Security challenges for internet technologies on mobile devices

- Geir Olsen [geiro@microsoft.com] , Senior Program Manager for Security – Windows Mobile, Microsoft Corp.
- Anil Dhawan [anild@microsoft.com], Program Manager for Internet Applications – Windows Mobile, Microsoft Corp.

The number of smart phone devices capable of offering Internet technology and experiences rivaling desktop computer standards is growing at a fast pace. Security and Privacy concerns for mobile devices rival or go beyond similar concerns for laptop computers as mobile devices are even more mobile by nature and are less likely to be "managed" by an IT organization.

In this paper we will discuss the challenges that exist and outline a set of principles and ideas we will use to frame a discussion on how to address security and privacy in the realm of Internet technology on mobile devices.

The paper will focus discussion on traditional web sites, web applications and "Widgets".

A web page can contain content from multiple sources (or software publishers); scripts download from these sources and execute concurrently in the web browser. Web browsers offer increasingly rich programming models and execution environments with some exposing programming abstractions for core operating system and device services like location services and camera.

Widgets are considered user experiences developed using web technologies and are typically *installed* to run locally from the device. The widget framework offers its own programming model rooted in the browser document object model coupled with interfaces that offer access to core operating system services such as HTTP/HTTPS/TCP, location services, and others.

Mobile operators and mobile device operating system vendors have security and privacy concerns for all software running on their devices. They often make no conceptual distinction between scripts running in the context of a browser and traditional software installed on the device.

Mobile operators, device manufacturers, and software vendors are making attempts at formulating security architectures for mobile code. Examples include the MIDP 2.0 security architecture and the OMTP application security framework. We see benefit from standards bodies working to minimize security and privacy risks for Internet technologies on mobile devices.

The Questions that need answers

1. How are Web page scripts and Widgets different from "native" applications?
2. What are the criteria for assessing trust?
3. What are the key elements of risk management and mitigation?
4. How should code identity be securely issued, managed and verified?
5. How should intent of code be disclosed and discovered?
6. What does it mean to act on intent, reputation and reliability information?
7. How should device capabilities be defined and discovered?

How are Web page scripts and Widgets different from "native" code?

Code downloaded and executed in the context of web pages and in web browsers differ from native applications as follows:

1. **Deployment model:** Scripts typically load and run with the web page which are downloaded or deployed from web servers. Scripts on a single web page can originate from many sources. There is no commonly accepted standard for assigning software identity to web page scripts.
2. **Available Programming Model:** Scripts downloaded in the context of a web page will have the DOM available through which the scripts gain access to browser and system resources. The trend is toward richer programming models providing access to an increasing number of OS and device services.
3. **Security Model:** Traditional software applications installed on a device has an assigned identity through which the code is awarded capabilities. There is no such model for scripts running in the context of a web page. In the case of native code the identity is typically traceable to the issuing authority of a digital certificate. There is no such widely accepted technology available to script running in a browser. Instead scripts are constrained by the object model and programming model they have at their disposal. Some OS security models allow for self-disclosure of intent through manifests. There is no similar infrastructure in web browsers.

It is expected from developers and open web sites alike that the user should be able to navigate from one end of the web site to another without being interrupted by security prompts. This is largely the case with the exception of those sites reliant on locally installed components to do their work.

What are the criteria for assessing trust?

What constitutes the criteria for assessing risk and/or trust for a software running on a device? In extensible software systems the degree to which a software component can be trusted is determined by:

1. The system's ability to securely associate the software's identity with a trusted authority
2. The system's ability to securely discover the software's intent
3. The system's ability to discover and act on intent, reputation and reliability information related to the software
4. The system's ability to provide a least privilege environment for the software
5. The system's ability to isolate the software from other software on the device
6. The system's ability to securely and in a timely fashion correct/ remove the software if needed

Some of these criteria are related to the software designed to run on the mobile device, others to the OS and the device, and others again to the deployment pipeline.

Risk Factor	High Risk	Low Risk	Comment
Identity	Unverifiable	Verifiable	
Intent (publisher)	Undiscoverable	Discoverable	
Intent Discovery and Response (device)	Broad Access	Custom Sandbox	The risk is lower if device architecture can act on discoverable intent
Publisher Reputation	Low	High	
Publisher Title Reliability	Low	High	A publisher can distribute many solutions. While previous "reputability" measure is related to publisher directly. This measure is related to a particular title or solution.
Isolation	No Isolation	Isolated	This relates to the device's ability to isolate one publisher's solution from another
Revocation/ Blocking	No revocation/ blocking capabilities	Revocation and Blocking capabilities	

Security risk assessment based on criteria outlined above does not distinguish between applications installed on the device or scripts running in the context of a browser session.

How can code identity be securely issued, managed and verified?

Some mobile devices require that code be signed before it is allowed to execute. This requirement is typically not associated with web page scripts and widgets. The degree to which a digital signature is used differs from platform to platform. The Android platform requires signatures, but certificates used

for signing can be self-signed.The iPhone platform requires code to be signed twice (first with developer certificate, secondly with app store certificate) before it can be sold in the Apple AppStore. The iPhone authenticates code based on code identity and has or uses tamper proofing and revocation capabilities. Windows Mobile does not require all applications to be signed; only when certain capabilities are used is the requirement in place. Microsoft uses different certificate types to distinguish trust level at which the signed software can run. Windows Mobile authenticates code identity at run-time and has tamper proofing and revocation features. Unsigned code can run on Windows Mobile devices, but run at the lowest privilege level.

Code identity for scripts running in the context of a web page can be based on hash value of script or can be inferred from the URL and a trusted certificate associated with the domain the script was loaded from. Scripts on a web page can in turn load and execute scripts from other domains. Each such script can access local device services like the location API. For privacy and usability reasons it is important that unique identities can be securely inferred by these services from the execution context to prevent excessive user prompting and/ or access to services by un-trusted scripts.

Code identity and code identity authentication are key elements of a device security and trust model.

How can intent of code be disclosed and discovered?

One element of risk posed by a software component can be derived from what capabilities the software component needs to offer all of its use cases. In most current software systems such intent is not disclosed and thus cannot be discovered or acted on. The .NET Framework introduced declarative security when it was initially released. It is possible in .NET to discover a software component's security intent and to act on that intent. The Android and Symbian platforms offers no special capabilities to a software component that does not disclose intent and provides a least privilege environment for software that does.

The ability to declaratively disclose intent is one area of interest where we see standards being developed.

What does it mean to act on intent, reputation and reliability information?

If an application discloses intent in the form of a list of required capabilities, how should a device security platform act on that information? Software self-disclosure information can be used in the following ways (not an exhaustive or prioritized list):

1. The information can be disclosed to a user at install time to empower the user to make a more informed decision when deciding for or against installing the application.
2. The disclosure information can be used to create a least privilege environment for the software, an environment where only capabilities requested is granted.
3. The information can be used in the deployment pipeline as input to the risk and risk mitigation elements of that pipeline. Higher risk is associated with a greater expressed need for capabilities. Higher risk software normally takes a different path through the deployment pipeline involving testing to ensure the software meets a higher quality bar.

In regards to “reputation” and “reliability” information we are talking about defining and building public or commercial infrastructure to support aggregation of “reputation” or “reliability” related information. Many shopping sites have reputation related rankings, stipulated from customer input. Reliability related information could include the ability to have software tested and then signed by a testing authority. We should think about allowing the inclusion of reputation and reliability “claims” in the application manifest. This type of information would allow mobile operators and IT the ability to define what level of risk they are willing to accept on some or all of their devices. Government agencies sometimes have strict secrecy and privacy requirements, while on the opposite end of the spectrum some consumer are willing to take risks to gain functionality. A solution that allows the owner of the device flexibility when making risk management decisions is ideal.

It is imperative to find the right balance between our ability to better protect end users and organizations against security and privacy threats and proving ample room for software developers to innovate and without hampering their time to market requirements. It is also essential that our quest for security does not negatively impact the user experience.

Mobile operators differ on willingness to take security and privacy risk. Current software development models thus face the issue of platform fragmentation. If one mobile operator defines risk and signing requirements at level A and another mobile operator defines risk and signing requirements at level B, the ISV risk having to create two binaries and having to put the two binaries through two different certification schemes. This adds unnecessary bureaucracy. If we rather could define a standard method of disclosing intent and classifying security risks and mitigations and handle risk mitigation in the deployment pipeline it is possible that developers would largely remain unaffected.

How can device capabilities be defined and discovered?

The ability to discover available capabilities is an element of a disclosure and consent framework. Disclosure and consent is terminology often used in relation to Privacy requirements.

A manifest document might contain a series of capability assertions that an implementation can process to impart trust/authorization. A central challenge here is coming up with a common schema with an appropriate level of granularity to selectively allow/revoke privileges based on various criteria.

Trust

Trust is viewed as the overarching objective of most security models. Trust is also a key element of usability. Before an end user can benefit from the value proposition offered by a software solution the user must first trust themselves with the device enough to explore. A well crafted user experience can help by responding in ways aligned with expectations of user. Best possible security is achieved when devices behaves according to users’ expectations, while maintaining device integrity, protecting users, device and network assets with the highest possible degree of discretion. A user is often left feeling less secure when confronted with signs of security risk even when the user is confronted with a security component developed to protect the user in some way. In general everyone wants to be secure, but no one likes to deal with security directly.

Risk Management

Security is risk management. With little risk the need for security measures is low, and conversely with great risk the need for security measures is high. Accurate measures of risk are essential for effective enforcement of security.

All major vendors of mobile device operating systems are developing and evolving security models that to some degree already factor in the elements outlined above. The challenges are many. Challenges already confronted by desktop and server operating systems are becoming increasingly relevant for mobile devices as well.

Transparency & Prevention

End users are often asked to make security decisions with little or no information about the application at hand. On many devices applications or services are installed after a user has downloaded application setup files to their device. People make better decisions when better informed. Prevention is an area that is largely overlooked in regards to opportunity to prevent end users from making bad security decisions. More transparency is needed; we need to know more about a software publisher and the software publisher's track record/ reputation. Alternatively or additionally it would be desirable to disclose a measure of security risk regarding a piece of software before the software is downloaded to a user's device.

Many current smartphone users buy applications for their phone on the Internet. These applications are bought from one of many Internet stores that sell smart phone software. The most popular of these sites, for example Handango, offer rating based on user reviews, but no other information that indicates to what degree the software solution poses a security risk to the end users device. Greater transparency is needed from software publishers to allow Handango and other access to security risk information. Security risk information associated with a software solution should be published with the title to allow the curious end user the ability to browse such information enabling make more intelligent decisions.

Trustworthy User Experiences

Studies show end users as poor decision makers in relation to decisions on security and privacy. Even with blinking banners warning them of not entering their credentials on a web form, many users still do it. A lot of work is needed to develop trustworthy and consistent user experiences across software solutions. A more streamlined and contextual disclosure and consent experience is one element of such a strategy. Street signs are standardized in many countries in the world, better and standardized visuals used in the disclosure user experience will over time make it easier for users to make educated decisions. Additional metadata in the form of software disclosure statements will provide additional data allowing a better basis for end users explicit consent scenarios.

Disclosure and Consent

Disclosure and consent is often used in the context of privacy to describe the process of obtaining user consent to collect and share personally identifiable information. Prominent disclosure refers to a user experience where the end user prompted prominently about an impending privacy sensitive operation. Consent refers to the end user's affirmative feedback on the disclosure. For example, an application may have built in a prominent disclosure that prompts the end user before their email address is sent to the software publisher, informing the user what is about to happen and asking for the user's permission (consent). If the end user provides such consent, the data is sent. If the consent request is declined, the application does not send the data.

Asking end users to make security decisions is often fraught with problems as many end users get confused about security questions and could easily make mistakes. Implicit consent is therefore another alternative that should be explored. Such implicit consent could be made in an automated fashion if:

1. The end user has previously agreed to allow such actions, in a more general form or for this case specifically
2. The administrator in charge of managing a device, such as the end user's employer, has issued policy that provides implicit consent for this type of operation
3. The end user has provided general guidance through control panel actions to allow software of a certain class or type free reign, or permission to perform this type of operation
4. The device policy coupled with trust associated with software publisher's or application is sufficient to provide implicit consent for operation

Disclosure and consent should also be considered at a technical level. The software itself can disclose intent through embedded manifests. An application can contain a signed manifest that discloses the capabilities the applications intends to use. The device security system can inspect the manifest evidence and decide if and where the application can run. Device security prevention and enforcement infrastructure can inspect policy and the application's manifest to determine if and where the application software can run. If the application contains a manifest that expresses intent the device can make a decision based on configured policy and ask end user's input when needed.

Enforcement

The run-time enforcement architecture provides the most important and often the weakest element of the end to end security story. History shows that if malicious software makes it onto a machine or device it often is able to break out of its enclosure to take control. Least privilege and isolation are two important security principles that if incorporated can help the scale tip in the other direction. A software module that is loaded to execute should ideally be loaded into an execution environment where only requested and assigned capabilities are present. If a software application only needs the ability to visualize information received from a web service the application should run in an execution environment where only network and display capabilities are present. This will satisfy the least privilege requirements. Isolation is achieved when a software solution is isolated from other software solutions at

run-time. The isolation will ensure that a malicious application can do minimal damage to other concurrently running software.

Correction

"Block lists", "allow lists" and revocation mechanisms are common correction mechanisms on mobile devices. Applications on the block list, or those not on the allow list, are prevented from running. Revocation provides a secure method of preventing software from running.

Implementations

Implementers should be afforded enough flexibility to enable trustworthy user experiences that are consistent with their client platform needs. For example, the disclosure and consent for device access within a Web/Widget context would ideally follow the native platform's user experience to help provide a trustworthy user experience. The set of standards would provide a shared foundation as described above while still enabling user experience integration and differentiation of the implementation.

At the same time, it is critical that the tactics involved in describing/invoking a security framework be done in a manner consistent with the Web programming paradigms. The adoption and success of a security framework in this space is gated on developers feeling comfortable and not over-burdened while building experience that leverage these frameworks.

Standards

In summary, we believe standards are needed for the following areas:

1. Code Identity
2. Declarative Self-Disclosure of Security Capability Needs
3. Disclosure and Discovery of device capabilities
4. Risk assessment criteria
5. Risk level definitions and symbols
6. Risk Mitigation Approaches & Quality Standards