Chris Cargile	CSCI656	April 21, 2014	Ch.13 HW Questions

Answering assigned questions from the Chapter required 3 h itself; reading the textbook required 1.5 hours, plus 1 hours reading about Firefox security mechanism (see footer on pg.2)

13.6.1) What is security? What are the properties that a software system must exhibit in order to be called secure? (15 mins.)

Security is an architectural design NFP which involves preserving the integrity, availability and confidentiality of information system resources. Software systems must allow only authorized parties to manipulate information, only in authorized ways (integrity) and/or even accessing information or functionalities of them (confidentiality and secrecy) and they must furnish the requested resources *always* when they are authorized to do so (availability) to exhibit the property of software security.

13.6.3) What challenges to security are introduced by decentralized applications? What is trust management? (20 mins.)

Threats to software systems induced by decentralized applications' include physical and abstract threats. Decentralized applications are susceptible to attackers based on threats that include impersonation; fraud; misrepresentation; collusion; D-o-S; the risk of users who enter into the system without references validating their trustworthiness; and other possibilities for manipulating 'trust scores' in the system, either directly or through out-of-band manipulations[1].

Trust management involves effort that is aimed at countering such threats so entities may establish and maintain trust relationships with each other. Trust management considers trust models considered by academics, and various reputation-based systems reflect such considerations in their architecture.

The considerations of systems which may exhibit conformance to the security and trust NFP includes possible options among: using authentication, separating internal beliefs and externally reported information, marking trust relationships explicit, establishing comparable trust (that is portable?), using digital identities, and making trust visible.

13.7.2) What are the security benefits and the security risks associated with the following architectural styles from Ch.4? (30 mins.)

a) pipe-and-filter: Connectors are represented by OS 'routers' in this style that transport the payload between the filtering components, so this system is only

[1] Out-of-band manipulations entail transitively affecting a trust perception by skewing the trust score of a mutually visible trust source so that an outside observer of a trust source sees the manipulated trust sources affected trust score and changes his/her score based on the fraud.

as secure as the components and connectors of the style permit. The risks faced by architectures of this style include all risks associated with any architectural style. The benefits of using this style are that the principle of least privilege may be achieved if the data source being piped in and access to OS processing mechanisms is restricted. In this way other principles of security can also be adhered to, utilizing access controls to protect the relevant aspects (data sources; and OS processing mechanisms, or interfaces to them) to this style. Critical resource sharing should not be a significant factor for implementers and architects unless they are designing this style upward from the OS level.

b) **blackboard:** The blackboard style affords the advantage of ensuring the principle of least common mechanism is achieved when data on the shared blackboard can be mediated, if the system design is adequate. Access controls might be used to protect the system if only authorized users can enter into the system to initiate exchanges with the shared blackboard/system.

c) **event-based:** protocols used to represent connectors in this architectural style pose the same risks to security as any in system's design – if the protocol is complex or proprietary it is likely to violate two principles: economy of mechanism and *open design*. Advantages afforded by this style includes usage of a singular interface through which components interact and messages are passed – the event bus – which can be capitalized upon by applying a uniform contract for securely interacting with the bus and ensuring all interactions are explicitly permitted. An implementation requiring explicitly authorized interactions which are uniformly ensured ensures this style is conformant to fail-safe defaults and complete mediation principles.

d) **C2:** the usage of a single connector type in the C2 style may make it so C2 lends itself to achieving conformance to the "economy of mechanism" principle if all components interact use the same basic security schema for engaging with the connector. Risks posed by this architectural style are many, like most styles (all of the risks to integrity and confidentiality within a system are posed by it).

13.7.3) Choose any software application, such as Firefox, and use UML sequence diagrams to show how the security mechanisms within the application

operate. (55 mins.)

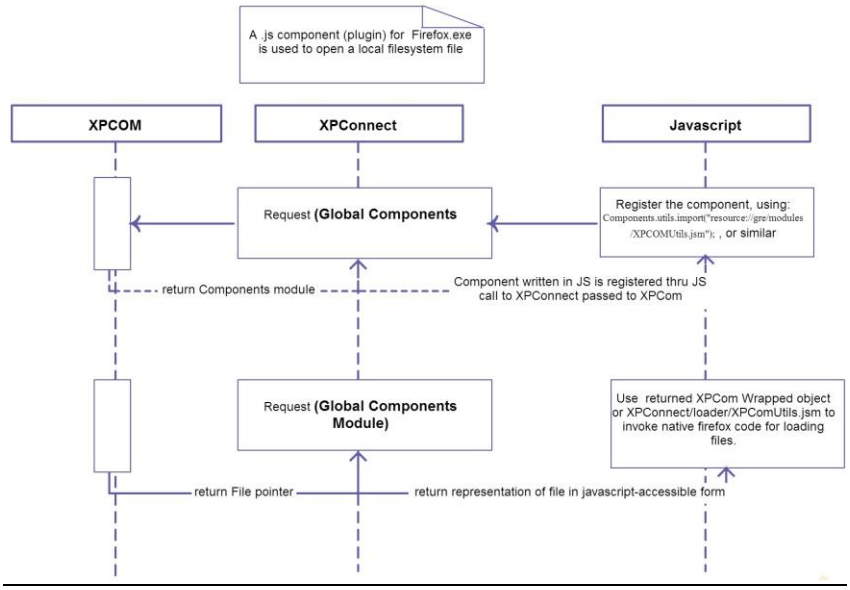

Figure: Sequence diagram of Firefox.exe Security Mechanisms

Links for pages that were used in understanding the mechanisms for building the above diagram (for a Javascript component which accesses local filesystem files) include:

- Creating Components: An overview of XPCOM
- XPConnect security membranes
- How to build an XPCOM Component in Javascript
- Example JS Component Code: (Reference Lines 279-289): demo code for obtaining a File pointer via native/XPCom capabilities