Dr. Dobbs Jolt Award Finalist 2014

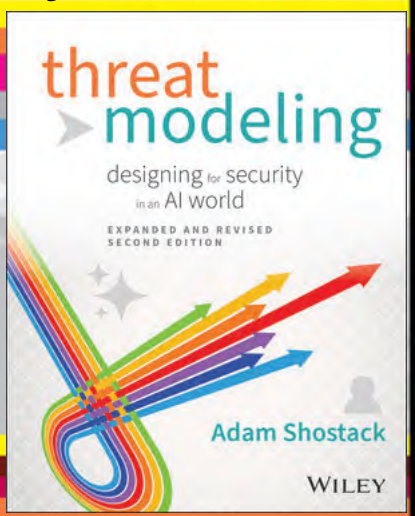
Adam Shostack
Microsoft's Threat Modeling Expert

Appendix B "Threat Trees" from

threat modeling

designing for security

New Edition Coming January 2027



WILEY

Threat Modeling: Designing for Security

Published by
John Wiley & Sons, Inc.
10475 Crosspoint
BoulevardIndianapolis, IN 46256
www.wiley.com

Published by John Wiley & Sons, Inc., Indianapolis, Indiana
Published simultaneously in Canada

ISBN: 978-1-118-80999-0
ISBN: 978-1-118-82269-2 (ebk)
ISBN: 978-1-118-81005-7 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2013954095

APPENDIX

B

Threat Trees

These threat trees are worked-through analyses, intended to act as both models and resources. Each tree is presented twice, first as a graphical tree and then as a textual one. The versions contain the same data, but different people will find one or the other more usable. The labels in the trees are, by necessity, shorthand for a longer attack description. The labels are intended to be evocative for those experienced with these trees. Toward this goal, some nodes have a label and a quoted tag, such as "phishing." Not all nodes are easily tagged with a word or an acronym. The trees in this appendix are OR trees, where success in any node leads to success in the goal node. The rare exceptions are noted in the text and diagrams.

This appendix has three sections: The main body is a set of 15 STRIDE threat trees. That is followed by three trees for running code on a server, a client, or a mobile device, as those are common attacker targets. The last tree is "exploiting a social program," illustrating how systems such as e-mail and instant messenger programs can be exploited. The appendix ends with a section on tricky filenames, and their use in certain classes of attacks which trick people.

STRIDE Threat Trees

These trees are organized according to STRIDE-per-element. Each has as its root node the realization of a threat action. These STRIDE trees are built on the ones presented in *The Security Development Lifecycle* (2006). The trees are focused on first-order threats. Once you have elevated privileges to root, you can do an awful lot of tampering with files on that system (or other mischief), but such actions are not shown in the trees, as including them leads to a maze of twisty little trees, all alike.

Each tree in this section is followed by a table or tables that explains the node and discusses mitigation approaches, both for those developing a system and those deploying it ("operations") who need stronger security. The term "few" in the mitigations column should be understood as meaning there is no obvious or simple approach. Each of the ways you might address a threat has trade-offs. In the interests of space and focused threat modeling, those trade-offs are not discussed in this appendix. In a very real sense, when you start considering those trade-offs, threat modeling has done its job. It has helped you find the threat. What you do with it, how you triage and address the bugs from those threats, is a matter of good engineering.

The trees presented in this section are shown in Table B-0. In this appendix, the labels differs from normal Wiley style, in ways that are intended to be easy (or easier) to use given the specific information in this appendix.

- Tables are numbered to align with the figures to make it easier for you to go back and forth between them. Thus, figure B-1 is referenced by Tables B-1a, B-1b, B-1c, B-1d and B-1e, while figure B-2 is referenced by a single table B-2.
- Many tables are broken into smaller logical units. The breakdown is driven by a desire to have tables of reasonable length.
- Where there are multiple tables per tree, they are referred to as subtrees. Each subtree is labeled with a combination of category ("spoofing") and subnode ("by obtaining credentials").
- Numbering of tables starts at zero, because you're a programmer and prefer it that way. (Or perhaps because we wanted to start figures at B-1, making this table hard to number.)

Table B-0 does not have a figure associated with it.

Table B-0: STRIDE-per-Element

		DFD APPLICABILITY			
THREAT TYPE	MITIGATION	EXTERNAL ENTITY	PROCESS	DATA FLOW	DATA STORE
S – Spoofing	Authentication	B-1*	B-2	B-3*	
T – Tampering	Integrity		B-4	B-5	B-6
R – Repudiation	Non-repudiation	B-7	B-7		B-8*
I – Information Disclosure	Confidentiality		B-9	B-10	B-11
D – Denial of Service	Availability		B-12	B-13	B-14
E – Elevation of Privilege	Authorization		B-15		

*(B-1) Spoofing an external entity is shown as spoofing a client in the following tables.

*(B-3) Spoofing of a data flow is at odds with how STRIDE-per-element has been taught.

*(B-8) Repudiation threats matter when the data stored is logs.

NOTE **The spoofing of a data flow is at odds with how STRIDE-per-element has generally been taught or presented, but it is in alignment with how some people naturally think of spoofing. It's less-tested content; and as you gain experience, you're likely to find that the threats it produces overlap heavily with spoofing of a client or tampering with a data flow.**

Each tree ends with "other," a node without further discussion in the explanatory text because given its unanticipated nature, what to add is unclear. Each tree is technically complete, as "other" includes everything. Less glibly, each tree attempts to focus on the more likely threats. Therefore, for example, although backup tape data stores are subject to both tampering and information disclosure, the disclosure threats are far easier to realize, so the backup threats are not mentioned in the tree for tampering with a data store. Generally, only security experts, those aspiring toward such expertise, or those with very high value targets should spend a lot of time looking for those "others." It is tempting, for example, when writing about tampering with a process, to declare that

those threats "matter less." However, which threats matter is (ahem) a matter of requirements.

As a community, we know relatively little about what threats manifest in the real world (Shostack, 2009). If we knew more about which threats are likely, we could create trees optimized by likelihood or optimized for completeness, although completeness will always need some balance with pedantry and usability. Such balance might be provided by tree construction or how the tree is presented. For example, a software system that contains the trees could present various subsets.

Spoofing an External Entity (Client/ Person/Account)

Figure B-1 shows an attack tree for spoofing by/of an external entity. Spoofing threats are generally covered in Chapter 3, "STRIDE," and Chapter 8, "Defensive Tactics and Technologies," as well as Chapter 14, "Accounts and Identity."

- Goal: Spoof client
- Obtain existing credentials
  - Transit
  - Federation issues
  - Change management
  - Storage
    - At server
    - At client
    - At KDC
    - At 3rd party
- Backup authentication
  - Knowledge-based authentication (KBA)
  - Information disclosure (e-mail)
  - Chained authentication
- Authentication UI
  - Local login Trojan ("CAD")
  - Privileged access Trojan (./sudo)
  - Remote spoof ("phishing")
- Insufficient authentication
  - Null credentials

- Guest/anonymous credentials
- Predictable credentials
- Factory default credentials
- Downgrade authentication
- No authentication
- Other authentication attack

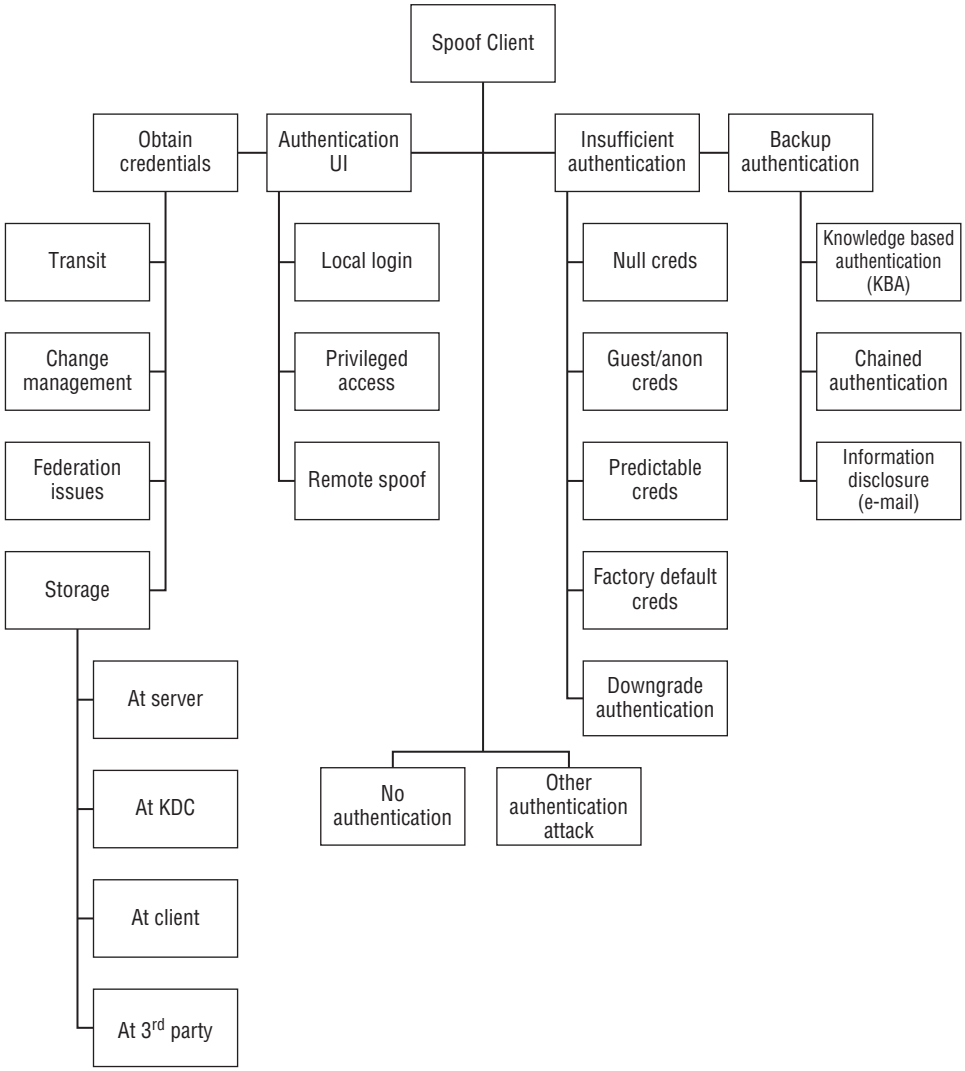


Figure B-1: Spoofing an external entity (client)

Also consider whether tampering threats against the authorization process should be in scope for your system.

Table B-1a: Spoofing by Obtaining Existing Credentials Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Transit	If the channel over which authenticators are sent is not encrypted and authenticated, an attacker may be able to copy a static authenticator or tamper with the connection, piggybacking on the real authentication	Use standard authentication protocols, rather than develop your own. Confirm that strong encryption and authentication options are used.	Tools such as SSL, IPsec, and SSH tunneling can be added to protect a weak transit.
Federation issues	If authentication is federated, failures at the trusted party can be impactful.	Change the requirements.	Logging, to help you know what's happening Tunneling, to improve network defense (but a federation system with weak cryptography will have other issues)
Change management	If your authentication change management is weaker than the main authentication, then it can be used as a channel of attack. This relates closely to backup authentication issues.	Ensure that change management is strongly authenticated.	Logging and auditing

Table B-1b: Spoofing by Obtaining Existing Credentials: Attacking Storage Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
At server	The server stores the authenticator in a way that's vulnerable to information disclosure or tampering. Information disclosure is worse with static approaches such as passwords than with asymmetric authentication.	Ensure that the authenticators are well locked down. Seek asymmetric approaches. If using passwords, see Chapter 14 for storage advice.	Consider adjusting permissions on various files (and the impact of doing so).
At client	The client stores authenticators in a way that an attacker can steal.	The balance between usability and credential prompting can be hard. Ensure that the credentials are not readable by other accounts on the machine. If you're designing a high-security system, consider hardware techniques to augment security (TPM, smartcards, hardware security modules).	Treat authentication as a probabilistic decision, and use factors such as IP or machine fingerprints to augment authentication decisions.
At KDC (key distribution center)	If a KDC is part of a symmetric-key authentication scheme, it may need to store plaintext authenticators.	To minimize the attack surface, design the KDC to perform as few functions as possible. Write your security operations manual early.	Protect the heck out of the machine with firewalls, IDS, and other techniques as appropriate.
At third party	People reuse passwords, and other parties may leak passwords your customers use.	Avoid static passwords altogether. If your usernames are not e-mail addresses, exploiting leaks is much harder. For example, if the Acme company leaks that "foobar1" authenticates Alice@example.com, attackers can simply try Alice@example.com as a username; but if the username which leaks is Alice, the odds are that another person has that username at your site.	Code to detect brute-force attempts, or an upswing in successful logins from a new location or IP.

Table B-1c: Spoofing Backup Authentication Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Knowledge-based authentication (KBA)	KBA schemes have many problems, including memorability and secrecy. See Chapter 14."	Use social authentication or other schemes rather than KBA.	Use a password management tool, and treat each KBA field as a password.
Information disclosure	Attacker can read backup authentication messages containing secrets that can be used to authenticate. This most frequently appears with e-mail, but not always.	Use social authentication. If you can't, don't store passwords in a form you can send, but rather create a random password and require the person change it, or send a one-use URL.	If the product you're using has this problem, it's probably non-trivial to graft something else onto it.
Chained authentication	Attackers who take over e-mail can abuse e-mail information disclosure as well as disclosures after taking over accounts. See "How Apple and Amazon Security Flaws Led to My Epic Hacking,"(Honan, 2012) for more information.	Use social authentication, and consider treating authentication in a probabilistic fashion. If you can't, avoid relying on authenticators that other organizations can disclose. (No claim that's easy.)*	These issues are hard to fix in development, and even harder in operations.

*It is tempting to suggest that you should not display data that other sites might use for authentication. That advice is a "tar pit," leading to more questions than answers. For example, some organizations use the last four digits of a SSN for authentication. If you don't display those last four digits, should you display or reveal other digits to someone who can authenticate as your customer? If you do, you may help an attacker construct the full number. It is also tempting to say don't worry about organizations that have made poor decisions, but backup authentication is a real challenge, and characterizing those decisions as poor is easy in a vacuum.

Table B-1d: Spoof Authentication UI Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Local login Trojan ("CAD")	An attacker presents a login user interface to someone	Use of secure attention sequences to reach the OS	Always press Ctrl+Alt+Delete (CAD) to help keep your login information secure.
Privileged access Trojan (./sudo)	An attacker alters the `PATH` variable so their code will load first.	Hard to defend against if the environment allows for customization	Don't use someone else's terminal session for privileged work.
Remote spoofing ("phishing")	A website* presents itself as a different site, including a login page.	Reputation services such as IE SmartScreen or Google's Safe Browsing	Reputation services that search for abuses of your trademarks or brand

* Currently, phishing is typically performed by websites, but other forms of remote spoofing are feasible. For example, an app store might contain an app falsely claiming to be from (spoofing) Acme Bank.

Table B-1e: Spoofing Where There's Insufficient Authentication Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Null credentials	The system allows access with no credentials. This may be OK, for example, on a website.	Add authentication.	Disable that account.
Guest/anonymous credentials	The system has a guest login. Again, that may be OK.	Remove the guest account.	Disable that account.

Continues

Table B-1e (*continued*)

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Predictable credentials	The system uses predictable usernames or a poor random generator for passwords.	If assigning passwords, use strong randomness. If usernames are unlikely to be easily discovered, arbitrary usernames may help.	Reset passwords, assign new usernames.
Factory default credentials	The system ships with the same credentials, or credentials that can be guessed, like those based on Ethernet ID.	Force a password change on first login, or print unique; passwords on each device/ documentation.	Change the password.
Downgrade authentication	An attacker can choose which of several authentication schemes to use.	Remove older/weaker authentication schemes, turn them off by default, or track which each client has used.	Turn off weaker schemes.

There is no table for either no authentication (which is hopefully obvious; change that if the requirements warrant such a change) or other authentication attack.

Spoofing a Process

Figure B-2 shows an attack tree for spoofing a process. Spoofing threats are generally covered in Chapter 3, and Chapter 8. If an attacker has to be root, modify the kernel, or similarly, use high privilege levels to engage in spoofing a process on the local machine; then, generally, such attacks are hard or impossible to mitigate, and as such, requirements to address them are probably bad requirements.

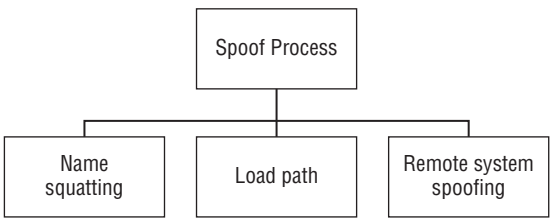


Figure B-2: Spoofing a process

Goal: Spoof process

- Name squatting
- Load path
- Remote system spoofing

Table B-2: Spoofing a Process

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Name squatting	The attacker connects to a communication endpoint, such as a file, named pipe, socket, or RPC registry and pretends to be another entity.	Use the OS for naming, permissions of synchronization points.	Possibly permissions
Load path	The attacker deposits a library, extension, or other element in a way that the process will trust at next load.	Name the files you want with full paths (thus, refer to %windir%\winsock.dll, not winsock.dll or ~/.login, not .login).	Check permissions on files.
Remote system spoofing	Confuse a process about what a remote process is by tampering with (or "spoofing") the data flows.	See tampering, spoofing data flow trees (B5 and B3).	See tampering, spoofing data flow trees.

Spoofing of a Data Flow

Figure B-3 shows an attack tree for spoofing a process. Spoofing threats are generally covered in Chapter 3, and Chapter 8; and many of the mitigations are cryptographic, as covered in Chapter 16, "Threats to Cryptosystems." Spoofing a data flow is an amalgamation of tampering with a data flow and spoofing a client, which may be a more natural way for some threat modelers to consider attacks against network data flows. It is not traditionally considered in STRIDE-per-element, but it is included here as an experimental approach.

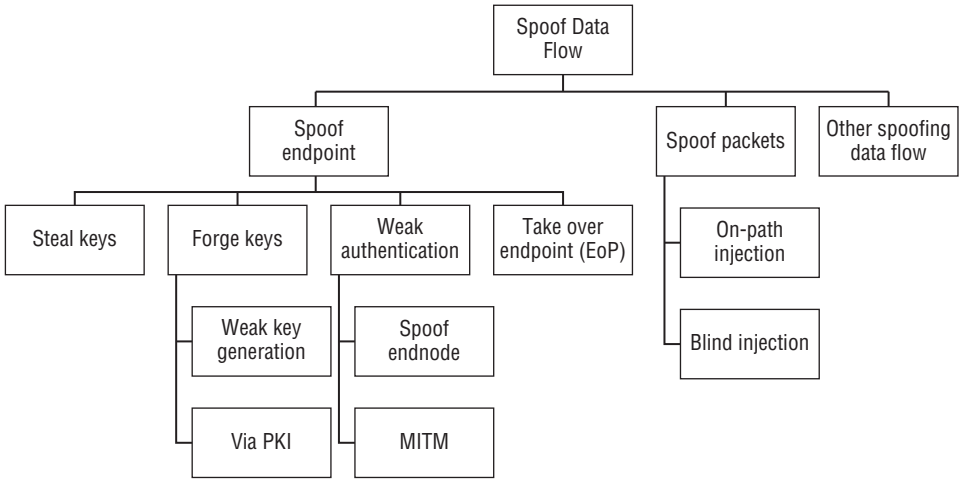


Figure B-3: Spoofing of a data flow

Goal: Spoof a data flow

- Spoof endpoint
  - Steal keys
  - Forge keys
    - Weak key generation
    - Via PKI
  - Weak authentication
    - Spoof endnode
    - MITM
  - Take over endpoint
- Spoof packets
  - Blind injection
  - On-path injection
- Other spoofing data flow

Table B-3a: Steal/forge Keys Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Steal keys	Information disclosure where the data is crypto keys. Generally, see the information disclosure trees (B11, and also B9 and B10).	OS tools for secure key storage	Hardware security modules

Table B-3b: Forge Keys Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Weak key generation	If a system is generating keys in a weak way, an attacker can forge them.*	Avoid generating keys at startup, especially for data center machines or those without hardware RNGs.	Regenerate keys
Via PKI	There are many ways to attack PKI, including misrepresentation, legal demands, commercial deals, or simply breaking in.†	Key persistence, perspectives/Convergence-like systems	Few

* Some examples of weak key generation include (Debian, 2008) and (Heninger, 2012).

† Examples of misrepresentation include Verisign issuing certificates for Microsoft (Fontana, 2001). A legal demand might be like the one issued to Lavabit (Masnick, 2013). Commercial deals include ones like Verisign's operation of a "lawful intercept" service (Verisign, 2007).

Table B-3c: Weak Authentication Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Spoof endnode	Any non-cryptographically secured method of referring to a host, including MAC address, IP, DNS name, etc.	Use cryptography properly in the authentication scheme.	Tunneling
MITM	Man-in-the-middle attacks	Strong authentication with proper crypto	Tunneling

Table B-3d: Spoof Packets Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
On-path injection	A network attacker can see the normal packet flow and insert their own packets.	Cryptographic channel authentication and/or message authentication	Tunneling
Blind injection	An attacker cannot see packets and learn things such as sequence numbers, or see responses, but injects packets anyway.	Cryptographic channel authentication and/or message authentication	Tunneling

The "take over endpoint" node should be self-explanatory.

Tampering with a Process

Figure B-4 shows an attack tree for tampering with a process. Tampering threats are generally covered in Chapter 3, and Chapter 8, and are touched on in Chapter 16.

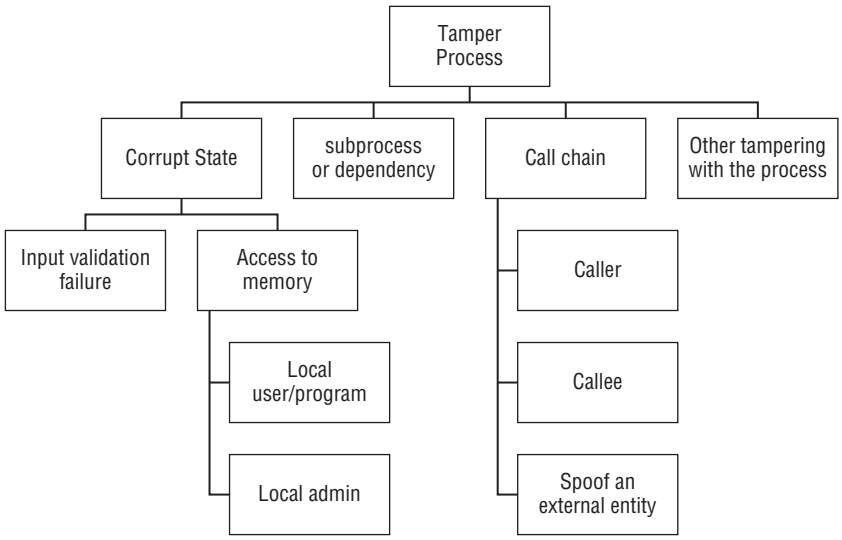


Figure B-4: Tampering with a process

Goal: Tamper with a process

- Corrupt state
  - Input validation failure
  - Access to memory
    - Local user/program
    - Local admin
- Call chain
  - Caller
  - Callee
  - Spoof an external entity
- Subprocess or dependency
- Other tampering with the process

Also consider whether these threats apply to subprocesses, or what happens if callers or callees are spoofed.

Table B-4a: Corrupt State Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Input validation failure	If inputs are not appropriately validated, memory corruption can result in EoP or DoS.	Carefully validate all input for the appropriate purpose.	None for the threat, sandboxing can contain impact
Access to memory (local user/program)	A user with authorized write access to memory can tamper with the process. It's important to realize that on many operating systems, this is the case for all programs the user runs. (Can you freely attach a debugger? If so, a program you're running can do the same.)	Create new account; Review shared memory permissions.	few
Access to memory (local admin)	A local admin with a debugger can be a threat (e.g., you're trying to use DRM).	Memory protection and anti-debugging schemes, generally used by DRM and other malware	separate machines

Table B-4b: Call Chain Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Caller	Untrusted code may call your code, passing it malicious parameters.	Input validation	Permissions can be used to ensure that lower-trust code can't execute untrustworthy applications.
Callee	Your callees can tamper with memory (e.g., via extension points).	Design more constrained APIs.	Ensure that only trusted/trustworthy callees are called, or trustworthy plugins are installed.
Subprocess or dependency	This is the same as the previous two, expressed differently in the hopes of being evocative.		

Tampering with a Data Flow

Figure B-5 shows an attack tree for tampering with a data flow. Tampering threats are generally covered in Chapter 3, and Chapter 8, cryptography will often play a part in addressing them, and is covered in Chapter 16.

Generally, if you don't prevent spoofing of a data flow, you have tampering and information disclosure problems.

Data flow threats can apply to channels or messages, or both. The difference is easiest to see with examples: E-mail messages travel over an SMTP channel, whereas HTML (and other format) messages travel over HTTP. The question of which you need (either, neither, or both) is a requirements question.

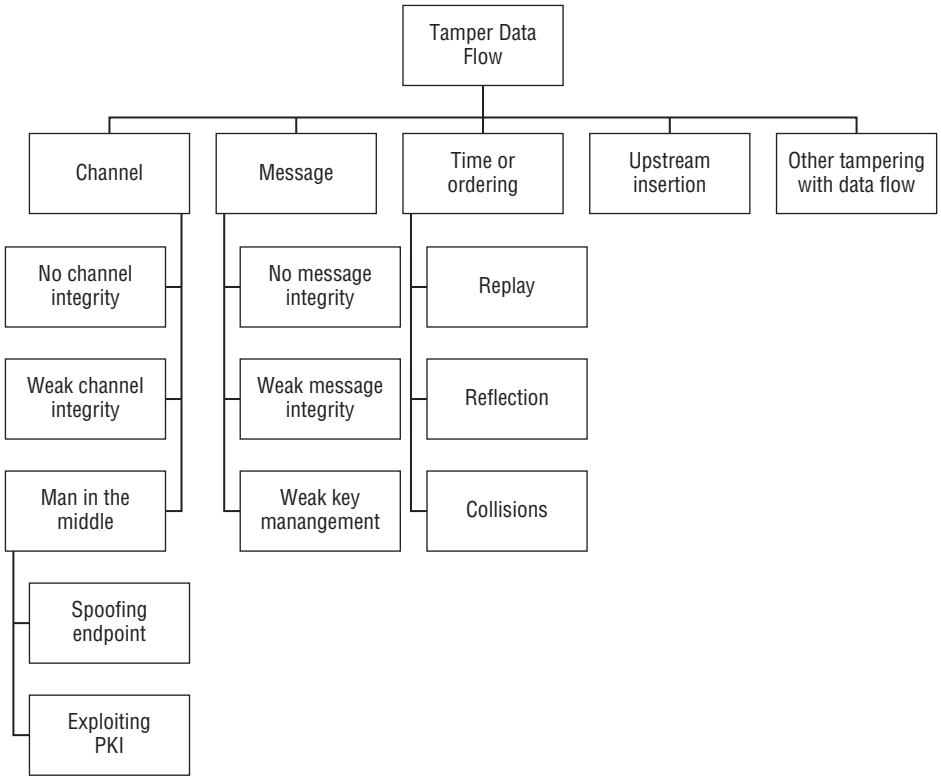


Figure B-5: Tampering with a data flow

Goal: Tamper with a data flow

- Message
  - No message integrity
  - Weak message integrity

- Weak key management
- Channel
  - No channel integrity
  - Weak channel integrity
  - Weak key management
  - Man in the middle
    - Spoof endnode (or endpoint)
    - Exploiting PKI
- Time or ordering
  - Replay
  - Reflection
  - Collisions
- Upstream insertion
- Other tampering with data flow

Table B-5a: Tampering with a Message Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
No message integrity	Nothing protects message integrity from tampering.	Add integrity controls.	Very dependent on the data flow. Tunneling may help address parts of the threat, but really provides channel integrity.
Weak message integrity	Weak algorithm, such as MD5	Use a better algorithm.	As above
Weak key management	Weak key management can lead to problems for the messages.	Use better key management.	As above

Table B-5b: Tampering with Channel Integrity Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
No channel integrity	Nothing protects channel integrity.	Add integrity controls.	Tunneling over an integrity-protected transport can help.
Weak channel integrity	Weak algorithm, such as MD5	Use a better algorithm.	As above

Continues

Table B-5b (*continued*)

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Weak key management	Key management can be bad for either channels or messages or both.	Use better key management.	As above
Man in the middle	Man in the middle attacks that allow data tampering	Strong authentication	Tunneling
Spoofing endpoint	See tree B3.		
Exploit PKI	Attack the PKI system	Key pinning	Convergence, Persistence

Table B-5c: Tampering with Time or Ordering Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Replay	Attacker resends messages that the system really sent .	Message identifiers, and tracking what's been seen	Tunneling may help.
Reflection	Attacker takes a message and sends it back to the sender.	Careful protocol design	Tunneling may help.
Collisions	Attacker sends (possibly invalid) messages with sequence numbers, causing real messages to be ignored.	Manage identifiers after validation.	Few

Table B-5d: Tampering via Upstream Insertion Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Upstream insertion	Rather than tamper with the data flow directly, convince an endnode to insert the data you want.	Input validation on places where messages could be inserted; output validation on places where you send them.	Possibly firewalling

Tampering with a Data Store

Figure B-6 shows an attack tree for tampering with a data store. Tampering threats are generally covered in Chapter 3, and Chapter 8.

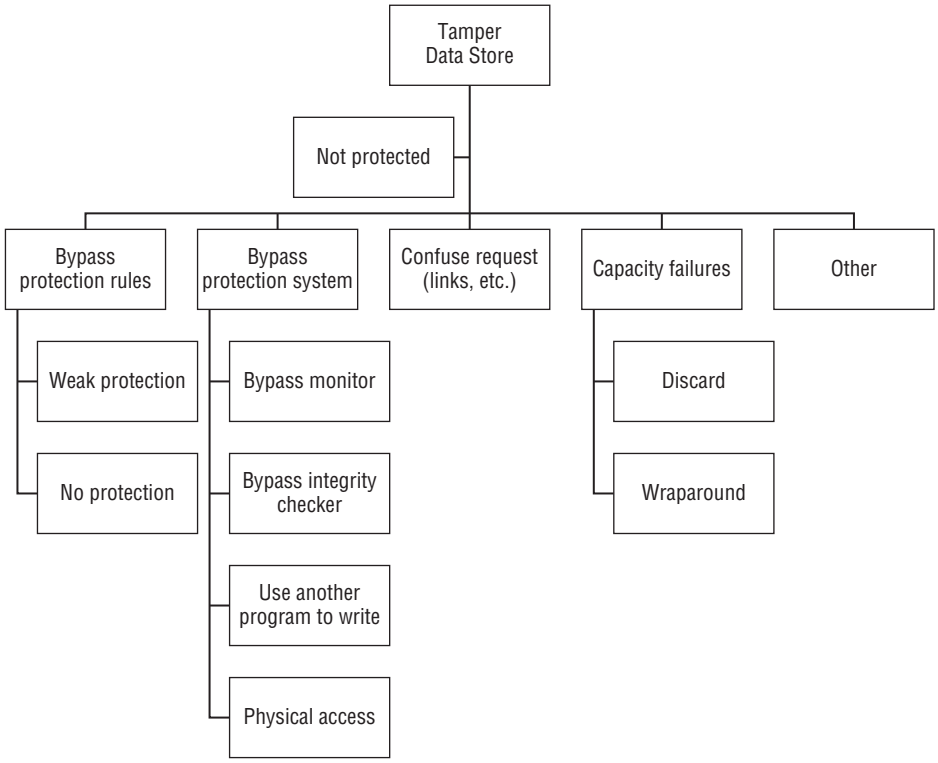


Figure B-6: Tampering with a data store

Goal: Tamper with a data store

- Not protected
- Confuse request
- Bypass protection rules
  - Weak rules
  - No protection
- Bypass protection system
  - Bypass monitor
  - Bypass integrity checker
  - Use another program to write
  - Physical access
- Capacity failures
  - Discard
  - Wraparound
- Other

You might consider tampering with the monitor or elevation of privilege against it; however, often at that point the attacker is admin, and all bets are . . . more complex. Capacity failures are an interesting balance of tampering and denial of service, but they can certainly result in tampering effects that are valuable to an attacker.

Table B-6a: Tamper with a Data Store Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Not protected	There's no protection for data; for example, it's on a filesystem with no permissions, or a globally writable wiki.	Add protections as appropriate.	Physical access control
Confuse request	Can a data element have multiple names, for example through links or inclusion?	Check permissions on the resolved object (after name canonicalization).	Probably none with reasonable effort

Table B-6b: Tamper by Bypassing Protection Rules Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Weak protection	The rules (ACLs, permissions, policies) allow people with questionable justification to alter the data.	Ensure your code creates data with appropriate permissions.	Change the permissions.
No protection	The rules allow anyone to write to the data store.	As above	As above

Table B-6c: Tamper by Bypassing a Protection System Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Bypass monitor	Exploit the lack of a "reference monitor" through which all access requests pass, or take advantage of bugs.	Good design, extensive testing	Use a better system.
Bypass integrity checker	Attack either the integrity checker code or its database (often out of scope).	None	Read-only database, boot from separate OS
Use another program to write	If you can't put data into a store, can another program do so on your behalf? Can you put it somewhere else?	Ensure you understand the data you're writing on behalf of other processes.	Remove/block the proxy program.
Physical access	Reboot the system into another OS.	Encrypted filesystem or integrity checks with a crypto key stored elsewhere	Encrypted filesystem, physical protection

Table B-6d: Tampering via Capacity Failures Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Discard	Refers to new data not being recorded	Shutdown or switch to wraparound.	More storage
Wraparound	Refers to the oldest data being deleted to make room	Shutdown or switch to discard.	More storage

The developer mitigations in Table B-6d are slightly less tongue-in-cheek than it may appear. After all, when you are out of storage, you're out of storage; and at that point you must choose either to allow the storage issue to stop the system or to make room in some fashion. Not shown in the table are compression and moving data to another store somewhere, both of which can be legitimate approaches.

Repudiation against a Process (or by an External Entity)

Figure B-7 shows an attack tree for repudiation against a process, or by an external entity. Repudiation threats are generally covered in Chapter 3, and Chapter 8, as well as in Chapter 14.

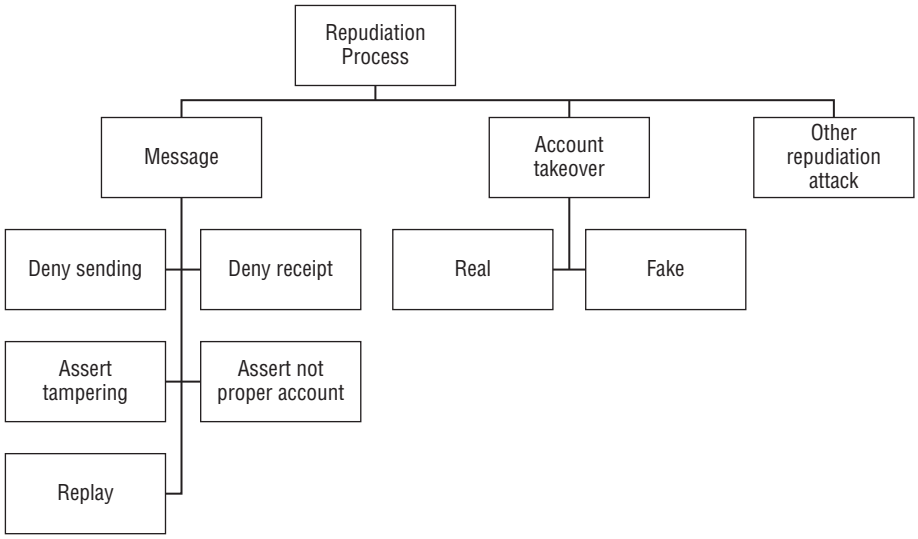


Figure B-7: Repudiation against a process

Goal: Repudiation

- Account takeover
  - Real
  - Fake
- Message
  - Deny sending
  - Deny receipt
  - Assert tampering
  - Assert not proper account

- Replay
- Other

Table B-7a: Repudiate by Account Takeover Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Real	The account was actually compromised.	Stronger authentication	Additional authentication tools
Fake	Someone asserts that the account was taken over.	Strong logging	Strong logging, penalties*

* If you impose penalties, you will inevitably impose them on innocent customers. Be judicious.

Table B-7b: Message Repudiation Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Deny sending	Claim that a message wasn't sent.	Digital signatures	Logging
Deny receipt	Claim that a message wasn't received.	Web bugs, logs	Logging, possibly firewalls to block web bugs
Assert tampering	Claim that a message has been altered.	Digital signatures	Log message hashes in a reliable way.
Assert not proper account	E-mail from barack.obama37@example.com is probably not from the president of the United States.	Meaningful IDs, nicknames (See Chapter 15, "Human Factors and Usability.")	Process?
Replay	If your messages are of the form "sell 1,000 shares now," then an attacker might be able to claim your sale of an extra 1,000 shares was in error.	Design protocols that are more precise.	Logging

Repudiation, Data Store

Figure B-8 shows an attack tree for repudiation in which logs are involved. Repudiation threats are generally covered in Chapter 3, and Chapter 8.

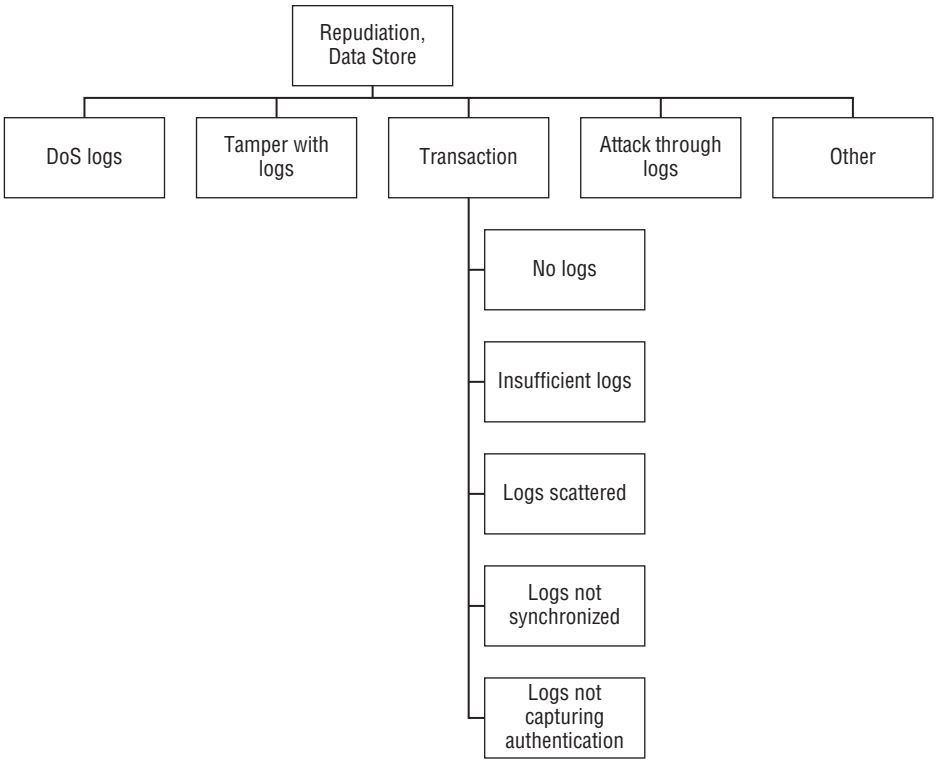


Figure B-8: Repudiation, Data Store

Goal: Repudiation (data store focus)

- Transaction
  - No logs
  - Insufficient logs
  - Logs scattered
  - Logs not synchronized
  - Logs not capturing authentication

- Tamper with logs
- DoS logs
- Attack through logs
- Other

Table B-8a: Transaction Repudiation Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
No logs	There are no logs.	Log	Log (It's better than bad, it's good!)
Insufficient logs	The logs don't tell you what you need.	Scenario analysis	Perhaps a logging proxy
Logs scattered	Your logs are all over the place, responding to a repudiation claim is too expensive.	Scenario analysis	Log consolidation
Logs not synchronized	Your systems have different times, meaning correlating your logs is hard.	Few	Set all systems to work in UTC and use a local time server.
Logs not capturing authentication	Your logs don't capture the inputs to authentication decisions, such as Geo-IP or fingerprinting.*	Log more	Few

* Also, manage privacy issues here, and information disclosure risks with logs exposing authentication information.

Table B-8b: Redupidation Attacks through Logs Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Attack through logs	Logs are often seen as "trusted" but usually contain information of varying trustworthiness.	Be careful about what assumptions you make, and ensure you document what your logs contain.	Few

See also denial of service (Figure and Tables B14) and tampering with data stores (Figure and Tables B-6).

Information Disclosure from a Process

Figure B-9 shows an attack tree for information disclosure from a process. Information disclosure threats are generally covered in Chapter 3, and Chapter 8.

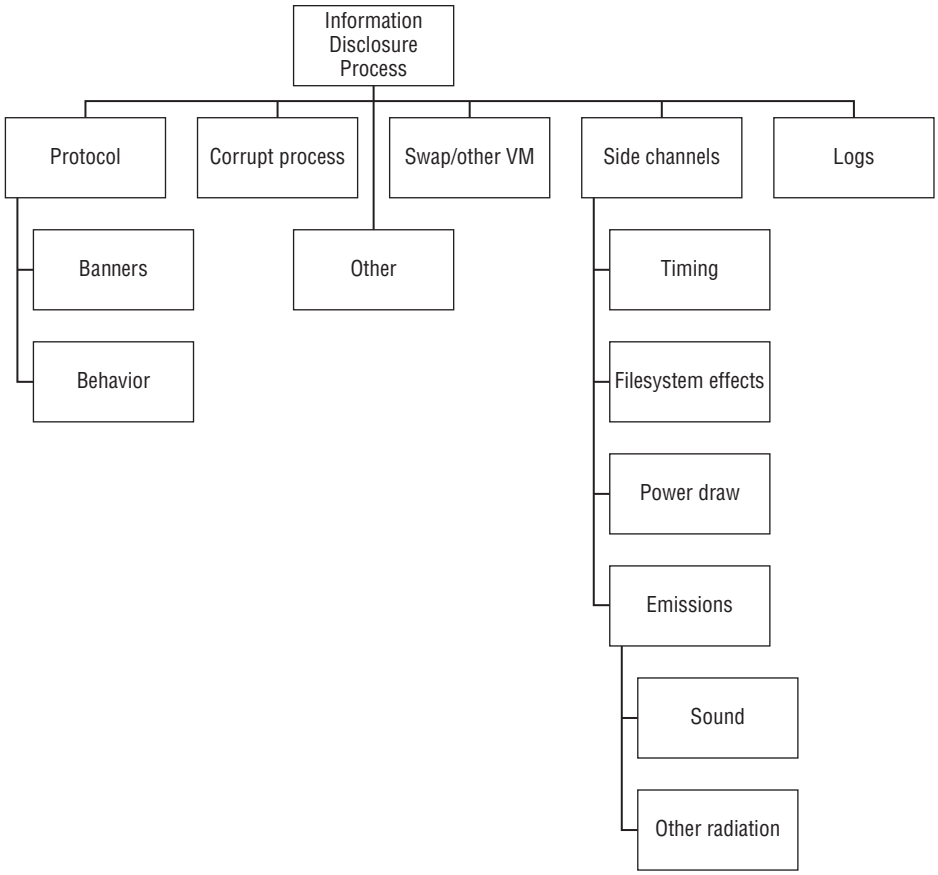


Figure B-9: Information disclosure from a process

Goal: Information disclosure from a process

- Side channels
  - Timing
  - Power draw
  - Filesystem effects
  - Emissions
    - Sound
    - Other radiation

- Protocol
  - Banners
  - Behavior
- Logs
- Corrupt process
- Other
- Swap/Virtual memory

As a class, side channels are like metadata; they're unintentional side-effects of computation, and they are often surprisingly revealing.

Table B-9a: Information Disclosure via Side Channels Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Timing	How long the code takes to complete can reveal information about its secrets, especially cryptographic secrets.	Design for cryptography to take constant time. Yes, that's annoying.	None
Power draw	Power draw can be surprisingly revealing about operations.	Cryptographic blinding can help.	If you have power supplies that monitor draw, ensure the logs they produce are protected.
Filesystem effects	Code will often write revealing information to disk.	Create a private directory and put your interesting tidbits in there.	Separate VMs.
Emissions			
Sound	Startlingly, processors make sound that can be used to learn about cryptographic keys (Shamir, 2013).	Architect to keep untrustworthy parties off (and far from) machines with important keys.	Remove microphones.
Other radiation	Other forms of radiation, including van Eck (sometimes called "TEMPEST") and light can reveal information.	There are fonts that are designed to make van Eck attacks challenging; they may violate rules regarding accessibility.	Shielding

Table B-9b: Information Disclosure via Protocol Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Banners	If your process announces what it is (for example, "HELO sendmail 5.5.1"), that may be valuable to an attacker.	Consider the risk trade-offs for banners; what value do you get by revealing a version?	Sometimes, banners can be altered; the value of doing so may or may not be worth the effort.
Behavior	Oftentimes, new versions of a program behave differently in ways that an attacker can observe.	It can be hard to avoid subtle behavior changes when you update code for security purposes.	Ensure that your security does not depend on keeping the versions you're running secret.

Table B-9c: Additional Information Disclosure Threats Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Logs	Logs often contain important process data. Ensure that you log the right data to the right logs. (For example, log login failures to a log that only admin can read; don't log passwords at all.)	If you control the logs (e.g., a database system with its own logs), design your logs with information disclosure in mind. If you're using system logs, don't attempt to change permissions on them.	Ensure that you keep log permissions properly set.
Swap/Virtual memory	Important secrets like cryptographic keys should not be swapped out.	Use the appropriate system calls to protect them.	No extra activity
Corrupt process	If you can tamper with or elevate privileges against a process, you can use that to disclose information.	Use a security development life cycle to prevent these.	None

Information Disclosure from a Data Flow

Figure B-10 shows an attack tree for information disclosure from a data flow. Information disclosure threats are generally covered in Chapter 3, Chapter 8 and Chapter 16.

Generally, if you don't prevent spoofing of a data flow, you have tampering and information disclosure problems.

Data flow threats can apply to channels or messages, or both. The difference is easiest to see with examples: E-mail messages travel over an SMTP channel, whereas HTML (and other format) messages travel over HTTP. The question of which you need (either, neither, or both) is a requirements question.

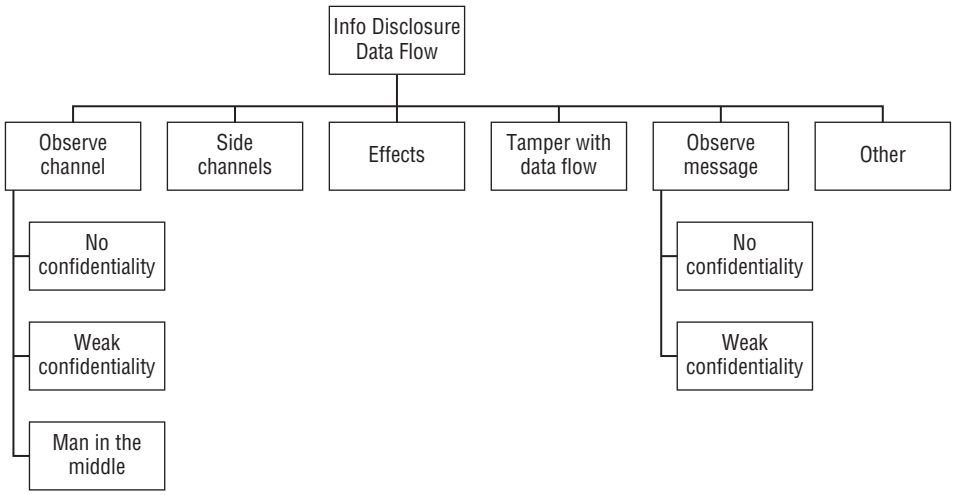


Figure B-10: Information disclosure from a data flow

Goal: Information disclosure from a data flow

- Observe message
  - No confidentiality
  - Weak confidentiality
- Observe channel
  - No confidentiality
  - Weak confidentiality
  - Man in the middle (See Tables B3 spoofing data flows.)
- Side channels
- Effects
- Other and spoofing

See also tampering with data flows (Figure and Tables B3 and B5, many of which can lead to information disclosure.

Table B-10a: Information Disclosure by Observing a Message Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
No confidentiality	Nothing protects the contents of a message.	Cryptographic (or permissions for on-system flows)	There may be message protection add-ons available, such as PGP, or tunneling for channel protection may help.
Weak confidentiality	The confidentiality of messages is weakly protected.	As above	As above

Table B-10b: Information Disclosure by Observing a Channel Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
No confidentiality	Nothing protects the contents of the channel. Even if you have good message protection, data about the messages (who talks to whom) can be revealing.	Encrypt the entire channel. If you're worried about who talks to whom, see B-10c.	Tunneling
Weak confidentiality	The contents of the channel are weakly protected.	Improve the encryption.	Tunneling
MITM	See B3, spoofing data flows.		

Table B-10c: Other Information Disclosure Threats Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Side channels	Data about who talks to whom can be interesting. See the discussion of traffic analysis in Chapter 3.	See Chapter 3.	Private network connections may help.
Effects	However well you protect the data flows, sometimes you take action, and those actions can be revealing.	None	Operational discipline

Information Disclosure from a Data Store

Figure B-11 shows an attack tree for information disclosure from a data store. Information disclosure threats are generally covered in Chapter 3, Chapter 8, and Chapter 16.

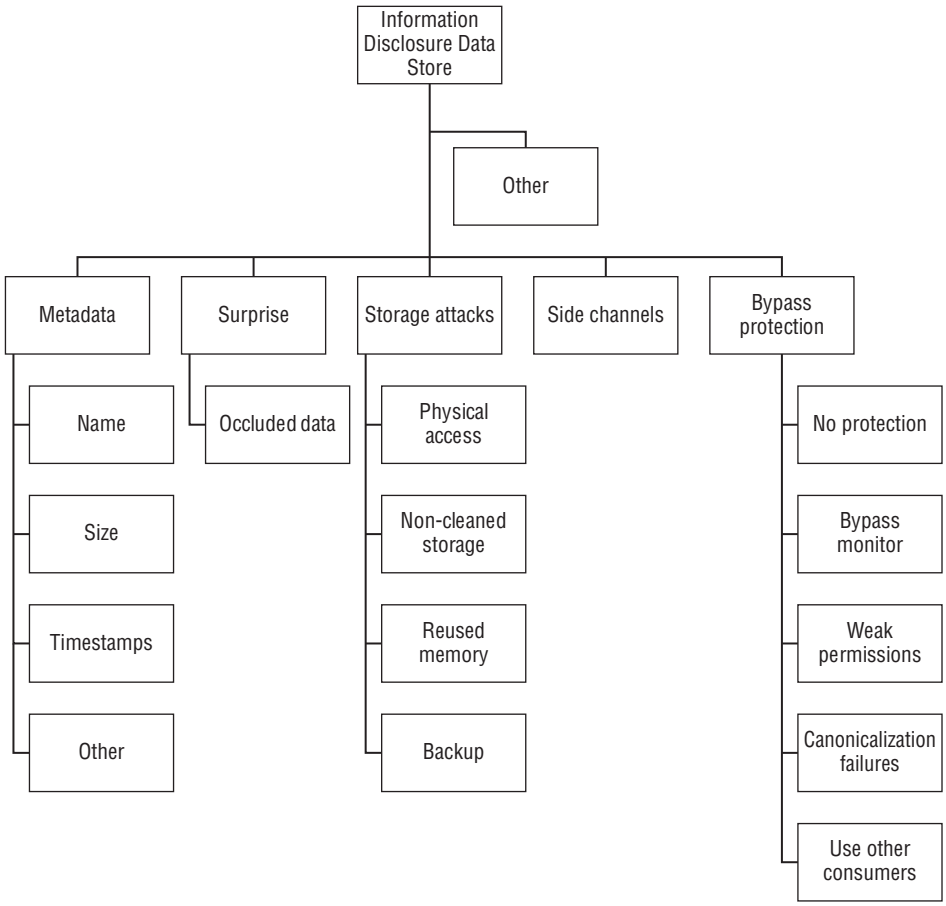


Figure B-11: Information disclosure from a data store

Goal: Read from a data store

- Bypass protection
  - Canonicalization failures
  - No protection
  - Bypass monitor

- Weak permissions
- Use other consumers

- Metadata
  - Name
  - Size
  - Timestamps
  - Other
- Surprise
  - Occluded data
- Side channels
- Storage attacks
  - Physical access
  - Non-cleaned storage
  - Reused memory
  - Backup
- Other

Table B-11a: Information Disclosure by Bypassing Protection Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Canonicalization failures	Can a data element have multiple names—for example, through links or inclusion? If so, different names may be processed differently.	Check permissions on the resolved object (after name canonicalization).	Probably none with reasonable effort
No protection	The system offers no protection, perhaps by design.	Use another system, or add protection.	Physical protection
Bypass monitor	Exploit the lack of a "reference monitor" through which all access requests pass, or take advantage of bugs.	Redesign is probably easier than the sorts of extensive testing that might find all the bugs.	Use a better system.

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Weak permissions	The permissions are too permissive.	Stronger permissions	Stronger permissions
Use other consumers	Find another program that can read the data and use it to read for you.	Don't open arbitrary files and pass on their contents.	Remove or block that program with permissions.

Table B-11b: Information Disclosure through Metadata and Side Channels Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Name	A filename can reveal information, such as "Plans to layoff Alice.docx".	Private directories, and allowing people to define where data is stored	Permissions
Size	The size of a file can reveal information.	Ensure that your files are of standard size. (This is rarely needed but it might be for your data types.)	Permissions
Timestamps	When a file is created can reveal interesting information.	Private directories, and allowing people to define where data is stored	Operational security to conceal timestamps
Side channels	Aspects of behavior such as a disk filling up or being slow can reveal information.	Possibly quotas, pre-allocating space to conceal actual usage	Reducing side channels is a large investment.
Other	If you're concerned about the preceding issues, there's a wide variety of ways to be clever.	Steganography, encryption	Isolation

Table B-11c: Information Disclosure from Surprise Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Occluded data	Data for purposes such as change tracking, etc., can be revealing.	Tools to help view, inspect, or remove such data	Procedures and training for publication

Table B-11d: Information Disclosure via Storage Attacks Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Physical access	Physical access is awesome if you believe your operating system will protect you.	Encryption	Physical security
Non-cleaned storage	When you release storage, does the OS clean it?	Manually overwrite sensitive data, usually repeatedly. Note that this works poorly with flash-based storage.	Destroy disks, rather than resell them. You can overwrite spinning media, but flash storage wear leveling makes information disclosure threats hard to manage.
Reused memory	When you release storage, does the OS clean it?	As above	None
Backup	What happens to those offsite tapes?*	Cryptography	Cryptography

* Threats to backup can also allow tampering. However, completing a tampering attack with a backup tape is far more complex than information disclosure through such tapes.

Denial of Service against a Process

Figure B-12 shows an attack tree for denial of service against a process. Denial-of-service threats are generally covered in Chapter 3, and Chapter 8.

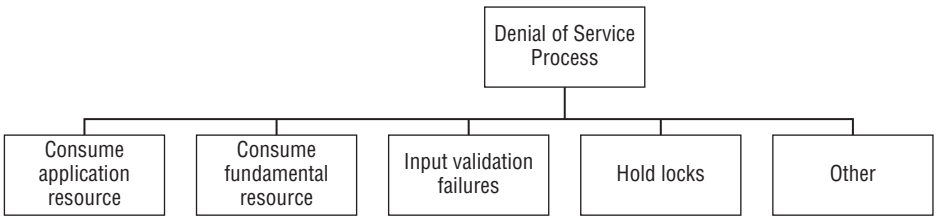


Figure B-12: Denial of service against a process

Goal: Denial of service against a process

- Consume application resource
- Consume fundamental resource
- Input validation failures
- Hold locks
- Other

Table B-12: Denial of Service against a Process

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Consume application resource	Application resources include connections, or buffers or structures that manage business logic.	Dynamically allocated resources	VMs or load balancing
Consume fundamental resource	Fundamental resources include disk, memory, or bandwidth.	Use OS quotas.	VMs or load balancing
Input validation failures	An input failure that crashes the application	Careful input validation	Process restarting. (Be careful that you don't allow DoS to turn into EoP by giving attackers as many chances as they need.)
Hold locks	To the extent that an application can hold locks, it can deny service to other locks.	Give up your locks quickly.	One VM per application

Denial of Service against a Data Flow

Figure B-13 shows an attack tree for denial of service against a data flow. Denial-of-service threats are generally covered in Chapters 3, and Chapter 8.

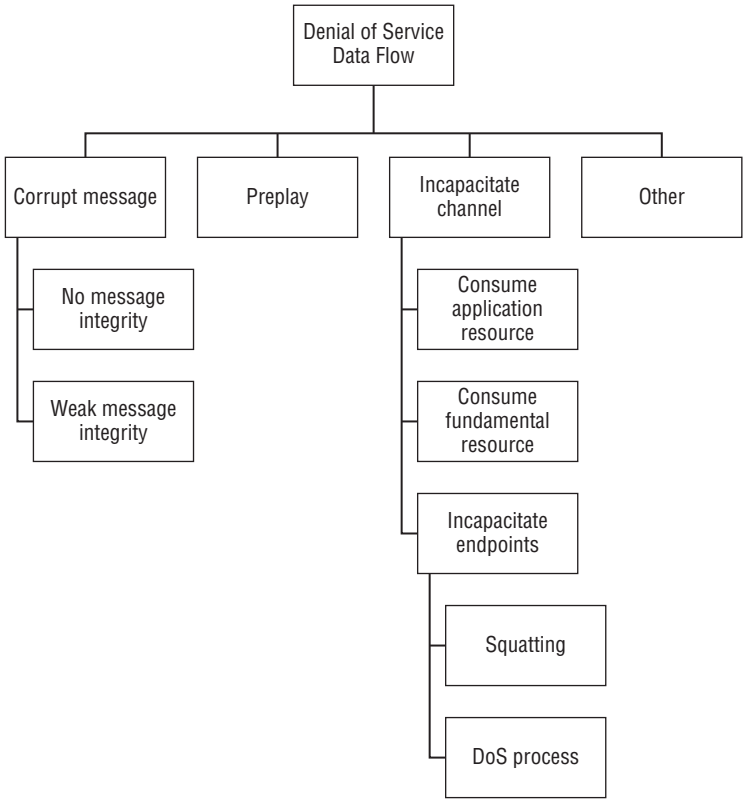


Figure B-13: Denial of service against a data flow

Goal: Denial of service against a data flow

- Preplay
- Corrupt messages
  - No integrity
  - Weak integrity
- Incapacitate channel
  - Consume fundamental resource
  - Consume application resource
  - Incapacitate endpoints
    - Squatting
    - DoS against process
- Other

Table B-13a: Denial of Service via Preplay Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Preplay	An attacker initiates a connection or action before you, absorbing cycles.	Proof of work doesn't work, but consider dropping connections that seem slow.	Mmm, capacity

Table B-13b: Denial of Service via Corrupt Messages Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
No integrity	If the channel has no integrity, an attacker along or close to the path can corrupt messages.	Add integrity checks.	Add capacity or, if feasible, tunneling.
Weak integrity	Similar to no integrity, but with an unkeyed checksum, or a non-cryptographic checksum	Use a cryptographically strong checksum with keying.	Add capacity or, if feasible, tunneling.

Table B-13c: Denial of Service By Incapacitating a Channel Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Consume fundamental resource	Typically, bandwidth is the fundamental resource, but it can also be CPU.	Use TCP, not UDP, so you can at least require that endpoints be able to respond.	Bandwidth—sweet, sweet bandwidth
Consume application resource	Anything that your application can provide can be consumed. For example, if you have a state-based firewall with static tables of state data, that table can be filled.	Avoid fixed allocations.	VMs and load balancers

Continues

Table B-13c (*continued*)

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Incapacitate endpoints	Do something to kill the endpoint.	Ensure your end-points can't be incapacitated.	fail-over
Squatting	Show up before the real application to claim a port, a named pipe, etc.	None	Permissions
DoS against process	Cause the process to spin in some way, making the channel unusable.	As above	VMs and load balancers

Denial of Service against a Data Store

Figure B-14 shows an attack tree for denial of service against a data store. Denial-of-service threats are generally covered in Chapter 3, and Chapter 8.

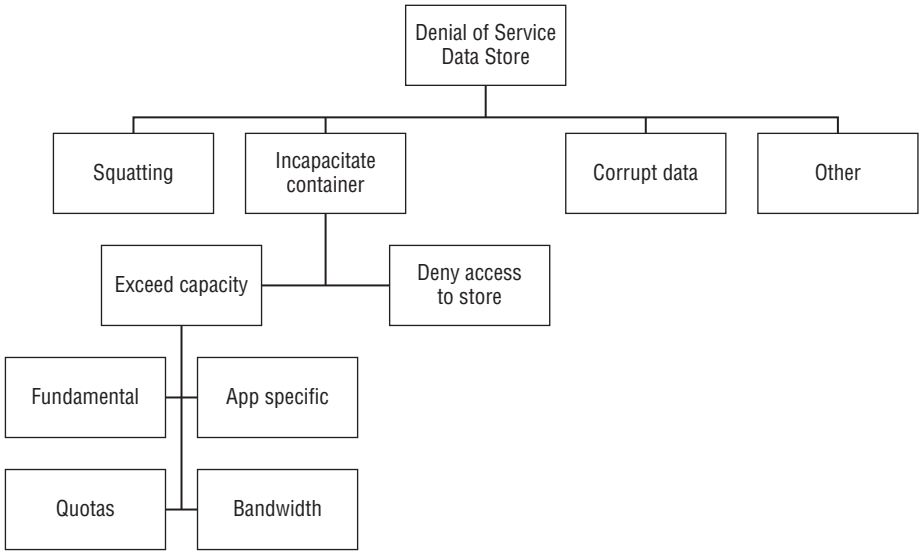


Figure B-14: Denial of service against a data store

Goal: Denial of service against a data store

- Squatting

- Corrupt data (See tamper with data store, Figure B-6.)
- Incapacitate container
  - Deny access to store
  - Exceed capacity
    - Fundamental
    - App specific
    - Quotas
    - Bandwidth
- Other

Exceed capacity or bandwidth means I/O operations per second, and if sustained, that can have knock-on effects as I/O is queued.

Table B-14a: Deny Service by Squatting Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Squatting	Show up before the real application to claim a port, a named pipe, etc.	Assign permissions to the object in question.	Assign permissions to the object in question.

Table B-14b: Deny Service by Incapacitating a Container Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Deny access to store	An attacker might deny access to a store by adding an ACL or taking and holding a lock.	Put the store somewhere that attackers are limited in their ability to add ACLs or locks. Test to see if you can access the store; fail gracefully.	Move the store somewhere an attacker can't change permissions, possibly using a link to redirect.
Exceed capacity	Fill the data store in some way, either fundamental, or app specific, or by hitting quotas or bandwidth constraints.		

Continues

Table B-14b (*continued*)

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Fundamental	Absorb a fundamental resource such as disk space.	Fail gracefully.	More storage
App specific	If the application has resource limits, fill them up; for an example, see US-CERT, 2002.	Avoid fixed allocations.	None
Quotas	A quota can work like a fundamental resource restriction, while holding the DoS to a single application or account, rather than a system.	Design choices about appropriate trade-offs	Deployment choices about appropriate trade-offs
Bandwidth	Even if the data store can hold more data, network bandwidth or buffers on either end of a connection can fill up.	Dynamic buffers may help postpone the issues.	More bandwidth, or (especially in cloud/data centers) move the processes closer to the system that is writing.

Elevation of Privilege against a Process

Figure B-15 shows an attack tree for elevation of privilege against a process. Elevation of privilege threats are generally covered in Chapter 3, and Chapter 8.

Goal: Elevation of privilege against a process

- Dynamic corruption
  - Input validation failure
  - Access to memory

- Static corruption (See tamper with data store, Figure B-6.)
- Insufficient authorization
  - Cross domain issues
  - Call chain issues
  - Design issues
    - Usability

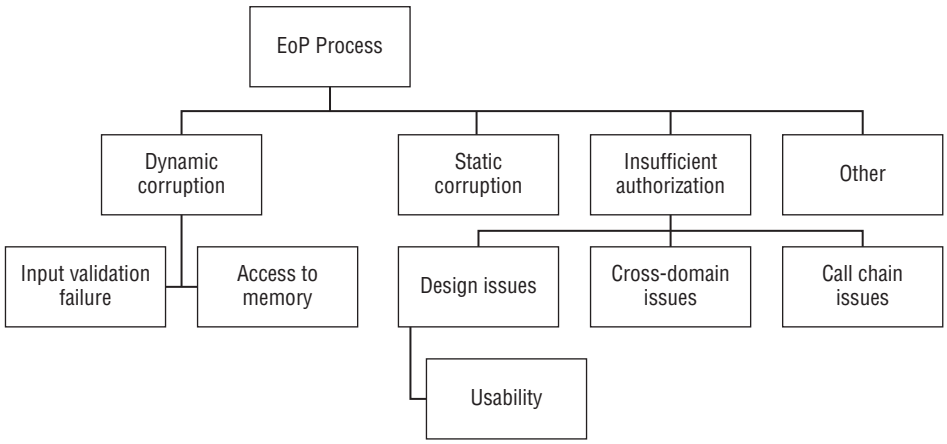


Figure B-15: Elevation of privilege against a process

Table B-15a: Elevate Privilege by Dynamic Corruption Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Input validation failures	Input can alter control flow (e.g., via stack smashing or heap overflow).	Careful design, input validation for purpose, fuzzing	Sandboxing can partially mitigate the effects.
Access to memory	Sometimes code will attempt to defend against administrators or other local accounts.	See "Tampering with a Process".	Use the OS.

Table B-15b: Exploit Insufficient Authorization to Elevate Privileges Subtree

TREE NODE	EXPLANATION	DEVELOPER MITIGATION	OPERATIONAL MITIGATION
Cross-domain issues	Applications that use a "same origin" policy or a "no cross domain" policy can be impacted by failures in that security model. These models are frequently used in web applications.	Ensure that security checks are centralized into a reference monitor that makes names canonical and then runs checks. Be careful with common domains and DNS issues that can result from load balancing or cloud services. (For example, is Amazon S3 or Akamai, along with all their customers, inside your trust boundary?)*	Your own domain(s) will probably help.
Call-chain issues	See "Tampering with a Process (Figure B4) above.		
Design issues (usability)	If your authorization system is hard to use, people are less likely to use it well.	Perform human-factors tests early in the design.	Tools to analyze permissions

* Amazon and Akamai are mentioned to be evocative, not to disparage their services.

Other Threat Trees

These trees are intended to be templates for common modes of attack. There are several tensions associated with creating such trees. First is a question of depth. A deeper, more specific tree is more helpful to those who are experts in areas other than security. Unfortunately, through specificity, it loses power to shape mental models, and it loses power to evoke related threats. Second, there is a tension between the appearance of completeness and the specificity to an operating system. For example, exploit domain trust is Windows specific, and it can be derived from either "abuse feature" or "exploit admin (authentication)" or both, depending on your perspective. As such, consider these trees and the audience who will be using them when deciding if you should use them as is or draw more layers.

Unlike the STRIDE trees shown earlier, these trees are not presented with a catalog of ways to address the threats. Such a catalog would be too varied, based on details of the operating systems in question.

Running Code

The goal of running code can also be seen as elevation of privilege with respect to a system. That is, the attacker moves from being unable to run code (on that system) to the new privilege of being able to run code (on that system). These trees also relate to the goal of exploiting a social program, which is presented next. The key difference is that the trees in this section focus on "run code," and that is not always an attacker's goal. These trees do not contain an "other" node, but the categories are designed to be broad.

On a Server

In this context, a server is simply a computer that does not have a person using it, but rather is running one or more processes that respond to network requests. The goal is "run code" rather than "break into," as breaking in is almost always a step along a chain, rather than a goal in itself, and the next step toward that goal is almost always to run some program on that machine. This tree, as shown in Figure B-16, does not distinguish between privilege levels at which an attacker might run code.

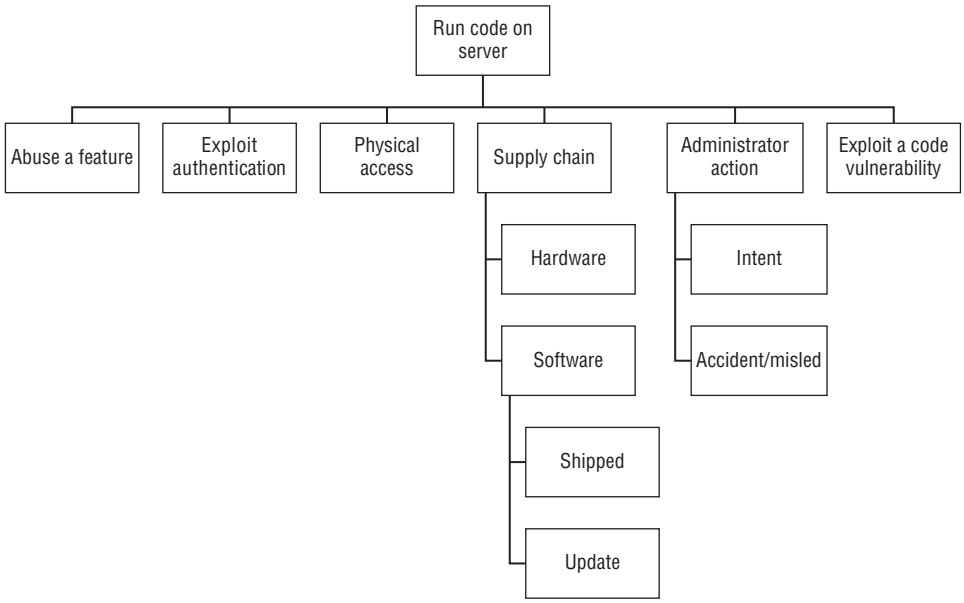


Figure B-16: Run code on a server

Goal: Run code on a server

- Exploit a code vulnerability
  - Injection vulnerabilities

- Script injection
- Code injection (including SQL and others)
- Other code vulnerabilities
- Abuse a feature
- Exploit authentication
- Physical access
- Supply chain
  - Tamper with hardware
  - Tamper with software
    - As shipped
    - Tamper with an update
- Administrator action
  - Intentional
  - Accident/mislead

Tampering with software updates may be targetable or may require a broad attack. Injection vulnerabilities are a broad class of issue where code/data confusion allows an attacker to insert their code or scripts.

On a Client

All of the ways that work to break into a "server" work against a client as well, although the client may have less network attack surface. As shown in Figure B-17, the new ways to break in are to convince the person to run code with additional or unexpected functionality or to convince them to pass unsafe input to code already on their machine. You can either convince them to run code knowing they're running code but not understanding what impact it will have, or confuse them into doing so, such as by using a file with a PDF icon and displayed name which hides its executable nature. (See the section below on "Attack with Tricky Filenames.") If you're convincing a victim to pass unsafe input to a program on their machine, that input might be a document, an image, or a URL (which likely will load further exploit code, rather than contain the exploit directly).

These attacks on the person can come from a variety of places, including e-mail, instant message, file-sharing applications, websites, or even phone calls.

Goal: Run code on a client

- As server
- Convince a person to run code
  - Extra functionality

- From a website
- Tricked into running

- Convince person to open an exploiting document (possibly via a document, image, or URL)
- Watering hole attacks
  - Website
  - Filesharing
  - Other
- "What's this?"

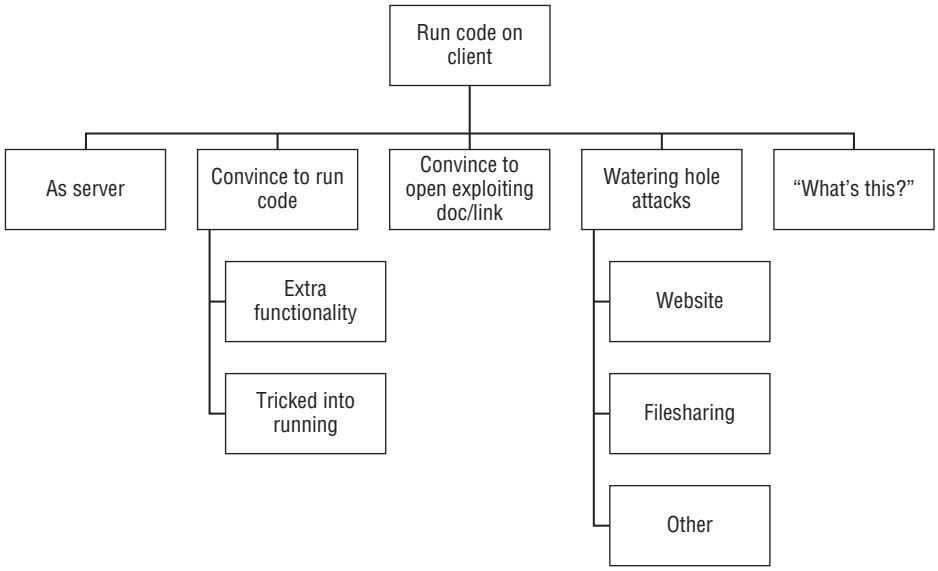


Figure B-17: Run code on a client

Those paying attention might notice that this is informed by the Broad Street Taxonomy, as covered in Chapter 18, "Experimental Approaches." Watering hole attacks are similar to "convince person to open an exploiting document," but they target a possibly broad class of people. That may range from targeting those interested in an obscure government website to those visiting a programming site. (Attack code in either place may involve some discretion based on target IP, domain, or other factors.) Watering hole attacks can also impact those downloading content from file-sharing services and the like. The last category ("What's this?") refers to attacking a person via a file on a file share, USB key, or other device, so curious users click an executable because they're curious what it does.

On a Mobile Device

The term *mobile device* includes all laptops, tablets, or phones. Attacks against mobile devices include all client attacks, and physical access has added prominence because it is easier to lose one of these than a refrigerator-size server. The additional attacks are shown in Figure B-18.

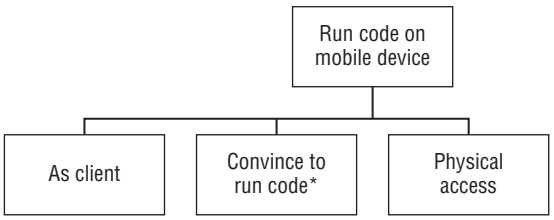


Figure B-18: Run code on a mobile device

Goal: Run code on a mobile device

- As client
- Convince to run code* (may be modified by app stores)
- Physical access has greater prominence

*Convincing someone to run code may be changed by the presence (or mandate) of app stores. Such mandates lead to interesting risks of jailbreaks carrying extra functionality.

Attack via a "Social" Program

For longer than the Internet has been around, people have been attacked at a distance. Most countries have a postal police of some form whose job includes preventing scammers from taking advantage of people. The Internet's amazing and cheap channels for connecting people have brought many of these online, and added a set of new ways people can be taken advantage of, such as exploiting vulnerabilities to take over their computers.

The threat tree shown in Figure B-19 can be used in two ways. First, it can be used as an operational threat model for considering attack patterns against e-mail clients, IM clients, or any client that a person uses in whole or in part to connect with other people. Second, it can be used as a design-time model to ensure that you have defenses against attacks represented by each part of the tree.

Goal: Attack via a "social" program

- Run code
  - Unattended exploit
  - Exploit vulnerability via document

- Reason to act + belief in safety
- Belief in safety

- Human action
  - Visit a website
    - Phishing
    - Exploit vulnerability*
    - Ad revenue
    - Other
  - Drive other action

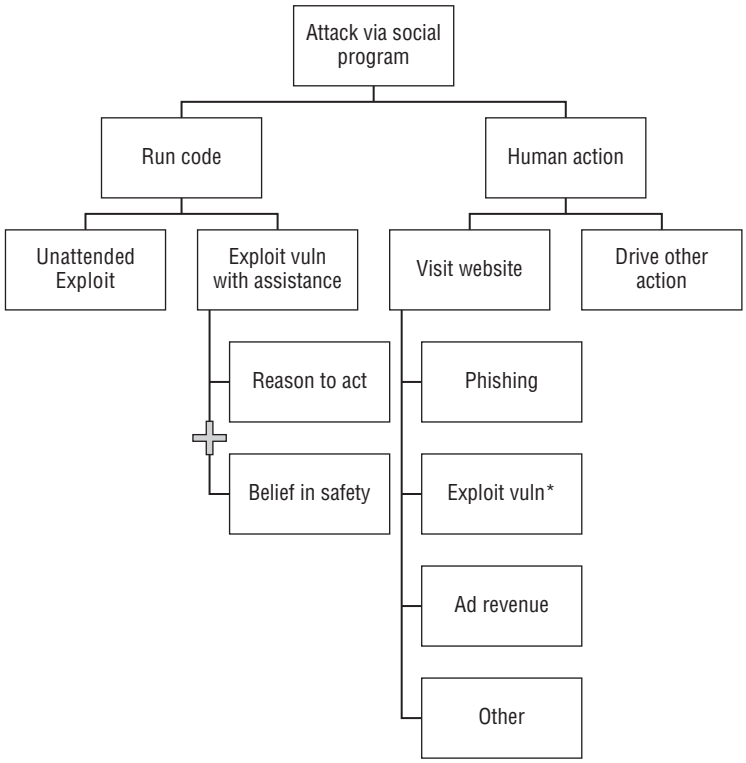


Figure B-19: Using a social program to attack

*The "Exploit vulnerability" reason to get a person to visit a website means that the attacker does something to convince a person to visit a website where there's exploit code waiting. It is duplicative of the "exploit vuln with assistance." It is sufficiently common that it's worth including in both places. (There is an alternative for attackers, which is to insert their exploit into an advertisement shown on the web.)

This tree intentionally treats attacks by administrators or the logged-in account as out of scope. The "logged-in account" here refers to acts taken either by that person or by code running with the privileges of that account. Once code is inside that trust boundary, it can most effectively (or perhaps only) be managed by code running at a higher privilege level. Almost by definition that is out of scope for such a social program. You might have an administrative module that runs at higher privilege and, for example, acts as a reference monitor for certain actions, but code running as the logged-in account could still change the user interface or generate actions.

Attack with Tricky Filenames

A simple model of convincing a person to act requires both a reason for them to act and a belief that the act is safe, or sufficiently safe to override any caution they may feel. The reasons that attackers frequently use to get people to act are modeled in Table 15-1 of Chapter 15. A belief in safety may come from a belief that they are opening a document (image, web page, etc.) rather than running a program. That belief may be true, and the document is carrying a technical exploit against a vulnerability. It may also be that the document is really a program, and named in a way that causes the user interface to hide parts of the name. Hiding parts of the name can be a result of extension hiding in Mac OS, Windows, or other environments. It can also be a result of various complex issues around mixing displayed languages whose text directions are different (read left to right or right to left). For example, what's the correct way to write Abdul's Resume? Should it be resume عبد? If, rather than having that resume in a file named .doc, it's a .exe, where should the .exe be displayed relative to عبد? On the one hand, the Arabic عبد should be displayed farthest right, but that will confuse those who expect the extension there. If the extension is on the far right, then the Arabic is displayed incorrectly. It's easy for a person to get confused about what a file extension really is.

Entertainingly, Microsoft Word took the text entered as "Abdul (in Arabic) Resume" and re-rendered it as "R" "e" "s" "u" "m" "e" "L/lam" etc. Abdul, and then re-arranged the question mark typed after the name to be between those words. (Which just goes to show...the issues are complex, and there's no obviously right answer.)