Proposal for Application Layer Security Solution

Introduction

Prior Knowledge

This proposal requires prior knowledge of the OSI network model, as defined in ISO 7498 and ISO, and TCP/IP (Cerf, Dalal, & Sunshine, 1974) network model, as defined in RFC 1122 (Braden, 1989). This document references these network architectures to establish a similar architectural model with regard to a specific layer of those architectures. This proposal also requires familiarity with URI syntax, as defined in RFC 3986 (Berners-Lee, Fielding, & Masinter, 2005), and knowledge of IRI features, as defined in RFC 3987 (Duerst & Suignard, 2005), since this proposal requires use of those technology definitions.

This proposal requires some conceptual familiarity of a session ID. A session ID is an identifier attached to a data packet so the consuming application will know which data on the wire to process and which data to ignore. An example is a browser with multiple open tabs. Received data is processed separately in each tab and never does data requested from one tab enter the processing of another tab. Each tab only processes data with a specific session ID and ignores all other data from the network interface. This precise separation between different applications, and even separation within a single application, is the purpose of session data in a layered architecture, or specifically the session layer of the OSI model.

Standards are a Mess

Without the presence of architecture to dictate how a body or system of technology should operate the standards upon such technology have little value upon the industry they service. Successful standards are specific in language and purpose to a single body of technology (Bradley & Byrd, 2007). Standards exist to dictate a presence of conformant operation, or consistency (World Wide Web Consortium), for the specific technology body between different vendors and possibly even different means of consumption by a higher order technology body. Strong standards do not exist to define execution of a technological function, because the work then becomes a technology itself and not a standard upon such a technology. In other words, standards exist to allow interchange of automated features between vendors, and even possibly between other technologies, but are not the technology they describe. When standards become confused upon the nature of their existence, such as actually defining a technological execution in excess of conformance, they tend to become more costly to accept into an automated process and tend to become a terminal work (James, 2008). When a standard becomes a terminal work future standards that are not so limited cannot consume the terminal standard without becoming a terminal work themselves. Standards evident as terminal works are weak standards, and yet may represent strong technology.

The application layer of networking contains thousands of such standards from several various standards bodies. Such standardizing bodies may include: International Standards Organization (ISO), Internet Engineering Task Force (IETF), World Wide Web Consortium (W3C), European Computer

Manufacturing Association (ECMA), Oasis, Institute of Electrical and Electronics Engineers (IEEE), and the list goes on into ad nauseam. Some of these organizations have published thousands of standards with different versions and updates to each. Very few people, even after decades of dedicated study and authoring, can describe the range of available standards with any degree of clarity, and perhaps nobody can fully address the nature of relationship between each of those standards. Standards in the application layer of networking are a mess.

Before a standard is published by a standards body it must pass through a lengthy and often challenging peer review process to ensure the standard addresses a subject of value and the changes it imposes are not harmful to the nature of networking or consuming processes. Once a standard is accepted and approved it may not be immediately published. A necessary delay may be imposed by the publishing standards body to prevent disruption with current events or conflict with other standards currently in development. When the standard is published it is not immediately consumed by the intended audience. The intended audience must run tests themselves to ensure the standard can be integrated accurately and fully without conflict to current method of operation. Even after a standard is consumed by the intended audience it may be years before the desired changes or protections are reflected upon the intended field of technology. End users tend to delay development towards new standards until an acceptable percentage of their market is capable of consuming or processing that standard independent of particular vendors. In other words, it takes years, possibly a decade or longer, for a standard to be fully integrated into the body of technology it is written to standardize. Since it takes so very long for a standard to become standard practice standards are rarely, if ever, retired. The only common method of retiring a standard is to replace it with either updated language or a revision. This means that once a standard enters an industry it is there to stay until all persons who remember the standard retire regardless of the barriers to innovation provided by a potentially weak standard.

Thousands of standards are published from many several standards bodies, sometimes in conflict to standards from other standards bodies (Severance, 1998) (Kostelnick, 1998). Some of those standards are confused to the nature of their existence and thus create a barrier to the nature of standardization. Because there are so many standards it is possible that nobody knows how they are all intended to operate together. Once a standard enters the market it is there almost forever. This is a hopeless state of technology complexity and a defeat of innovation, but it does not have to be that way.

The chaos mentioned above may describe a variety of technology or industry fields, but it certainly applies to the application layer of networking directly. Other fields of innovation do not seem so burdened and overwhelmed by a plethora of possibly conflicting standards, because many other fields of development and industry are governed by models of separation. The natural procession of industrialization then is to create standards to provide conformance and when the quantity of standards or the relationships between standards becomes too complex create a model of separation to govern the standards.

Models of separation exist under various names, such as: taxonomy, body of knowledge, domains, architecture, outline, or doctrine. An unintended benefit of models of separation is the scope definition upon which a standard applies amongst a body of standards and how those relationships are defined. This benefit is important as it limits harm from weak terminal standards, where that harm might otherwise be not evident or so limited. This paper provides a model of separation to the application layer of networking where such a separation does not prior exist, and refers to such a model as a layered architecture with the name Network Application Model. This paper takes the position that the absence of such architecture has allowed an environment of somewhat isolated ad hoc solutions whose presence has produced an environment where technologies' relationships to each other is in conflict to secure operation without violation to the intended operation of those technologies.

Layering Considered Harmful

Section 3 of RFC 3934 (Wasserman, 2004) is titled, 'Layering Consider Harmful', which makes the point that strict layering of network data is harmful to efficiency of processing and is disruptive to context of the processing of data in whole. This paper agrees with the second point made if the layering of information relevant to network processing consists of removing data from the whole apart from individual processing of information for an information body where that information retains integrity apart from fragmentation. This paper disagrees with the second point when the whole of data remains available to derive context even in accordance with strict layered processing. The point regarding efficiency suggests each prior layer of processing is necessary before processing from the next layer in the architectural stack and that such fragmented processing is disruptive to efficiency apart from a single processing of the data in whole. This paper disagrees with this position as processing conflicts that arise from deregulation of a layered processing scheme are disruptive and costly, with regards to security consideration and network data collisions, more so than the excessive processing required by such regulation. The disruption and resultant faults are the effectuality of ad hoc development absent the limitations of a structured conformance. This paper makes the case that an unregulated application environment is more costly and disruptive, due to the volume of security incidents in vulnerabilities and cost of breaches, than the implementation and conformance to a regulated architecture.

The Threat

The most significant threat to security of information is primarily the result of three weaknesses resident in the application layer of networking. Weaknesses in the application layer of network architectures account for almost every annually reported vulnerability with four of the top five reported vulnerabilities reported for 2009 existing as a form of client-side scripting from web browsers (Symantec Corporation, 2010). The web is the most vulnerable software interface ever created and so it is the most frequently compromised, however worms propagated through email may account for more severe and rapidly spreading infections. The web is so vulnerable that during 2008 more than 13% of 12,000 tested websites can be compromised completely automatically and 49% contain critical vulnerabilities (Gordeychik, et al., 2010). Many distributed botnet systems are managed from application interfaces present in Internet Relay Chat (IRC) (Symantec Corporation, 2010). No application protocol is to blame in isolation. The problem is a lack of regulation upon application execution in a networked environment.

The public internet is 30 years old and the web is 20 years old (Howe, 2010). No solution to this security problem has ever been proposed. Further to the point, there is no security standard for application processing across a multiple networked parties using the web or any other sort of general web security standard. The latest form of HTML, HTML 5, only intends to amplify the quantity and severity of the problem through integration of scripted dynamic media interfaces, more reliance upon client-side scripting to promote an application environment instead of an application layer to a networked environment, and scripted access to static media data (Hickson, 2010). This proposal makes the claim that if necessary regulations are supplied for application execution in a networked environment a significant majority of the current threat would be incapable of existing.

With the severity of acceleration of security compromises year over year, especially with consideration for attacks intended to mitigate a trusted and encrypted network session, the application layer cannot be considered a trusted environment without the necessary regulation. A trusted environment is absolutely required for commerce exchange, transmission of personally identifiable information, a networked application interface (cloud computing), or data storage. A solution must be proposed and executed or the internet, by result, must be considered a harmful environment where all data theft and application compromise is the legally acceptable result of an otherwise not trusted environment. This document intends to propose a solution and not a mitigation to the security problem. An application cannot hope to become secure until it is changed to operate securely, and that is not occurring on the Internet's application layer as described previously with HTML 5 (Hickson, 2010).

Of these three weaknesses the most dangerous is by means of direct execution of malicious code, or media, often through social engineering means. 49% of web based attacks in 2009 were the result of malicious PDF media containing client-side scripting (Symantec Corporation, 2010). The most significant quantity of attacks is by means of client-side scripting execution (Symantec Corporation, 2010). The third weakness is unauthorized automated resolution of stored addressable data, which allows malicious code to replicate over a network.

Introduction to Client-side Scripting

Client-side scripting is any means of application execution where that execution is resident upon data supplied to a user or code executed upon data separate from the compiled application performing that execution. Client-side scripting can occur in various different manners. Client-side scripting is programmatic code that is written by remote parties to be executed by a separate user-agent application without means to verify integrity or authenticity. Since such programming code is written by strangers and silently executed by visiting consumers it may result in an attack that is immediate, massively distributed, always effective, and largely unknown to both the executing party and the distributing party.

JavaScript

The most frequent form of client-side scripting is JavaScript language from the World Wide Web. JavaScript is a unique language based upon a lambda model of inheritance and a unique form of object-orientation. This language is supplied in plain text characters either directly in the code of a webpage, or as separate files requested by a webpage. Nearly every form of dynamic or interactive event or response that occurs in a webpage is a result of JavaScript execution in reflected as an action or interactive condition supplied by a user, referred to as an event.

JavaScript can also open a separate communication, using the XMLHttpResponse programming object, and download additional executable code or send data without notifying a user. This form of separated communication is frequently referred to as Asynchronous JavaScript and XML, or AJAX. If the AJAX is a part of malicious JavaScript supplied to a user it can be used to send secure personal data from a user to a third party without regard for encryption between a user and a web server. Malicious AJAX can supply false content data to defame the source content requested by the web user. JavaScript does have a native security model often referred to as the same origin policy that prevents JavaScript from reading into an area of a browser window served from a separate scheme and authentication part of a URI string. The same origin policy protects the leaking of private data into malicious advertisements displaying in an iframe HTML object, for example, but does not prevent AJAX from sending data to separate domain.

Flash/Actionscript

Flash is an interactive vector animation media with supplied scripting code compiled to a unique binary data file that is executed by Flash Player software. Since the script code is compiled into the intended data/media it cannot be altered after the fact. The script code, known as Actionscript, is a newer form of JavaScript that executes from the Flash Player in response to user events mimicking a native application effect.

Malicious Actionscript can be written to compromise software vulnerabilities in the Flash Player and any possibly software executing the Flash Player. Such attacks against the execution of compiled applications from user supplied scripting code are typically referred to as memory corruption attacks. In these types of attacks a user will execute a Flash media file with malicious code that interrupts the memory address space assigned from the operating system software to the Flash Player application. This form of attack targets a known vulnerability in the Flash Player software that allows the malicious code author to compromise the operating system software or execute malicious code outside the Flash Player software. If the Flash Player software is compromised when executing from a web browser software the malicious coder can also execute code against known vulnerabilities in the containing web browser.

Acrobat/PDF

The vulnerabilities in Adobe Acrobat software are nearly identical to those described in the Flash Player software since a PDF file can become interactive if supplied with client-side code that is executed by the Adobe Acrobat software.

ActiveX

ActiveX is a client-side scripting language invented by Microsoft that executes in a manner similar to JavaScript, but can also access programmatic features of Microsoft's web browser, Internet Explorer, and execute limited features of the graphic user interface and file system, known as Explorer, of Microsoft's operating system. ActiveX features all the same vulnerabilities as JavaScript. If it is able to execute the XMLHttpRequest object then it has the same vulnerabilities as AJAX. Since ActiveX has access to software outside Internet Explorer it can also execute vulnerabilities outside Internet Explorer and violate privacy outside Internet Explorer. In 2009 ActiveX vulnerabilities accounted for 3 of the top 10 web attacks (Symantec Corporation, 2010).

The Solution for Client-Side Scripting

Client-side scripting must be abandoned in all forms. Acceptable programmatic application code is either executed directly against the operating system software directly, or entirely is absent a network interface or network processing software. When programmatic code exists outside those two conditions it must be considered potentially malicious, and therefore not safe. This implies all client-side scripting must be considered potentially malicious even if it is supplied from a trusted source, or supplied as compiled code supplied with media, or is a function of an interactive means.

The Rule

The solution to this problem of client-side scripting is to create a layered application model by which data and software intended for exchange over a network are fundamentally separated. **In a networked environment data and programmatic code, intended for evaluation or execution in a programmatic manner, must not integrate into a single source, ever, on any level.** Data is information that exists in a static state and does not supply logic for evaluation. Programmatic code is any means of supplying logic for evaluation by a computer. When this rule is violated the data is always open to attack by a malicious means in a networked environment and the user is always open to compromise when accessing their data. The means to achieving the solution is to create an application environment that does not contain client-side scripting and accomplishes the interaction and usability objectives inspired by client-side scripting.

The rule must apply to all application processing units receiving data or transmitting data across a network. This means every processing unit regardless of status as client, server, or something else. In 2008 52% of web vulnerabilities occurred on the server side while 48% occurred on the client side (Gordeychik, et al., 2010).

Data received from a network interface that refers to programmatic code to be delivered across a network interface is considered a violation of the separation rule even if the code referenced by the data is a separate file. Even if the programmatic code is separated from the data into separate files the transfer of that programmatic code is the result of a request for data and not a request for programmatic code. Since, in this case, the request for data cannot be separated from the request for programmatic code the separation rule is violated. Data received from a network interface must not be able to request programmatic code unless the requested programmatic code is locally available and resident upon the local processing computational device. If received data does make a request for programmatic code across a network interface that request must be ignored.

Programmatic code received from a network interface must, likewise, not be able to request data unless that data is locally resident upon the local processing computational device. This rule applied to programmatic code would prevent any application software from serving as a protocol processing application, and so exceptions to this rule must be allowed. Valid exceptions must become a vehicle to violate or bend the rule of separation. To prevent such violation an exception to separation as it applies to requests from programmatic code must adhere to these conditions:

1. The programmatic application must not be bundled with data intended for distribution across a network. A programmatic application in violation of this rule must be presumed malicious.
2. The application must prompt the user and receive authorization from the user that it seeks to send and/or receive data from a network. This prompt and authorization is not required to occur more than once per application protocol.
3. If the application is received from a network interface the application, as part of the installation process into the operating system or first time execution, would verify its integrity by hash comparison between a generated hash of itself and an unpublished hash value at an authorized publisher that is verified by the executing user.

The Need for a Commonly Shared Structure

If there exists a structure of data requirements for data resident in the OSI application layer the desired effect of client-side scripting can be achieved in an application locally installed opposed to temporarily transmitted code that is not trusted. An exemplification of such a structure is to divide the application layer into a layered model called the Network Application Model.

This model features 7 layers, where the most foundational layer is at the bottom and each rising layer is progressively lesser foundational, that define the processing requirements and resolution of transmitted application data. Each layer has a specific purpose and is required to be separated from each other layer. It is this uniform separation that makes the model functionally successful. Data is required, according to each layer's definition, at each layer. If a layer does not contain the required data the request must be considered corrupt and a user-agent software may request the data again or abandon the communication altogether. This model is to apply to all applications processing data from a distant party including service applications not affiliated with an end user.

The Network Application Model

7. Audio/Visual
6. Data
5. Type
4. Address
3. Persistence
2. State
1. Protocol

The intended objective of this model is to define the precise application requirements of data so that the engaged session is not inherently bound to the context of a single application protocol. This means a requesting entity may propose a network communication with a given session ID and change communication from the current protocol to another.

Consequently, no more than one application must be actively assigned to a single session ID at any given time to avoid potential data processing collisions from a transmission coupling effect. If application data is sent to an application that is already assigned a session ID the application must reject the data unless the application prompts a user to make a choice. This limitation is significantly important with regard to an inactive state as defined below in the state layer.

When a protocol that offers encryption is exchanged for a protocol that does not offer encryption, such as HTTPS to HTTP, the encryption is terminated, and that termination must be complete and permanent for the provided session. Encryption must not be resumed automatically upon resuming use of the protocol offering encryption. In transition from an unencrypted protocol to an encrypted protocol the requestor must always be challenged by the replying entity. If no challenge is provided from the replying entity to the requestor upon changing to an encrypted protocol the transmission must be presumed hostile and abandoned by the requestor's processing application. Such a challenge must force a response from the impacted user that is not automatically or application generated. This process must occur at every instantiation of an encrypted transmission, or authentication must be denied by the replying entity with presumption that requestor is a malicious agent.

First Example

An example would be a user communicating to another user, by means of a proprietary instant messaging service, could propose to communicate using a separated proprietary cloud based software under the context of the same session ID so that when this second form of communication is complete both users respectfully return to their instant message communication without interruption. In this example a user who needed to overcome the finite limitations of one communication protocol without breaking session was able to change the protocol to execute a separate application and, once completed, return to the instant message software as though there is no interruption to the engaged conversation or application limitation preventing that engagement.

Second Example

Another example is a user wishing to navigate to an online map service from the web. In this case the user would navigate the web using the HTTP protocol until the user arrived at the desired location. Once at the desired location the user would need to process a map media and then supply input and request changes to output. Currently, the task of supplying additional input to achieve more specific output is the result of executing client-side scripting upon a tool set to allow zooming and panning around a visual map. The removal of client-side scripting means the user no longer supplies changes to the map through the context of the web page. A separate application not reliant upon client-side scripting must be used and can be used in the context of a webpage so long as the programmatic code is resident upon the application and only data is transmitted by execution of that application.

Another solution to the prior example is software that universally accepts vendor specific application programming interface (API) instruction modules to interface between a user's interaction and a dynamic request for data at the vendor. This solution would be valid, in accordance with the Network Application Model, so long as it never couples data with the API instructions. To ensure the application is safe and not compromised software the API should contain a hash value that it sends to the remote destination along with each data request where the data request is authenticated. Upon authentication of the hash value the requested data is returned to the requestor. The generic software accepting the API should deny integration of any API for which a hash value cannot be verified at the trusted destination. Such a mechanism would make providing spoofed software/interfaces more difficult, and if the hash value is compiled to byte code from the originating source it would not be open to malicious alteration. Such a process should resemble certificate authentication to validate encryption of a trusted HTTPS connection.

Third Example

Another example is where a user clicks a button in a form on a HTML web page and a calendar appears to assist a user in making date decisions or even a decision for a span of dates. Client-side scripting would likely generate the calendar referencing the current date. What is defined as the number of days in a month and where in the week the first day of the month must appear with regard to a fixed reference point in the client-side scripting code based upon a known static calendar. The current date referenced by the client-side scripting comes from interpretation of the processing application, which references the date from the operating system, which references the date from date stored on the resident computer's hardware CMOS. Even if there exists a running application service to regularly synchronize date and time information from a remote network source the service likely only supplies changes to the date and time settings saved in the CMOS via the operating system and not dynamically supply this data to applications executing on the operating system in conflict of the data stored on the CMOS. Since everything on the computer receives the date and time data from the same single source the only reason to process any client-side scripting is to provide programmatic instruction to access the date and to generate a custom calendar application. The consuming application is capable of providing a calendar using a similar code processing means, which only leaves the instruction to dynamically access a dynamic calendar application apart from an otherwise static document.

This calendar problem can be solved with various methods not requiring client-side scripting. One possible solution is to supply a named element that is recognized by the consuming application and returns the desired instruction. Another solution is to supply a known keyword to the consuming application where that key is defined using either a syntax means and/or a vocabulary to a processing API compiled into the consuming application.

Network Application Model – Processing Syntax

Character Set

The character data stored and transmitted at a Network Application Model must be encoded as UTF8 characters. Processing and conversion of UTF8 characters to other encoding schemes must match the conformance required by the IRI specification defined in RFC 3987 (Duerst & Suignard, 2005).

Layer Name

The lowercase English name of each layer must precede the contents of each layer and be wrapped in triple colons. The layer name must be preceded by and followed by a single paragraph separator character (U+2029).

Network Application Layer Encapsulation

The first characters of the Network Application Model must be ">#" and the final characters must be a paragraph separator character (U+2029) followed by "#<".

White Space Minimization

All white space must be tokenized prior to transmission. Tokenized white space is where all white space characters are converted to a space character and all consecutive spaces after the first in a series of consecutive spaces must be removed.

Syntax Failure Conditions

- If a layer name is not supplied or is not properly formatted processing must fail.
- If the ">#" start sequence is not present processing must fail.
- Termination of payload delivery for a TCP based protocol is fixed at a known point. For data services transmitted over UDP based protocols only the first fully received data packet requires the complete Network Application Model. If the terminating character sequence, paragraph separator character (U+2029) and "#<", is not present upon completion of the delivered payload processing must fail.
- Syntax verification must occur before any data is interpreted for execution.
- The contents of each layer must be interpreted as text data only and never executed directly as application instructions to prevent code injection.

Network Application Model – Example

```
>#
:::protocol:::
http
:::state:::
static
:::persistence:::
]persistence[none]/persistence[
:::address:::
http://example.com/webpage.html
:::type:::
application/xhtml+xml;iso88591
:::data:::
]dataLayer[web page HTML code]/dataLayer[
:::audio/visual:::
relativepath/stylesheet.css
relativepath/second.css
#<
```

Layers of the Network Application Model

Protocol Layer of the Network Application Model

The protocol layer contains the protocol of the application service. Some examples include HTTP, SNMP, IRC, XMPP, and FTP. The protocol name must be stated explicitly in this layer and not presumed from the request by the requesting application or from the IP port. The only valid value must be the name of the protocol in lowercase Latin alpha or Latin numbers characters. If the value supplied in this layer contains characters of another type the data request must be deemed corrupt. If the network data transaction is the result of a user request, such as a HTTP "get", the protocol must be explicitly defined by the requesting application. If the protocol is the result of a data exchange not requested by a user, such as the push nature of SMTP, the protocol must be stated in the application data header. Decisions upon execution or delivery of the session ID to separate conforming application data arriving to a single network node from a network interface must be decided before interpretation of the next Network Application Model layer.

State Layer of the Network Application Model

The state layer contains a keyword to designate the expected duration of a data session, as known to applications by virtue of a session ID. This layer allows an application to make decisions about session congestion to improve processing efficiency upon data received from multiple continuous separate application requests from a single network interface. If the state data is not supplied by the remote source the value must be “static”. The value for the state layer must be processed prior to processing of the next layer in the Network Application Model. This layer may receive one of these following values:

- static
- active
- inactive

The value static explicitly states that the limit of the data session is limited to the transmission of requested files only and upon delivery of the requested file the session is destroyed without regard for processing decisions supplied by the application.

The value active determines the data session expects to continually receive data and must continually remain open and available, such as a UDP stream or IRC connection. An active state is necessary for the processing of streaming media, such as radio or television, where the data is supplied continuously and without end. An active state may be terminated from a remote source by disconnecting the communication on the remote end, or may be terminated by the processing application upon user request or in conformance to a user supplied or application supplied configuration. All communications in an active state must be terminated prior to the processing termination of the processing application and should be terminated by the operating system software if the active state remains open after the processing application has terminated in case of an improper termination.

The value inactive determines the processing application is not continuously receiving data, but should expect data to be supplied to the designated session ID automatically and process the data upon receipt. An example of such data is news updates, where the news is frequently updated but such updates are not continuous. In an inactive state the processing application should expect to reserve the session ID and not release it even upon the termination of a data stream. An inactive state must be locally terminated in the same manners defined for the active state.

In an inactive state a network session can be terminated remotely and remain open locally so as to conserve network resources of the remote party that can be recycled and open a new connection only when it is ready to transmit data. It should be noted that a remote source cannot retain a session ID if the session is remotely terminated. This suggests that all actively running processing applications will receive the data for processing and will process that data if so capable. Multi-application processing of a single communication can be regulated through definitions supplied to the persistence layer and by denial from applications already assigned a session ID.

Persistence Layer of the Network Application Model

The persistence layer provides a means for a processed communication to be remembered for future communication sessions. There is no formal definition for data supplied in this layer as the data may be custom to the responding entity of the communication. The contents of this layer must begin with the character sequence “]persistence[“ and end with “]/persistence[“ to prevent the custom contents of this layer from conflicting with the syntax of the Network Application Model. A value is required for this layer with the default value being “none”.

Since it is the responding entity that determines the persistence value the originating value must always be “none”. Applications should provide an option to allow users to disable persistence for a particular responding entity and/or all responding entities of a given application protocol. An example of data supplied in this layer would be a web cookie that contains data of user setting details. Passwords and personally identifiable information should never be supplied as part of a value in this layer unless that data is first hashed by a hash algorithm and not identified as a password or other authentication details. If this layer does not contain a value or the starting character sequence or the ending character sequence then the data must be considered corrupt as all three parts are required. The value supplied in the persistence layer must be processed prior to interpretation of the next layer.

Address Layer of the Network Application Model

The address layer defines a globally unique address of the requested resource. The syntax for this value in a global space must follow the syntax requirements of URI, RFC 3896, or follow the syntax requirements of IRI, RFC 3897. The syntax for this value in a local space must follow the File URI/IRI scheme. This value must be processed prior to interpretation of the next layer. The strictest conformance to the syntax requirements of the prior specified definitions must be adhered to in order to prevent code injection to a URI/IRI string or the hiding of such through encoded character sequences. Since strict conformance to syntax is required for safer interoperation violations upon those syntax definitions must result in corruption and fail to process. Absolute address values must always be supplied in the global space as relative address values cannot be resolved individually.

For secure operation a proprietary syntax may exist in a private network application environment. A private environment implies a proprietary protocol and software specifically written to process that protocol specifically at all points. That implies a proprietary processing application at every user and every processing server interface on the network and access to the specific protocol does not exist outside the proprietary software.

Type Layer of the Network Application Model

The type layer defines the media type of the resource and the encoding scheme representing the character encoding of the data. The syntax for this layer would be a mime major type separated only by a forward slash, “/”, from a mime subtype. The mime type and encoding scheme are separated only by a semicolon character, “;”. All alpha character data in the layer must be lowercase and spaces must not exist. All hyphens in the character encoding value must be removed. Malformed data in this layer should fail processing and result in a failed data request. This layer must be processed before interpretation of the next layer. An example:

application/xhtml+xml;iso88597

Data Layer of the Network Application Model

The data layer contains the human consumable content and/or data structure. Formats that would exist in this layer are the contents of web pages, XML documents, JSON documents, plain text, spreadsheets, and any other data that humans would read. The contents of this layer must first begin with "]dataLayer[" and terminate with "]/dataLayer[". Those two character sequences are necessary to alert processing applications exactly where the contents of the data layer begin and end before continued processing of the Network Application Model. This encapsulation is necessary so that syntax conventions defined for the Network Application Model are not in conflict with similar syntax conventions expressed in the contents of the data layer.

This layer must be processed according to the definitions supplied in the type layer after consideration for the layer's start and end character sequences specified in the prior paragraph. If the type layer specifies a media type that is incompatible with the contents of the data layer then processing must fail. If the type layer specifies a media type that matches the intended processing for the content of the data layer, but the contents of the data layer are incompatible with the protocol specified in the protocol layer then processing must fail. This type convention is necessary to prevent data from appearing in one form and executing in a different form unknown to the user. If the contents of this layer do not contain the start character sequence, some amount of character data, or an end character sequence the data must be presumed corrupt and must fail. This layer must be processed prior to processing of the next layer.

Audio/Visual Layer of the Network Application Model

The audio/visual layer contains assisting formatting instructions for the data comprised in the data layer in accordance with instructions defined in header data or structured meta-data conventions resident in the data layer. This layer commonly represents a convention of presentation. Another name was chosen to remove ambiguity between this content and the presentation layer of the OSI model. This layer also represents more than mere presentation as it can accept reference to audio signals for closed captioning and other accessibility benefits.

The syntax for this layer must be addresses of the required assisting files. The addresses can be relative addresses to the information contained in the address layer and may be formatted as either a URI or IRI. The most strict syntax conformance to the definitions supplied in RFC 3986 for URI or RFC 3987 for IRI must be adhered to in order to prevent code injection on an address string or a hidden injection through encoded character sequences. Any address that cannot be resolved must be ignored so that corruption to information in this layer degrades gracefully without rejection to processing of the data. Each address must be separated by a single paragraph separator character (U+2029). If the data is not assisted by visual or audio processing instructions then this layer must be supplied a value of "none".

Malicious Files and Media

Malicious files can be sent to users in email or downloaded by users across the web. Malicious files can come in absolutely any form so long as the executing software is able to perform the malicious intent of the file's creator.

Files can be supplied with a naming convention that masks the named file extension upon the file. A file extension is period separated group of alpha-numeric characters at the end of a file name. In Windows operating systems the extension is commonly three characters long, but can be more. In a Linux/Unix a file extension is never required if the user directly tells an application to execute a file. File extensions typically identify how a file is supposed to be executed by a consuming application, of which application should execute the file. This is problematic since an operating system can be configured to visibly hide file extensions or a file can be executed in a manner unexpected by the consuming user.

Files, especially media and script files from the web, can be supplied to the user as fragments of an attack. A fragmentation attack occurs where each of the participating files is entirely benign on its own, but when used with other files or data becomes a malicious piece of code executing without the victim's wishes.

Solution to Malicious Files

The only solution to this problem is for every file sent to be sent with a mime type identifier. The mime type describes exactly the type of media. Applications must recognize files by their mime type and not by any other characteristic, such as file extensions. The files must be executed in accordance with their noted mime type only and all supplied client side scripting must be processed as string literal data only. If a file cannot be execute according to its mime type then it is corrupt and must not be processed by any other means. When a file is marked as corrupt the application may notify the user or may request the file again, but the application must not notify the distant end of the corruption since there is no expectation for an application to process such a communication. Either the requesting application or the responding application may drop or block the network connection upon an unspecified number of repeat requests.

Uniformly Encapsulating the Mime Type

The mime type must be delivered in a uniform manner so that applications know exact what to expect and where to expect such a description without malicious intervention. If such a uniform manner deviates from the standard syntax or is absent the file cannot be associated with a mime type that is uniformly described and must therefore be presumed corrupt. It is the responsibility of the application sending a file across a network interface that the uniform encapsulation is present and well formed. The syntax for such an encapsulation is as follows:

1. A start character sequence of a greater than character, a pound character, and a paragraph separator character (U+2029).
2. A right square bracket, the mime type in conformance to the naming methods provided by the Internet Assigned Naming Authority (IANA), a left square bracket, and a paragraph separator character (U+2029).
3. The actual file data.
4. An end character sequence of a paragraph separator character (U+2029), a pound character, and a less than character.

Example:

```
>#
]image/jpeg[
compiled binary data
#<
```

Stored Addressable Data

Stored addressable data is an organized or listed data of network addresses or other network identifiers. Stored addressable data may include web browser bookmarks, web browser history, email contact list, IRC server list, and any other list of addressable contents for networked communication. Simple, predictable, and unauthorized access to stored addressable data allows self replicating malicious software to spread across a network without limit if there nothing specifically blocking such traffic. This is primarily of concern to a specific type of malicious software referred to as 'worms'. Worms commonly spread through email by accessing an email contact list found as resident data in the executing email software, but can easily thread through other means if available. Three of the top ten malicious code families introduced in 2009 were worms (Symantec Corporation, 2010). If stored addressable data is not available worms cannot spread, because there must exist some method available to traverse a network that is open and directions on where to traverse upon that open means.

Stored addressable data must be stored in an encrypted form and must not be decrypted until the data is requested. When the information is decrypted the decrypted data must be stored in volatile memory space only. This means an email client, for example, must not open or decrypt a user's email contact list until the contact list is requested by that user even though the email client application may already be executing. When the user has finished using the addressable data list the list should be closed and destroyed from memory if there are no changes to the list or encrypted and saved. The decrypted data must be saved in volatile memory only.

To ensure such lists are separated from the consuming application such lists must not be affiliated with the same kernel process as the consuming application, but should be a separate subordinate process. Subordination is important, because if the processing application closes unexpectedly then the stored addressable data also crashes.

Malicious Security Problems not solved by this proposal

Security is Not Privacy

A common error in security analysis, by people not specifically experienced in a security profession, is the equivocation of privacy to security. These two values are inherently related, but are not the same, and each has problems separate from the other. A primary principle of security is confidentiality. Confidentiality is the means of a system or a system's manager to prevent unauthorized access. Confidentiality is also the prior established and regulated means of information disclosure, where that disclosure is in accordance to defined internal policies and procedures and in accordance with external laws and regulations. Confidentially is not a classification means and is not a method of integrity. If a communication or data is not deemed confidential then disclosure is unlimited. Since confidentiality is not a method of integrity there is no way to preserve the accuracy of data when that data is available in an unlimited capacity.

This means that privacy is not security. Privacy is a personal limit upon disclosure of any information to anybody where confidentiality is not established. In other words, when security is not available, privacy is all a person or resource has. Since privacy is not security there is no limit or barrier to a person's freedom to violate privacy. The violation of privacy by the owner of that privacy is commonly referred to as self disclosure. In a vacuum where information exists entirely without cost to other information this is not a problem. In this narrow circumstance abandoning privacy impacts little or no harm.

Information rarely exists in a vacuum. When privacy is abandoned the data conveyed may have use unknown to the disclosing party. Unintended uses upon disclosed information may result in means to violate security upon associated data or resources that are secured. Even though privacy is not security and is not limited in the manner that security implies it may allow a means to interrupt security elsewhere.

This paper makes no attempt to solve problems associated with privacy violations.

Privacy Disclosure

This proposal will not keep users from disclosing private data. Users, for example, should not disclose their home address and details when residents will be off the property for any extended duration. Disclosure of private data is an administrative failure. This example is often referred to as 'over sharing' and may result in higher insurance premiums (Orlowski, 2010). There is no solution for individuals who disclose their own private data. When a business or group entity discloses private data of supporting users those users are obligated to seek administrative and financial recourse.

Inside Attacker

An inside attacker is a person or group of persons who are members of an organization and seek to harm that organization, such as an employee launching an attack against the employer's computer network. Such problems can only be solved through proper use of various administrative processes that are both automated and human managed.

Social Engineering

This proposal operates on the 7th layer of the OSI model. It does not operate on the proverbial 8th layer, the user layer. Social engineering is the manipulation of people to achieve a malicious intent or circumvent a security procedure. The only solution to prevent social engineering and phishing attacks is user training and adherence to security policies or common security best practices.

Code Injection

This proposal does not directly address code injection, however the suggestions of this paper concerning code and data separation may significantly impact the propensity of injection based attacks. Data supplied through a network interface must always be scanned by a processing application to ensure syntax characters are not present and not present in an encoded form. In such a case that syntax characters are harmful to the processing application or consuming storage medium the application must escape, transform, or remove those characters until no harm is present to either consuming applications or application processing upon storage mediums. Code injection implies an intentional and malicious violation of the separation of data from programmatic code stated earlier. In 2009 60% of identities exposed were the result of targeted hacking attacks that primarily used code injection attack methods (Symantec Corporation, 2010).

Cross-Site Request Forgery

Cross-site request forgery (CSRF) is where data, anonymously transmitted as a HTTP GET string in the form of a URI to a consuming web server application, is processed by a web server allowing a third party to appear as a user who recently authenticated their session and kept their session open. Such an attack represents two failures in security and potentially a bad practice. One failure is authentication at the application layer only, but that authentication does not take into account network data from any other layer of the OSI model. The result is that the application cannot tell where on a network, or the Internet, a session originated and so any other person able to access that application in a like manner can appear to be somebody known to a processing application. The second failure is the processing of an anonymously supplied string into a session response without verification. All data supplied anonymously to a server with regard to session activity must not be processed from a protocol that is not encrypted unless a challenge/response mechanism is prompted to force the engaging party through an authentication process. In other words all data relative to an application session must first be transmitted through an encrypted means only and the application must validate the encryption against at least two layers in the OSI model for future challenge response for the remaining active state of the application session. The probability of this attack is significantly reduced through incorporation of automatic session timeouts set to a short idle duration.

Passwords in the Clear

This proposal will not keep users from sending authentication data as plain text over the wire. The only solution for such a problem is to prevent access to any protocol that processes plain text data for authentication.

Password Guessing

This proposal does not address password guessing. A month prior to this writing a Swiss security researcher discovered that a brute force mechanism for password guessing using a rainbow table configured for faster response from a solid state hard disk could crack a 14 character password, even with special characters and numbers and capitalization variation, in roughly 6-8 seconds (Leyden, 2010). Password management is a function of proper application design. Do not store hashed passwords in a predictable location and ensure there are limitations upon access to that data, such as compiling the hash values to prevent access to the stored data from unauthorized tools.

Summary

In summary there exists a massive security epidemic associated with networked communications and the majority of that problem can be isolated to application processing of data received from a network interface. The security problem is represented by three problems each with their own solution.

1. The first problem is client-side scripting. This problem can be solved by applying a strict separation between static data and programmatic code. If the separation is violated the data must not be processed. In order for the solution to exist there must be an alternative condition to replace the problematic client-side scripting. The alternative condition is a layered application model to allow protocol interchange to a single session ID.
2. The second problem is malicious software passed across a network. This problem can be solved, in combination with the prior step, by forcing a mime type identifier onto each data file or transferred programmatic code. The transferred file must be identified by and executed in accordance with that mime type identification only. If such execution fails then the file or programmatic code must be considered corrupt.
3. The third problem is propagation of unauthorized software across a network interface. This problem can be solved by eliminating unauthorized access to data that identifies a list of network locations.

This paper takes the position that at the time of this writing no solution to application layer security solution is in public circulation. This solution provided by this paper may not be the most correct solution, however if there exists no other solution the solution provided herein then becomes the most correct solution. If the solution provided herein is thus the most correct solution then the solution must be supported or adoption or those of dissenting positions must concede security is either irrelevant or not of value.

Persons at Risk

The solution proposed by this paper expects to solve and eliminate a significant majority of security problems associated with networking and the Internet. This paper also provides a means of organization by which technologies and standards may be organized against. This paper does not, however, make any attempt to solve the human factor. Persons lacking education of basic security awareness put themselves at risk for exploitation or manipulation. In this regard the problem is not a technology problem and a technology solution may be at best impractical and at worst harmful or violating. It is not reasonable to expect privacy to be preserved if a technology means is provided to solve a problem of people's abuse of their security through privacy disclosure.

Sources

Berners-Lee, T., Fielding, R., & Masinter, L. (2005, January). *Uniform Resource Identifer (URI): Generic Syntax.* Retrieved January 1, 2010, from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc3986.txt

Braden, R. (1989, October). *Requirements for Internet Hosts -- Communication Layers.* Retrieved April 6, 2010, from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc1122.txt

Bradley, R., & Byrd, T. (2007, January). *Information technology architecture as a competitive advantage-yielding resource: a theoretical perspective.* Retrieved April 22, 2010, from Association for Computing Machinery: http://portal.acm.org/citation.cfm?id=1359765

Cerf, V., Dalal, Y., & Sunshine, C. (1974, December). *SPECIFICATION OF INTERNET TRANSMISSION CONTROL PROGRAM.* Retrieved April 6, 2010, from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc675.txt

Duerst, M., & Suignard, M. (2005, January). *Internationalized Resource Identifiers (IRIs).* Retrieved January 1, 2010, from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc3987.txt

Gordeychik, S., Grossman, J., Khera, M., Lantinga, M., Murray, C., Wysopal, C., et al. (2010, January 10). *Web Application Security Statistics.* Retrieved April 23, 2010, from The Web Application Security Consortium: http://projects.webappsec.org/Web-Application-Security-Statistics

Hickson, I. (2010, April 20). *HTML 5*. Retrieved April 23, 2010, from World Wide Web Consortium: http://dev.w3.org/html5/spec/Overview.html

Howe, W. (2010, March 24). *A Brief History of the Internet*. Retrieved April 23, 2010, from Walt Howe's Internet Learning Center: http://www.walthowe.com/navnet/history.html

James, J. (2008, August 27). *HTML 5 Editor Ian Hickson discusses features, pain points, adoption rate, and more.* Retrieved April 22, 2010, from TechRepublic: http://blogs.techrepublic.com.com/programming-and-development/?p=718

Kostelnick, C. (1998, November). *EJ581313 - Conflicting Standards for Designing Data Displays: Following, Flouting, and Reconciling Them.* Retrieved April 22, 2010, from Education Resources Information Center: http://www.eric.ed.gov/ERICWebPortal/custom/portlets/recordDetails/detailmini.jsp?_nfpb=true&_&ERICExtSearch_SearchValue_0=EJ581313&ERICExtSearch_SearchType_0=no&accno=EJ581313

Leyden, J. (2010, March 12). *SSD tools crack passwords 100 times faster*. Retrieved April 23, 2010, from The Register: http://www.theregister.co.uk/2010/03/12/password_cracking_on_crack/

Orlowski, A. (2010, February 22). *Web2.0rhea means 'higher insurance premiums'*. Retrieved April 23, 2010, from The Register: http://www.theregister.co.uk/2010/02/22/web20rrhea_insurance/

Severance, C. (1998, January). *Conflict and Consensus: The Role of Standards.* Retrieved April 22, 2010, from Dr. Chuck's Interactive Personal Portfolio: http://www.dr-chuck.com/dr-chuck/papers/columns/r1138.pdf

Symantec Corporation. (2010). *Symantec Global Internet Security Threat Report - Trends 2009.* Mountain View: Symantec Corporation.

Wasserman, M. (2004, October). *RFC3934 - Updates to RFC 2418 Regarding the Management of IETF.* Retrieved April 22, 2010, from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc3987.txt

World Wide Web Consortium. (n.d.). *About W3C Standards*. Retrieved April 22, 2010, from World Wide Web Consortium: http://www.w3.org/standards/about.html