krsul, and E. H. Spafford. Use of taxonomy of security faults. Technical Report TR-96-051, Department of Computer Science, Purdue University, 1996.

[5] R. G. Bace. *Intrusion Detection*. Technology Series. Macmillan Technical Publishing, 2000.

[6] C. C. Center. Cert advisory ca-2000-13: Two input validation problems in `ftpd`. `http://www.cert.org/advisories/CA-2000.html`, 2000.

[7] W. Du and A.P.Mathur. Vulnerability testing of software system using fault injection. Technical Report Coast TR-98-02, Department of Computer Science, Purdue University, 1998.

[8] I. Krsul, E. H. Spafford, and M. V. Tripunitara. Computer vulnerability analysis. Technical Report Coast TR 98-07, Department of Computer Science, Purdue University, 1998.

[9] MCNC and Duke University. Sitar : A scalable intrusion-tolerant architecture for distributed services. Technical report, Research Proposal to DARPA BAA-00-15, 2000.

[10] S. Northcutt and J. Novak. *Network Intrusion Detection: An Analysts' Handbook*. New Riders, September 2000.

[11] P.A.Lee and T.Anderson. *Fault Tolerance: Principles and Practice*. Springer Verlag, 1990.

[12] R.J.Ellison, D.A.Fisher, R.C.Linger, H.F.Lipson, T.A.Longstaff, and N.R.Mead. Survivability: Protecting your critical systems. *IEEE Internet Computing*, 3(6):55–63, 1999.

[13] W.A.Arbaugh, W.L.Fithen, and J.McHugh. Windows of vulnerability: A case study analysis. *IEEE Computer*, 33(12):52–59, December 2000.